NAME

intro — introduction to acorn32 special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

DEVICE SUPPORT

This section describes the hardware supported on the acorn32 (RiscPC) platform. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4)

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

vidcvideo	VIDC20 device interface
com	NS8250-, NS16450-, and NS16550-based asynchronous serial communications device interface
ea	Ether3/Ether5 podule device interface.
eb	EtherB network slot device interface.
eh	EtherH network slot device interface.
ie	Ether1 podule device interface.
lpt	Parallel port device interface
mem	Main memory interface
wd	Standard Internal RiscPC IDE device interface
fd	Standard Internal RiscPC floppy device interface
asc	Acorn SCSI I card device interface
csc	Cumana SCSI II card device interface
ptsc	Powertec SCSI II card device interface
oak	Oak SCSI I card device interface

HISTORY

The acorn32 **intro** appeared in NetBSD 1.2.

NAME

intro — introduction to alpha special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

DEVICE SUPPORT

This section describes the hardware supported by NetBSD/alpha. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4)

HARDWARE

DEC and Compaq have produced a series of the Alpha CPU, some of which are listed below, along with some systems which contain them.

The NetBSD Project distributes binary programs for its Alpha port compiled for the lowest common denominator CPU instruction set, to guarantee binary compatibility across all supported Alpha systems. However, it is possible to sacrifice binary compatibility for additional performance on later model CPUs with performance enhancing instructions (e.g. the 21164-A and later with the BWX extensions). This requires recompiling from source code, with appropriate options given to cc(1) to indicate the target CPU.

"EV" stands for "Extended VAX" (or "Electro Vlassic") and the number following is a reference to the CMOS process used to make the chips. "LCA" stands for Low Cost Alpha, and "PCA" stands for PC-architecture Alpha.

21064 **EV4** (100-200 MHz, 0.75 micron)

AlphaPC 64 (EB64)

Jensen family
- DECpc AXP 150 (Jensen)
- DEC 2000/300 (Jensen)
- DEC 2000/500 (Culzen)

Avanti family
Digital's lower-end PCI-based workstations.

AlphaStation 200 4/100-166 (Mustang)
AlphaStation 400 4/166 (Chinet)

Sable family
AlphaServer 2000 4/200 (Demi-Sable)
AlphaServer 2100 4/200 (Sable)

Pelican family
Low-end TURBOchannel based workstations.

DEC 3000/300 (150 MHz) (Pelican)
DEC 3000/300X (175 MHz) (Pelican+)
DEC 3000/300L (100 MHz) (Pelica)
DEC 3000/300LX (125 MHz) (Pelica+)

Sandpiper family
High-end TURBOchannel based workstations.

DEC 3000/400 (133 MHz) (Sandpiper)
DEC 3000/600 (175 MHz) (Sandpiper+)

Flamingo family
High-end TURBOchannel based workstations.

DEC 3000/500 (150 MHz) (Flamingo)
DEC 3000/500X (200 MHz) (Hot Pink)
DEC 3000/800 (200 MHz) (Flamingo II)

21064-A **EV45** (225-333 MHz, 0.50 micron)

DEC 3000/700 (225 MHz) (Sandpiper45)
DEC 3000/900 (275 MHz) (Flamingo45)

Alpha XL 233-266 (XL)
AlphaPC 64 (EB64+)

Avanti family
Digital's lower-end PCI-based workstations.

AlphaStation 200 4/233 (Mustang+)
AlphaStation 205 4/133-333 (LX3)
AlphaStation 250 4/300 (M3+)
AlphaStation 255 4/133-333 (LX3+)
AlphaStation 300 4/266 (Melmac)
AlphaStation 400 4/233-300 (Avanti)

Sable family
AlphaServer 2000 4/233-275 (Demi-Sable)
AlphaServer 2100 4/233-275 (Sable)

AlphaServer 2100A (Lynx)

21066 **LCA4** (166-233 MHz, 0.75 micron)

NoName family
Digital's lowest-end family of PCI-based systems.

DEC AXPpci33 (NoName)
Universal Desktop Box AXPpci166MT (UDB/Multia)

21066 evaluation motherboard (EB66)

21066-A **LCA45** (233 MHz, 0.50 micron)

21066-A evaluation motherboard (EB66+)

21068 **LCA4s** (66-233 MHz, 0.75 micron)

Alpha Book (Burns)
Universal Desktop Box AXPpci233MT (UDB/Multia)

21164 **EV5** (250-366 MHz, 0.50 micron)

Alcor family
AlphaStation 500/266-333 (Maverick)
AlphaStation 600/266-300 (Alcor)
Alpha XL 300-433 (XLT)

Sable family
AlphaServer 2000 5/250-300 (Demi-Gamma)
AlphaServer 2100 5/250-300 (Gamma Sable)

Mikasa family
AlphaServer 1000 5/300 (Pinnacle)

Noritake family
AlphaServer 1000A 5/300 (Pinnacle)

Rawhide family (KN300)
AlphaServer 4000 5/266-300 (Wrangler)
AlphaServer 4000 5/266-300 (Durango)
AlphaServer 4100 5/266-300 (Dodge)

AlphaServer 8200 and 8400 (KN8AE)

21164 evaluation motherboard (EB164)

21164-A **EV56** (400-766 MHz, 0.35 micron, BWX)

Alcor family
AlphaStation 500/333-500 (Bret)

Personal Workstation (PWS)
PWS 433a/433au (Miata)
PWS 500a/500au (Miata)
PWS 600a/600au (Miata)

Sable family
AlphaServer 2100 5/375-400 (Gamma Sable)
AlphaServer 2000 5/375-400 (Demi-Gamma)

Mikasa family
AlphaServer 1000 5/333-500 (Primo)

Noritake family
AlphaServer 1000A 5/333-500 (Primo)
AlphaServer 600A 5/500 (Alcor-Primo)
AlphaServer 800 5/333-500 (Corelle)

Rawhide family (KN300)
AlphaServer 4000 5/400-666 (Wrangler)
AlphaServer 4000 5/400-666 (Durango)
AlphaServer 4100 5/400-666 (Dodge)

AlphaServer 1200 5/400-666 (Tincup)
AlphaServer 1200 5/400-666 (DaVinci)

EB164 family
AlphaPC 164 motherboard (EB164)
AlphaPC 164LX motherboard (EB164)

DigitalServer 3300 (rebadged AlphaServer 800 for NT)
DigitalServer 5300 (rebadged AlphaServer 1200 for NT)
DigitalServer 7300 (rebadged AlphaServer 4100 for NT)

AlphaServer 8200 and 8400 (KN8AE)

APi AlphaPC 164UX motherboard (Ruffian)

21164-PC **PCA56** (400-600 MHz, 0.35 micron, MVI, no L2 cache)

AlphaPC 164SX motherboard (EB164)

PWS 466au (Miata)
PWS 550au (Miata)

21264 **EV6** (450-600 MHz, 0.35 micron)

AlphaServer 8400 (KN8AE)

APi UP1000 and UP1100; AMD 751-based EV6 systems.

264DP, XP1000, DS10, DS20, APi UP2000, UP2000+ Tsunami-based systems.

21264-A **EV67** (600-833 MHz, 0.28 micron)

AlphaServer GS60E
AlphaServer GS140

21264-B **EV68AL** (833-1250 MHz, 0.18 micron)

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

apecs	DECchip 21072/21071 Core Logic chipset
asc	TURBOchannel single-channel SCSI adapter
cia	DECchip 2117x Core Logic chipset
dwlpx	DEC DWLPA and DWLPB PCI adapter
gbus	internal bus on AlphaServer CPU modules
irongate	APi UP1000 AMD751 Core Logic + AGP chipset
jensenio	DEC 2000/300 (Jensen) I/O module
kft	KFTIA and KFTHA Bus Adapter Node for I/O hoses
lca	DECchip 21066 Core Logic chipset
mcbus	MCBUS system bus found on AlphaServer 4100 systems
mcpcia	MCPCIA MCBUS-to-PCI bus adapter

sableio	AlphaServer 2100 (Sable) STD I/O module
tcasic	TURBOchannel host bus support
tlsb	AlphaServer 8x00 TurboLaser System bus
tsc	DECchip 21272 Core Logic chipset
tsp	DECchip 21272 Core Logic chipset PCI controller
ttwoga	DEC T2 Gate Array
ttwopci	DEC T2 Gate Array PCI controller

TURBOchannel devices are supported through the `tc`(4) bus and associated device drivers.

PCI devices are supported through the `pci`(4) bus and associated device drivers.

ISA devices are supported through the `isa`(4) bus and associated device drivers.

EISA devices are supported through the `eisa`(4) bus and associated device drivers.

PCMCIA devices are supported through the `pcmcia`(4) bus and associated device drivers.

Console devices using ISA, EISA, or PCI video adaptors and standard AT or PS/2 keyboards are supported by the machine independent `wscons`(4) console driver.

HISTORY

This alpha **`intro`** appeared in NetBSD 1.6.

NAME

intro — introduction to amiga special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

AMIGA DEVICE SUPPORT

This section describes the hardware supported on the Amiga. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system will have to be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4)

HISTORY

The Amiga **intro** man page first appeared in NetBSD 1.1

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

afsc	A4091 low level SCSI adapter interface
ahsc	A3000 low level SCSI adapter interface
atzsc	A2091 low level SCSI adapter interface
ed	DP8390-based Ethernet interface
es	SMC91C90-based Ethernet interface
fdc	Floppy disk controller device
fd	Floppy disk device
grf	frame buffer for Custom Chips and graphics cards
grfcl	color graphics driver for GVP Spectrum/Picasso II, II+ and IV/Piccolo/Piccolo SD64 cards
grfcv	color graphics driver for the Cybervision 64
grfcv3d	color graphics driver for the Cybervision 64/3D
grfet	color graphics driver for Domino/Domino16M proto/oMniBus/Merlin cards
grful	color graphics driver for the A2410

grfrh	color graphics driver for Retina BLT Z3 and Altais cards
grfrt	color graphics driver for the Retina Z2
gtsc	GVP low level SCSI adapter interface
gvpbus	GVP custom bus
kbd	Amiga Keyboard device
kmem	kernel virtual memory
ite	Amiga Internal Terminal Emulator
ivsc	IVS low level SCSI adapter interface
le	AMD 7990 and AMD 79C960 Lance-based Ethernet interface
mem	physical memory
mfcs	MultiFaceCard II/II serial interface
mgnsc	Magnum 40 low level SCSI adapter interface
otgsc	12 Gauge low level SCSI adapter interface
par	8520 built-in parallel interface
ser	8520 built-in serial interface
wesc	Warp Engine low level SCSI adapter interface
wstsc	Wordsync II low level SCSI adapter interface
zssc	Zeus low level SCSI adapter interface
zbus	Amiga Zorro II/III bus

NAME

intro — introduction to special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

DEVICE SUPPORT

This section describes the hardware supported by NetBSD/arc. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4)

HARDWARE

NetBSD/arc supports a variety of systems conforming to the ARC machine specification. The following systems are supported:

Acer PICA
DESKstation rPC44
DESKstation Tyne
MIPS Magnum 4000
NEC Express 5800/230 PCI R4K
NEC Express 5800/240 EISA R4K
NEC Express RISCserver
NEC ImageRISCstation
NEC RISCserver 2200
NEC RISCstation 2200 EISA
NEC RISCstation 2200 PCI
NEC RISCstation 2250

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

arcsisabr	DESKstation rPC44 ISA host bridge
jazzio	Jazz internal bus host bridge
jazzisabr	Jazz ISA/EISA bus bridge
necpb	NEC RISCstation PCI host bridge
tyneisabr	DESKstation Tyne ISA host bridge

The following devices on the Jazz internal bus are supported.

asc	NCR 53c9x-based SCSI interface
com	NS16550-based serial communications interface
fdc	Floppy disk controller
lpt	Parallel port
mcclock	DS1287 real-time clock
oosiop	Symbios/NCR 53c700-based SCSI interface
osiop	Symbios/NCR 53c710-based SCSI interface
pckbc	PC keyboard controller
sn	SONIC Ethernet
timer	Interval timer
vga	VGA graphics

PCI devices are supported through the `pci`(4) bus and associated device drivers.

ISA devices are supported through the `isa`(4) bus and associated device drivers.

Console devices using ISA, Jazzio, or PCI video adaptors and standard AT or PS/2 keyboards are supported by the machine independent `wscons`(4) console driver.

UNSUPPORTED DEVICES

The following devices are not supported, due to unavailability of either documentation or sample hardware:

AD1848 audio on Jazzio
EISA devices
VXL framebuffer on MIPS Magnum and RISCstation 2200 EISA

HISTORY

This arc **`intro`** appeared in NetBSD 2.0.

BUGS

DESKstation rPC44 and Tyne support is currently broken.

NAME

intro — introduction to atari special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the **SYNOPSIS** section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

LIST OF PLATFORMS SUPPORTED

Platforms supported by the atari port:

TT030	A standard TT030 model with at least 4Mb of RAM.
Falcon	A standard Falcon with at least 4Mb of RAM. An FPU is not required as the default kernels include FP-emulation support.
Hades	A standard Hades with either an 040 or 060 processor and at least 8 Mb of RAM.

DEVICE SUPPORT

This section describes the hardware supported on the atari (atari-clone) platform. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed, not all devices exist on all models.

Standard builtin devices:

clock	System clock
fd	Floppy device as found on the Falcon/TT030
grf	The standard internal video as found on the Falcon and TT030.
grfet	The et4000-PCI video as found on the HADES
hdfd	Floppy device as found on the Hades (NEC 765 compatible)
isa	ISA I/O bus (Hades only)
kbd	Standard keyboard
lpt	Parallel port device interface
mem	Main memory interface
ncrscsi	Onboard 5380 SCSI-bus

nvr	Non-volatile RAM interface
pci	PCI I/O bus (Hades only).
ser0	Serial1 (when connector available).
vme	VME I/O bus
wd	IDE interface (not on TT030)
zs0	Serial2 and modem2 ports.

SEE ALSO

`config`(1), `autoconf`(4), `netintro`(4)

HISTORY

The atari **`intro`** appeared in NetBSD 1.3.

NAME

intro — introduction to special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

DEVICE SUPPORT

This section describes the hardware supported by NetBSD/cobalt. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

HARDWARE

The following systems are supported:

Qube/Raq 1
Qube/Raq 2

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

gt PCI bridge
com serial communications interface

PCI devices are supported through the pci(4) bus and associated device drivers.

SEE ALSO

config(1), autoconf(4), gt(4)

HISTORY

This cobalt **intro** appeared in NetBSD 2.0.

NAME

intro — introduction to dreamcast special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

DEVICE SUPPORT

This section describes the hardware supported on the Dreamcast platform. Software support for these devices come in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time, it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

Standard builtin devices:

g2bus	"G2" internal I/O bus
gapspci	PCI bridge used in expansion port peripherals
gdrom	Builtin GD-ROM optical disc drive
pvr	Framebuffer device using the builtin NEC PVR graphics subsystem
aica	Builtin AICA sound system

Controller port peripherals are supported though the maple(4) bus and associated device drivers.

Network interfaces:

rtk	Ethernet driver for the HIT-0400 Broadband Adapter
mbe	Ethernet driver for the HIT-0300 LAN Adapter

SEE ALSO

config(1), autoconf(4), netintro(4)

HISTORY

The **intro** man page appeared in NetBSD 2.0.

NAME

intro — introduction to evbarm special files and hardware support

DESCRIPTION

The evbarm port is really a collection of ports of NetBSD to a range of development and evaluation boards based on the ARM Architecture.

This section describes the supported boards, the special files, related driver functions, and networking support available in each system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the `config`(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log `/var/log/messages` due to errors in device operation; see `syslogd`(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in `netintro`(4).

SUPPORTED BOARDS

Each supported board requires a custom kernel for that board.

The following boards are currently supported by the evbarm configuration:

Integrator	The Integrator/AP development system from ARM Ltd. Supported processor boards include the CM10200E and CM920T.
IQ80310	The reference platform for the XScale-based IOP310 I/O processor from Intel.
IQ80321	The reference platform for the XScale-based IOP321 I/O processor from Intel.
Npwr	The Npwr board, from Team ASA, is based on the IOP310 processor from Intel and targetted at the network-attached storage space.
IXM1200	The reference platform for the XScale-based IXP1200 Network processor from Intel.
SMDK2410	The reference platform for the ARM920T-based S3C2410 processor from Samsung.
SMDK2800	The reference platform for the ARM920T-based S3C2800 processor from Samsung.
BRH	The BRH (Big Red Head) is an evaluation and development platform from ADI Engineering, based on the XScale-based I80200 processor
NTNP425B	An evaluation and development platform from NOVATEC, using the XScale-based IXP425 processor.
DBPXA250	An evaluation and development platfrom from Intel, using the XScale-based PXA250 processor.

DEVICE SUPPORT

This section describes some of the hardware supported on the various boards. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see `mknod`(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see `socket`(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in `autoconf`(4). A list of the supported devices is given below.

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

On the Integrator:

plcom	The PrimeCell PL010 UART.
plrtc	The PrimeCell PL030 Real-time Clock
fxp	Intel i82559 Ethernet PCI card.
ahc	Adaptec aic7880 SCSI controller.

Other PCI based cards may also work, but have not been tested.

On the IQ80310:

fxp	On-board Intel i82559 Ethernet
com	On-board NS16550-compatible serial ports

Other PCI devices in the PCI expansion slots. On the IQ80321:

wm	On-board Intel i82544EI Gigabit Ethernet
com	On-board NS16550-compatible serial port
iopaau	On-chip Application Accelerator Unit
iopwdog	On-chip watchdog timer.

Other devices in the PCI expansion slots.

On Npwr:

wm	On-board Intel i82544EI Gigabit Ethernet
siop	On-board LSI Logic 53c1010 Ultra160 SCSI
com	On-board NS16550-compatible serial port

On the IXM1200:

fxp	On-board Intel i82559 Ethernet
nppb	On-board Intel i21555 Non-Transparent PCI-PCI Bridge
ixpcom	On-chip serial port

On the SMDK2410:

sscom	On-chip serial ports
ohci	On-chip USB host controller

On the SMDK2800:

sscom	On-chip serial ports
sspci	On-chip Host-PCI bridge

Other devices in the PCI expansion slots.

On the BRH:

com	On-board NS16550-compatible serial ports
fxp	On-board i82559 Ethernet controller

Other devices in the PCI expansion slots.

On the NtNP425B:

ixpcom	On-chip serial ports
ixpwdog	On-chip watchdog timer

Other devices in the PCI/mPCI slot.

On the DBPXA250:

com	On-board NS16550-compatible serial port
sm	On-board SMC91C96 Ethernet controller
sacc	On-board SA-1111 StrongARM companion chip
pckbd	PS/2 keyboard
lcd	640x480 LCD

Other devices in the PCMCIA and CF card slots.

SEE ALSO

`config`(1), `autoconf`(4)

HISTORY

The evbarm **`intro`** appeared in NetBSD 2.0.

NAME

intro — introduction to hp300 special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the `config`(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log `/var/log/messages` due to errors in device operation; see `syslogd`(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in `netintro`(4).

HP DEVICE SUPPORT

This section describes the hardware supported on the HP 9000/300 series. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see `mknod`(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see `socket`(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system will have to be rebooted.

The autoconfiguration system is described in `autoconf`(4).

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Pseudo-devices are not listed. Devices are indicated by their functional interface. Occasionally, new devices of a similar type may be added simply by creating appropriate table entries in the driver; for example, new CS/80 drives.

com	serial interfaces (98644, 98626, and built-in apci and dca)
ct	7946/9144 CS/80 cartridge tape
dcm	HP 98642A communications multiplexer
dio	DIO/DIO-II bus
dnkbd	Domain keyboard
dvbox	HP98730 ‘‘DaVinci’’ device interface
fhpib	‘‘fast’’ HP-IB interface
frodo	Frodo ASIC (a.k.a. Apollo Utility Chip)
gbox	HP98700 ‘‘Gatorbox’’ device interface
grf/ite	Topcat/Gatorbox/Renaissance frame buffer
hil	HIL interface
nhpib	Built-in and 98625 HP-IB interface
hyper	Hyperion device interface
intio	HP300 internal I/O space driver
ite	HP Internal Terminal Emulator
le	98643 Lance-based Ethernet interface
ppi	HP-IB printer/plotter interface
rbox	HP98720 ‘‘Renaissance’’ device interface
rd	CS/80 disk interface
rmp	HP Remote Maintenance Protocol family
spc	HP98265A SCSI interface

topcat HP98544-98550 ‘‘Topcat’’ and ‘‘Catseye’’ device interface

SEE ALSO

`config`(1), `autoconf`(4)

Building 4.3 BSD UNIX Systems with Config (SMM:2).

HISTORY

The HP300 **`intro`** appeared in 4.3BSD−Reno.

NAME

intro — introduction to special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

DEVICE SUPPORT

This section describes the hardware supported on the HP PA-RISC platform. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Character and block devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in autoconf(4).

SUPPORTED SYSTEMS

NetBSD hp700 should run on systems similar to those, listed below, as long as they're based upon the following:

- HP PA-RISC 1.1 processors:
  PA7000
  PA7100/PA7150
  PA7200/PA7250
  PA7100LC
  PA7300LC
- Viper memory controller;
- GSC bus controllers:
  ASP
  LASI Super-integrated I/O
  WAX Basic I/O
- SCSI I/O-processors:
  NCR53c700/710 Narrow Single-Ended
  NCR53c720 Fast Wide Differential
- Intel i82596 CA/DX LAN coprocessors
- Venom, (H)CRX-8 and (H)CRX-24 video controllers
- EISA Adapters:
  i82350 (Mongoose)

WAX-EISA
- PCI Adapters:
  Dino GSC-PCI bridge
  Cujo GSC-PCI 64bit bridge
- Human Interface Loop (HIL) keyboard and mouse
- PS/2 port keyboard and mouse

Below a list of HP 9000/700 models targetted for support is listed, including basic system characteristics.

Model	MHz	CPU	Caches, KB	Expansion
705	35	7000	32/64	N/A
710	50	7000	32/64	N/A
720	50	7000	128/256	EISA, GSC
730	66	7000	128/256	EISA, GSC
750	66	7000	256/256	4 EISA, 2 SGC
715	33	7100	64/64	EISA/SGC
715	50	7100	64/64	EISA/SGC
715	75	7100	256/256	EISA/SGC
725	50	7100	64/64	3 EISA, EISA/SGC
725	75	7100	256/256	3 EISA, EISA/SGC
735	100	7100	256/256	EISA, SGC
742i	50	7100	64/64	N/A
745i	50	7100	64/64	4 EISA
745i	100	7100	256/256	4 EISA
747i	50	7100	64/64	2 EISA, SGC, 6 VME
747i	100	7100	256/256	2 EISA, SGC, 6 VME
755	100	7100	256/256	4 EISA, 2 SGC
735	125	7150	256/256	EISA, SGC
755	125	7150	256/256	4 EISA, 2 SGC
712	60	7100LC	32/32	GIO, TSIO
712	80	7100LC	128/128	GIO, TSIO
712	100	7100LC	128/128	GIO, TSIO
715	64	7100LC	128/128	EISA/GSC
715	80	7100LC	128/128	EISA/GSC
715	100	7100LC	128/128	EISA/GSC
715XC	100	7100LC	512/512	EISA/GSC
725	64	7100LC	128/128	EISA, 3 EISA/GSC
725	100	7100LC	128/128	EISA, 3 EISA/GSC
743i	64	7100LC	128/128	2 GSC-M/2(4), VME
743i	100	7100LC	128/128	2 GSC-M/2(4), VME
748i	64	7100LC	128/128	2 GSC-M/2(4), 4 EISA/PCI, 6 VME
748i	100	7100LC	128/128	2 GSC-M/2(4), 4 EISA/PCI, 6 VME
SAIC	60	7100LC	32/32	GIO, TSIO, 2 PCMCIA
SAIC	80	7100LC	128/128	GIO, TSIO, 2 PCMCIA
J200	100	7200	256/256	GSC, 2 EISA, 2 EISA/GSC
J210	120	7200	256/256	GSC, 2 EISA, 2 EISA/GSC
J210XC	120	7200	1MB/1MB	GSC, 2 EISA, 2 EISA/GSC
C100	100	7200	256/256	GSC, 3 EISA/GSC
C110	120	7200	256/256	GSC, 3 EISA/GSC
744	132	7300LC	64/64	2 GSC-M/2(4), VME

744	165	7300LC	64/64+512	2 GSC-M/2(4), VME
745	132	7300LC	64/64	2 GSC-M/2(4), 4 EISA/PCI
745	165	7300LC	64/64+512	2 GSC-M/2(4), 4 EISA/PCI
748	132	7300LC	64/64	2 GSC-M/2(4), 4 EISA/PCI, 6 VME
748	165	7300LC	64/64+512	2 GSC-M/2(4), 4 EISA/PCI, 6 VME
A180	180	7300LC	64/64	2 HSC/PCI
A180C	180	7300LC	64/64+1MB	2 HSC/PCI
B132L	132	7300LC	64/64(+1MB)	GSC/PCI, GSC/PCI/EISA
B132L+	132	7300LC	64/64(+1MB)	GSC/PCI, GSC/PCI/EISA
B160L	160	7300LC	64/64(+1MB)	GSC/PCI, GSC/PCI/EISA
B180L+	180	7300LC	64/64(+1MB)	GSC/PCI, GSC/PCI/EISA
C132L	132	7300LC	64/64(+1MB)	2 GSC/PCI/EISA, 2 GSC/EISA
C160L	160	7300LC	64/64(+1MB)	2 GSC/PCI/EISA, 2 GSC/EISA
RDI	132	7300LC	64/64(+1MB)	2 CardBus
RDI	160	7300LC	64/64(+1MB)	2 CardBus
RDI	180	7300LC	64/64(+1MB)	2 CardBus

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

`asp`(4) The ASP I/O controller; found in most of the older machines which don't have LASI or WAX. Includes GSC-bus controller, to which most of the devices are attached. Most of I/O is still on external chips though.

`com`(4) The RS-232 ports.

`cpu`(4) The Central Processor itself, makes sure the machine does something useful.

`tlp`(4) DEC/Intel 21140, 21143, 21145 and clone 10/100 Ethernet conroller.

`dino`(4) The `pci`(4) bridge on most of the newer systems.

`harmony`(4) CS4215/AD1849 audio.

`hil`(4) Human Interface Loop, sporting several input devices, e.g. keyboards, mice, tablets and such.

`iee`(4) i82596 CA/DX LAN controller. Found in every 700, either included in the MBA or on an external chip on mainboard.

`lasi`(4) LSI?, found in almost all HP 9000/700 workstations. Integrates most of the bus and I/O functions into one chip.

`lpt`(4) The Centronics printer port.

`mem`(4) Memory files and memory controller.

`mongoose`(4) The `eisa`(4) bus controller on most of the older 700 machines.

`oosiop`(4) Symbios/NCR 53C700 SCSI I/O processor.

`osiop`(4) Symbios/NCR 53C710 SCSI I/O processor.

`pdc`(4) The PROM interface, allows to call the routines in the machine's PROM for things like initial console output and such.

`siop`(4) LSI/Symbios Logic/NCR 53C8xx SCSI I/O processor.

`sti`(4) The system graphics driver.

`wax`(4) The other reincarnation of `lasi`(4) used for cheap ASIC implementations for add-on devices.

SEE ALSO

`config`(1), `autoconf`(4) `http://www.openpa.net/`

HISTORY

The hp700 **`intro`** first appeared with OpenBSD 2.0. It was ported to NetBSD 2.0 by Jochen Kunz.

NAME

intro — introduction to hpcsh special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the `config`(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log `/var/log/messages` due to errors in device operation; see `syslogd`(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in `netintro`(4).

DEVICE SUPPORT

This section describes the hardware supported on the hpcsh platform. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see `mknod`(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see `socket`(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in `autoconf`(4). A list of the supported devices is given below.

SUPPORTED SYSTEMS

The NetBSD/hpcsh port supports HITACHI SuperH family based Windows CE PDA machines. Currently supported models are

HP Jornada Series:

- HP 620LX
- HP Jornada 680, 680e, 690, 690e

Hitachi Persona Series:

- PERSONA HPW-50PAD
- PERSONA HPW-230JC
- PERSONA HPW-650PA

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

Standard SuperH on-chip peripheral devices:

adc	analog/digital converter
sci	serial communication interface
scif	serial communication interface with FIFO
shb	bus for SuperH on-chip peripheral devices

Companion chips:

hd64461	HD64461 Windows CE Intelligent Peripheral Controller

hd64461pcmcia
HD64461 integrated PCMCIA controller
hd64461video
HD64461 integrated LCD controller
hd64465 HD64465 Windows CE Intelligent Peripheral Controller
hd64465pcmcia
HD64465 integrated PCMCIA controller
hd64465video
HD64465 integrated LCD controller

Pointer devices:

j6x0lcd	LCD screen of HP Jornada 680 series machines
j6x0tp	touch screen of HP Jornada 680 series machines
psh3lcd	LCD screen of HITACHI PERSONA SH3 series machines
psh3tp	touch screen of HITACHI PERSONA SH3 series machines

SEE ALSO

`adc`(4), `j6x0lcd`(4), `j6x0tp`(4), `psh3lcd`(4), `psh3tp`(4), `shb`(4)

BUGS

This manual page is incomplete.

NAME

intro — introduction to i386 special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the `config`(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log `/var/log/messages` due to errors in device operation; see `syslogd`(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in `netintro`(4).

DEVICE SUPPORT

This section describes the hardware supported on the i386 (PC-clone) platform. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see `mknod`(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see `socket`(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in `autoconf`(4). A list of the supported devices is given below.

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

Standard builtin devices:

com	NS8250-, NS16450-, and NS16550-based asynchronous serial communications device interface
lpt	Parallel port device interface
fdc	Standard NEC 765 floppy disk controller.
mca	MCA I/O bus.
mem	Main memory interface
npx	Numeric Processing Extension coprocessor and emulator
pci	PCI I/O bus.
eisa	EISA I/O bus, either as main bus or via PCI-EISA bridge.
isa	ISA bus and ISA devices, either as main bus or via PCI-ISA bridge.
isa	isa I/O bus.
isapnp	‘‘bus’’ for ISA devices with PnP support.
speaker	console speaker device interface

PCMCIA devices are supported through the `pcmcia`(4) bus and associated device drivers.

Cardbus devices are supported through the `cardbus`(4) bus and associated device drivers.

USB devices are supported through the `usb`(4) bus and associated device drivers.

Console devices using ISA, EISA, or PCI video adaptors and standard AT or PS/2 keyboards are supported by the machine independent `wscons`(4) console driver.

Disk, tape and SCSI devices:

aha	Adaptec 154x ISA SCSI adapter boards.
ahb	Adaptec 1742 EISA SCSI adapter boards.
ahc	Adaptec 274x, 284x, 2940 and 3940 VL/EISA/PCI SCSI adapter boards.
aic	Adaptec AIC-6260, Adaptec AIC-6360, Adaptec 152x, and SoundBlaster SCSI boards.
bha	Buslogic BT-445 (ISA), BT-74x (EISA), and BT-9[45][68] (PCI) SCSI boards.
mcd	Mitsumi CD-ROM drives.
ncr	Symbios (formerly NCR) PCI SCSI adapter boards.
pciide	PCI IDE controllers.
sea	Seagate/Future Domain SCSI cards. ST01/02, Future Domain TMC-885, and Future Domain TMC-950.
uha	Ultrastor ISA and EISA SCSI adapter cards. Ultrastore 14f, Ultrastore 34f, and Ultrastore 24f.
wdc	Standard ISA Western Digital type hard drives controllers. MFM, RLL, ESDI, and IDE.
wt	Wangtek and compatible ISA controllers for QIC-02 and QIC-36 tapes.

Network interfaces:

de	Ethernet driver for dc21040, dc21042, and dc21140-based 10Mbit and 100Mbit PCI Ethernet adaptors, including DE-430, DE-450 DE-500, SMC EtherPower, and Znyx.
fea, fpa	FDDI driver for Digital DEFEA (EISA) and DEFPA FDDI adaptors.
ed	Western Digital/SMC 80x3 and Ultra, 3Com 3c503, and Novell NE1000 and 2000 Ethernet interface
eg	3Com 3c505 Ethernet board.
el	3Com 3c501 Ethernet board.
ep	3Com EtherLink III (3c5x9) Ethernet interface
ie	Ethernet driver for the AT&T StarLAN 10, EN100, StarLan Fiber, and 3Com 3c507.
iy	Ethernet driver for the ISA Intel EtherExpress PR0/10 adaptor.
le	Ethernet driver for BICC Isolan, Novell NE2100, Digital DEPCA cards, and PCnet-PCI cards.
tl	Ethernet driver for ThunderLAN-based Ethernet adaptor.

Serial communication cards:

ast	multiplexing serial communications card first made by AST.
boca	Boca BB100[48] and BB2016 multiplexing serial communications cards. NS8250-, NS16450-, and NS16550-based asynchronous serial communications device interface, or internal modems that provide a serial-chip compatible interface.
cy	Cyclades Cyclom-4Y, -8Y, and -16Y asynchronous serial communications device interface
rtfps	a multiplexing serial communications card derived from IBM PC/RT hardware.

Sound cards:

gus	Gravis Ultrasound non-PnP soundcards.
guspnp	Gravis Ultrasound PnP soundcards.
pas	ProAudio Spectrum soundcards.
pss	Personal Sound System-compatible soundcards, including Cardinal Digital SoundPro 16 and Orchid Soundwave 32.
sb	Soundblaster, Soundblaster 16, and Soundblaster Pro soundcards.
wss	Windows Sound System-compatible sound cards based on the ad1848 chip.

Mouse and pointer devices:

joy	joystick game adaptor
lms	Logitech-style bus mouse device interface
mms	Microsoft-style bus mouse device interface
pms	PS/2 auxiliary port mouse device interface

Serial mice can be configured on any supported serial port.

SEE ALSO

`config`(1), `autoconf`(4), `netintro`(4)

HISTORY

The i386 **`intro`** appeared in NetBSD 1.0.

NAME

intro — introduction to mac68k special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

MAC68K DEVICE SUPPORT

This section describes the hardware supported on the Mac. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system will have to be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4)

HISTORY

The Mac68k **intro** man page first appeared in NetBSD 1.3.

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

adb	Apple Desktop Bus event interface
ae	DP8390-based Ethernet interface
asc	Apple Sound Chip
esp	NCR 53C9x built-in SCSI interface
grf	on-board and NuBus-based video interface
kmem	kernel virtual memory
ite	Mac68k Internal Terminal Emulator
mem	physical memory
ncrscsi	NCR 5380 built-in SCSI interface
sbc	NCR 5380 built-in SCSI interface
sn	DP83932-based Ethernet interface (SONIC)
zsc	Zilog Z8530 built-in serial interface

NAME

intro — introduction to macppc special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

MACPPC DEVICE SUPPORT

This section describes the hardware supported on the Power Macintosh. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system will have to be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4), awacs(4), bm(4), gem(4), mc(4), tlp(4)

HISTORY

The macppc **intro** man page first appeared in NetBSD 1.5.1, based on the NetBSD/mac68k intro(4).

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

PCMCIA devices are supported through the pcmcia(4) bus and associated devices.

Cardbus devices are supported through the cardbus(4) bus and associated devices.

USB devices are supported through the usb(4) bus and associated devices.

Additionally, the following specific devices are supported:

adb	Apple Desktop Bus event interface.
awacs	Audio Waveform Amplifier and Converter — Apple Sound Chip (supported systems include 603 and 604 based machines, and Open Firmware 3 based machines).
bm	BMac Ethernet.
de	DECchip 21x4x based Ethernet cards (see also tlp(4)).
esp	NCR 53C9x built-in SCSI interface.
gm	GMac Ethernet.
kmem	kernel virtual memory.

mc	MACE Ethernet.
mem	physical memory.
mesh	MESH built-in SCSI interface (most Old World machines up to the 1999 series of G3 PowerBooks).
nvram	NVRAM access.
ofb	PCI based frame buffer with Open Firmware support.
openfirm	Open Firmware access.
tlp	DECchip 21x4x based Ethernet cards.
wdc	Standard on-board IDE controller.
zsc	Zilog Z8530 built-in serial interface.

NAME

intro — introduction to mvme68k special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the `config`(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log `/var/log/messages` due to errors in device operation; see `syslogd`(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in `netintro`(4).

MVME68K DEVICE SUPPORT

This section describes the hardware supported on the Motorola MVME68K series of single board computers . Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see `mknod`(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see `socket`(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system will have to be rebooted.

The autoconfiguration system is described in `autoconf`(4). A list of the supported devices is given below.

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

clmpcc	Cirrus Logic CD2401 four channel Communications controller
clock	Hardware counter/timer clock driver
ie	Intel 82596-based Ethernet interface
kmem	kernel virtual memory
le	AMD Lance-based Ethernet interface
lpt	Onboard parallel printer interface
mem	physical memory
mcchip	MVME162 and MVME172 MCChip (similar to PCCchip2)
ncrsc	NCR 53c710 SCSI I/O Processor
pcc	MVME147 PCC Chip
pcctwo	MVME167 and MVME177 PCCchip2
vmepcc	MVME147 VMEbus interface chip
vmetwo	MVME1[67]x Enhanced VMEbus interface chip
wdsc	WD33c93 SCSI Bus Interface Controller
zsc	Zilog Z8530 Dual-port Serial Interface on MVME147 and MVME162

SEE ALSO

`config`(1), `autoconf`(4)

HISTORY

The mvme68k **intro** man page first appeared in NetBSD 1.5.

NAME

intro — introduction to pmax special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

DEVICE SUPPORT

This section describes the hardware supported on the pmax (MIPS-based DECstation/DECsystem) platform. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4)

SUPPORTED SYSTEMS

The following systems are supported:

DECstation 2100 and 3100
: also known as "pmax". The 2100 and 3100 differ only in CPU clock speed.

DECsystem 5100
: also known as "MIPSMATE".

DECstation 5000/200
: also known as "3MAX". The 5000/200 has a 25 MHz R3000 and is the first-generation TURBOchannel platform.

DECstation 5000/1xx
: also known as "3MIN" or "kmin". The 5000/1xx comes in 20 MHz, 25 MHz, and 33 MHz versions and is numbered appropriately. Two 12.5 MHz TURBOchannel slots are provided.

DECstation 5000/2x
: also known as "Personal DECstation" or "MAXINE". The 5000/xx comes in 20 MHz and 25 MHz variants. A baseboard 1024x786 framebuffer, and two 12.5 MHz TURBOchannel slots are provided.

DECstation 5000/240 and DECsystem 5900
: also known as "3MAXPLUS". These systems have a 40 MHz R3400 CPU and three 25 MHz TURBOchannel slots. The 5900 is an expanded-cabinet version of the 5000/240.

TURBOchannel systems (except the 5000/200) can be upgraded to an R4000 or R4400 CPU by upgrading the CPU daughterboard.

The Qbus-based DECsystem 5400 and 5500 are not supported.

The multi-processor XMI-bus DECsystem 5800 is not supported.

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

asc	NCR 53c94-based SCSI interface, either on DECstation 5000-series baseboard or PMAZ-AA SCSI option card.
bba	baseboard audio on 5000/xx systems.
dz	serial driver for DEC custom four-port serial device (dc7085 DZ-11 clone) on the baseboard of DECstation 2100/31000, 5100, and 5000/200 systems.
zsc	serial driver for Zilog SCC asynchronous/synchronous devices on the baseboard of DECstation 5000-series systems (excluding 5000/200).
le	Ethernet driver for baseboard or TURBOchannel option cards.
ioasic	Adaptor for the baseboard IO ASIC on second-generation TURBOchannel machines. An ioasic must be configured on a 5000/1xx, 5000/xx, and 5000/240 if support for baseboard devices or the TURBOchannel bus is desired.
wsdisplay	Pseudo-device driver supporting glass-tty console emulation on DEC framebuffers, DEC mice, and LK-201 family keyboards.
sii	DEC custom SCSI adaptor on DECstation 2100, 3100, 5100, and VAXstation 3100.
pm	DECstation 2100/3100 baseboard framebuffer
tc	Adaptor for the TURBOchannel I/O expansion bus. This must be included if any TURBOchannel option cards are supported, or for the baseboard Ethernet and SCSI devices on a 5000/200.
cfb	PMAG-B TURBOchannel 1024x876 unaccelerated 2-D framebuffer.
sfb	PMAGB-BA TURBOchannel 1280x1024 accelerated framebuffer.
mfb	PMAG-AA TURBOchannel 1280x1024 mono/greyscale unaccelerated framebuffer.
tfb	PMAG-JA and PMAG-RO TURBOchannel 1280x1024 unaccelerated framebuffer.
px	PMAG-C TURBOchannel 2-D accelerated graphics board.
pxg	PMAG-D, E and F TURBOchannel 2-D/3-D accelerated graphics boards.

UNSUPPORTED DEVICES

The following devices are not supported, due to unavailability of either documentation or sample hardware:

LoFi	DEC Western Research Labs audio card

The floppy-disk drive on the MAXINE baseboard is currently not supported.

HISTORY

This pmax **intro** appeared in NetBSD 1.2.

NAME

intro_pmppc — introduction to pmppc special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the `config`(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log `/var/log/messages` due to errors in device operation; see `syslogd`(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in `netintro`(4).

PMPPC DEVICE SUPPORT

This section describes the hardware supported on the Artesyn PM/PPC card. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see `mknod`(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see `socket`(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system will have to be rebooted.

The autoconfiguration system is described in `autoconf`(4). A list of the supported devices is given below.

SEE ALSO

`config`(1), `autoconf`(4), `cpc`(4), `cs`(4), `rtc`(4)

HISTORY

The evbppc **intro_pmppc** man page first appeared in NetBSD 5.0.

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

PCI devices are supported through the `pci`(4) bus and associated devices.

USB devices are supported through the `usb`(4) bus and associated devices.

Additionally, the following specific devices are supported:

com	serial ports in the CPC700
cpc	PCI host controller CPC700
cs	Ethernet driver
rtc	Real Time Clock driver

NAME

intro — introduction to sgimips special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the `config`(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log `/var/log/messages` due to errors in device operation; see `syslogd`(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in `netintro`(4).

DEVICE SUPPORT

This section describes the hardware supported by NetBSD/sgimips. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see `mknod`(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see `socket`(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in `autoconf`(4). A list of the supported devices is given below.

HARDWARE

The following systems are supported:

O2	IP32 (“Moosehead”)
Indy/Challenge S	IP24 (“Guinness”)
Indigo 2/Challenge M	IP22 (“Fullhouse”)
Indigo R4k	IP20 (“Blackjack”)
Indigo R3k	IP12 (“Hollywood”)
Personal Iris 4D/3x	IP12 (“Magnum”)

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

crime	found on the O2
dpclock	real-time clock
dsclock	real-time clock
gio	PCI-like bus
hpc	High performance Peripheral Controller
imc	Indigo R4k/Indy/Challenge S/Indigo2 bus arbiter
mace	found on the O2
mec	O2 MAC110 ethernet
newport	entry framebuffer on Indy and Indigo2
pic	Personal Iris 4D/3x and Indigo R3k bus arbiter

sq	SEEQ 8003/80C03 ethernet
tl	Set Engineering GIO 100baseTX Fast Ethernet
tlp	Phobos G130/G160 10/100 GIO Fast Ethernet
wdsc	WD33C93 SCSI interface

PCI devices are supported through the `pci`(4) bus and associated device drivers.

SEE ALSO

`config`(1), `autoconf`(4), `crime`(4), `dpclock`(4), `dsclock`(4), `gio`(4), `hpc`(4), `imc`(4), `mace`(4), `mec`(4), `newport`(4), `pic`(4), `sq`(4), `tlp`(4), `wdsc`(4)

HISTORY

This sgimips **`intro`** appeared in NetBSD 2.0.

NAME

intro — introduction to sparc special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

DEVICE SUPPORT

This section describes the hardware supported on the SPARC platform. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4), bwtwo(4), cd(4), cgeight(4), cgfour(4), cgfourteen(4), cgsix(4), cgthree(4), cgtwo(4), ch(4), fd(4), kbd(4), le(4), magma(4), mem(4), ms(4), openprom(4), scsi(4), sd(4), ss(4), st(4), tcx(4), uk(4)

SUPPORTED SYSTEMS

The following Sun SPARC system architectures and models are supported:

sun4 — first generation SPARC systems on VMEbus:
Sun 4/100 series (14.28 MHz)
Sun 4/200 series (16.67 MHz)
Sun 4/300 series (25 MHz)

sun4c — desktop SPARC systems with Sbus:
SPARCstation 1 (20 MHz)
SPARCstation 1+ (25 MHz)
SPARCstation 2 (40 MHz)
SPARCstation SLC (20 MHz)
SPARCstation ELC (33 MHz)
SPARCstation IPC (25 MHz)
SPARCstation IPX (40 MHz).

sun4m — desktop SPARC systems with Mbus for CPUs, and Sbus:
SPARCclassic (50 MHz microSPARC I)
SPARCstation LX (50 MHz microSPARC I)
SPARCstation 4 (70 MHz microSPARC II)
SPARCstation 5 (70, 85, 110 MHz microSPARC II)

SPARCstation 5 (170 MHz TurboSPARC)
SPARCstation 10M (36 MHz SuperSPARC I)
SPARCstation 20M (50 MHz SuperSPARC I)
SPARCstation 10 (Mbus modules)
SPARCstation 20 (Mbus modules)

The SPARCstation 2 and IPX can be upgraded with a Weitek PowerUP CPU that is clock-doubled (i.e. internally it runs at 80 MHz). NetBSD supports this configuration.

Hardware level clones of these systems from other manufacturers will likely work (e.g. Xerox, Tatung, Axil, Cycle); other systems which have a SPARC CPU but do not use Sun's hardware architecture (e.g. Solbourne) will likely not work.

The sun4m architecture with Mbus modules for the CPUs is supported with the following modules with only one CPU:

SM41 40 MHz SuperSPARC I with 1MB SuperCACHE

SM51 50 MHz SuperSPARC I with 1MB SuperCACHE

SM61 60 MHz SuperSPARC I with 1MB SuperCACHE

SM71 75 MHz SuperSPARC II with 1MB SuperCACHE

SM81 85 MHz SuperSPARC II with 1MB SuperCACHE

HS11 100 MHz Ross Technology hyperSPARC

HS21 125 MHz Ross Technology hyperSPARC

M151 150 MHz Ross Technology hyperSPARC

This list is not exhaustive; NetBSD is continuously being improved, and may well run on Mbus CPU modules not listed here.

There is also some support for Sun JavaStation computers based on the microSPARC CPU.

NetBSD does not yet properly support multiprocessor systems, but will run on one processor of a multiprocessor system.

The Sun 4/400 series, and sun4d (SPARCcenter 1000, 1000E, and 2000) are not supported.

The sun4u (UltraSPARC 64-bit) architectures are supported by NetBSD/sparc64.

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

audio AMD 79C30 obio (sun4c) and dbri (sun4m) audio controller

bpp Bi-directional Parallel port

bwtwo black and white obio frame buffer

cgeight 24 bit VMEbus color frame buffer

cgfour 8 bit obio (sun4 P4 bus) color graphics frame buffer

cgfourteen 24 bit Sbus color frame buffer

cgsix 8 bit obio (sun4c & sun4m), Sbus color graphics frame buffer

cgthree	8 bit VMEbus, Sbus, and obio (sun4m) color graphics frame buffer
cgtwo	8 bit VMEbus color frame buffer
dbri	Dual Basic Rate Interface (BRI) ISDN (SPARC LX & SPARCstation 10) (only the audio component is supported)
eeprom	Sun non-volatile configuration RAM driver
esp	NCR53C90 ESP100 (Sun 4/300), ESP100A (sun4c), ESP200 (sun4m) SCSI controller FSBE/S (X1053A, part # 501-2015) Fast SCSI-2/Buffered Ethernet Sbus controller
fd	Intel 82072 obio (sun4c) or Intel 82077 obio (sun4m) floppy disk drive controller
ie	Intel 82586 Ethernet controller (Sun 4/100)
isp	Qlogic ISP Sbus SCSI controller
kbd	Sun type 2, type 3, type 4, and type 5 keyboards (on zs)
le/lebuffer	AMD 7990 LANCE Ethernet controller (Sun 4/200, 4/300, sun4c, sun4m, Sbus)
magma	Magma Sp Serial/Parallel board device driver
ms	Sun mouse (on zs)
openprom	Sun Open boot PROM (what became IEEE 1275) configuration driver
power	sun4m power management; the `halt`(8) and `shutdown`(8) commands can use it to power down the system.
si	NCR5380 "SCSI-2" VMEbus (Sun 4/200, Sun 4/400) SCSI controller
sw	NCR5380 obio (Sun 4/100) "SCSI Wierd" SCSI controller
tcx	8 or 24 bit Sbus color graphics frame buffer
xd	Xylogics 753/7053 VMEbus SMD disk controller
xy	Xylogics 450/451 VMEbus SMD disk controller
zs	Zilog 8530 serial controller

UNSUPPORTED DEVICES

The following devices are not supported, due to unavailability of either documentation or sample hardware:

dbri	Dual Basic Rate Interface (BRI) ISDN (SPARC LX & SPARCstation 10)

HISTORY

This sparc **intro** appeared in NetBSD 1.3. Large chunks of text carefully recycled (shamelessly appropriated) from NetBSD/pmax **intro**.

NAME

intro — introduction to sparc64 special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

DEVICE SUPPORT

This section describes the hardware supported on the SPARC64 platform. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4), cd(4), cgsix(4), ch(4), kbd(4), le(4), magma(4), mem(4), ms(4), openprom(4), scsi(4), sd(4), ss(4), st(4), tcx(4), uk(4)

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

auxio	Auxiliary I/O & LED
bpp	Bi-directional Parallel port
cgsix	8 bit obio (sun4c & sun4m), Sbus color graphics frame buffer
com	PC-style serial port
eeprom	Sun non-volatile configuration RAM driver
esp	ESP200 SCSI controller FSBE/S (X1053A, part # 501-2015) Fast SCSI-2/Buffered Ethernet Sbus controller
fdc	Floppy Disk Controller
ffb	Creator & Creaor3D graphics frame buffer
isp	Qlogic ISP Sbus and PCI SCSI controller
kbd	Sun type 2, type 3, type 4, and type 5 keyboards (on zs or com)

le/lebuffer	AMD 7990 LANCE Ethernet controller
lpt	PC-style parallel port
magma	Magma Sp Serial/Parallel board device driver
ms	Sun mouse (on zs or com)
openprom	Sun Open boot PROM (what became IEEE 1275) configuration driver
power	power management `halt`(8) and `shutdown`(8) commands can use it to power down the system.
sab	Siemens 82532 & 83538 serial controller
zs	Zilog 8530 serial controller

PCI devices are supported through the `pci`(4) bus and associated devices.

PCMCIA devices are supported through the `pcmcia`(4) bus and associated devices.

Cardbus devices are supported through the `cardbus`(4) bus and associated devices.

USB devices are supported through the `usb`(4) bus and associated devices.

UNSUPPORTED DEVICES

The following devices are not supported, due to unavailability of either documentation, sample hardware, or willing volunteer:

atifb	ATI 3D Rage Pro VGA graphics adapter (Ultra5, Ultra10)
fdc	sun4u floppy drive controllers (EBus based machines only)
cgfourteen	24 bit Sbus color frame buffer
cgthree	8 bit Sbus color frame buffer

HISTORY

This sparc64 **`intro`** appeared in NetBSD 1.6. Large chunks of text carefully recycled (shamelessly appropriated) from NetBSD/sparc **`intro`**.

NAME

intro — introduction to sun2 special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

DEVICE SUPPORT

This section describes the hardware supported on the Sun2 platform. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4), cd(4), sd(4), ss(4), st(4)

SUPPORTED SYSTEMS

The following Sun2 system architectures and models are supported:

sun2 Sun2 systems: (MC68010)
Sun 2/120, 2/170 (10 MHz)

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

bwtwo black and white obio frame buffer

ie Intel 82586 Ethernet controller (Sun 2/120, 2/170)

ec 3Com 3c400 Multibus Ethernet controller (Sun 2/120, 2/170)

kbd Sun type 2, type 3, type 4, and type 5 keyboards (on zstty)

ms Sun mouse (on zstty)

sc Sun "SCSI-2" Multibus SCSI controller

zs Zilog 8530 serial controller

HISTORY

This sun2 **intro** appeared in NetBSD 1.6. Large chunks of text carefully recycled (shamelessly appropriated) from NetBSD/sun3 intro(4).

NAME

intro — introduction to sun3 special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

DEVICE SUPPORT

This section describes the hardware supported on the Sun3 platform. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system must be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4), cd(4), sd(4), ss(4), st(4)

SUPPORTED SYSTEMS

The following Sun3 system architectures and models are supported:

sun3 Sun3 systems: (MC68020)
Sun 3/50 (16 MHz)
Sun 3/60 (20 MHz)
Sun 3/100 series (16.67 MHz)
Sun 3/200 series (25 MHz)

sun3x Sun3X systems: (MC68030)
Sun 3/80 (20 MHz)
Sun 3/470 (33 MHz)

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

bwtwo black and white obio frame buffer

cgtwo 8 bit VMEbus color frame buffer

cgfour 8 bit obio color graphics frame buffer

eeprom Sun non-volatile configuration RAM driver

esp NCR53C90 ESP100 (Sun 3/80) SCSI controller

fd Intel 82072 obio (Sun 3/80) floppy disk drive controller

ie Intel 82586 Ethernet controller (Sun 3/100, 3/200, 3/470)

kbd Sun type 2, type 3, type 4, and type 5 keyboards (on zs)

le/lebuffer
AMD 7990 LANCE Ethernet controller (Sun 3/50, 3/60, 3/80)

ms Sun mouse (on zs)

si NCR5380 "SCSI-3" VMEbus SCSI controller

vme VMEbus support (Sun 3/100, 3/200, 3/470)

xd Xylogics 753/7053 VMEbus SMD disk controller

xy Xylogics 450/451 VMEbus SMD disk controller

zs Zilog 8530 serial controller

UNSUPPORTED DEVICES

The following devices are not supported, due to unavailability of either documentation or sample hardware:

sc Sun "SCSI-2" VMEbus SCSI controller

HISTORY

This sun3 **intro** appeared in NetBSD 1.3. Large chunks of text carefully recycled (shamelessly appropriated) from NetBSD/pmax intro(4).

NAME

intro — introduction to vax special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

VAX DEVICE SUPPORT

This section describes the hardware supported on the DEC VAX-11. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see physio(4) and mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device on either the UNIBUS (or Q-bus) or MASSBUS and, if found, enable the software support for it. If a UNIBUS device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a UNIBUS device which did not autoconfigure, the system will have to be rebooted. If a MASSBUS device comes "on-line" after the autoconfiguration sequence it will be dynamically autoconfigured into the running system.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4), netintro(4)

"Building 4.3 BSD UNIX Systems with Config", *SMM*, 2.

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Pseudo-devices are not listed. Devices are indicated by their functional interface. If second vendor products provide functionally identical interfaces they should be usable with the supplied software. **Beware, however, that we promise the software works ONLY with the hardware indicated on the appropriate manual page.** Occasionally, new devices of a similar type may be added simply by creating appropriate table entries in the driver.

acc	ACC LH/DH IMP communications interface
ad	Data translation A/D interface
css	DEC IMP-11A communications interface
crl	VAX-8600, 8650 console RL02 disk
ct	C/A/T or APS phototypesetter
ddn	ACC ACP625 DDN Standard Mode X.25 IMP interface
de	DEC DEUNA 10Mb/s Ethernet controller
dh	DH-11 emulators, terminal multiplexor
dhu	DHU-11 terminal multiplexor
dmc	DEC DMC-11/DMR-11 point-to-point communications device
dmf	DEC DMF-32 terminal multiplexor and parallel printer interface
dmz	DEC DMZ-32 terminal multiplexor
dn	DEC DN-11 autodialer interface
dz	DZ-11 terminal multiplexor

ec	3Com 10Mb/s Ethernet controller
en	Xerox 3Mb/s Ethernet controller (obsolete)
ex	Excelan 10Mb/s Ethernet controller
fl	VAX-11/780 console floppy interface
hdh	ACC IF-11/HDH IMP interface
hk	RK6-11/RK06 and RK07 moving head disk
hp	MASSBUS disk interface (with RP06, RM03, RM05, etc.)
ht	TM03 MASSBUS tape drive interface (with TE-16, TU-45, TU-77)
hy	DR-11B or GI-13 interface to an NSC Hyperchannel
ik	Ikonas frame buffer graphics device interface
il	Interlan 1010, 1010A 10Mb/s Ethernet controller
ix	Interlan NP-100 10Mb/s Ethernet controller
kg	KL-11/DL-11W line clock
lp	LP-11 parallel line printer interface
mt	TM78 MASSBUS tape drive interface
np	Interlan NP-100 10Mb/s Ethernet controller (intelligent mode)
pcl	DEC PCL-11 communications interface
ps	Evans and Sutherland Picture System 2 graphics interface
qe	DEC DEQNA Q-bus 10 Mb/s Ethernet interface
rx	DEC RX02 floppy interface
tm	TM-11/TE-10 tape drive interface
tmscp	TMSCP-compatible tape controllers (e.g., TU81, TK50)
ts	TS-11 tape drive interface
tu	VAX-11/730 TU-58 console cassette interface
uda	DEC UDA-50 disk controller
un	DR-11W interface to Ungermann-Bass
up	Emulex SC-21V, SC-31 UNIBUS disk controller
ut	UNIBUS TU-45 tape drive interface
uu	TU-58 dual cassette drive interface (DL-11)
va	Benson-Varian printer/plotter interface
vp	Versatec printer/plotter interface
vv	Proteon proNET 10Mb/s and 80Mb/s ring network interface

HISTORY

The section 4 **intro** appeared in 4.1BSD.

NAME

intro — introduction to x68k special files and hardware support

DESCRIPTION

This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPSIS section of each configurable device gives a sample specification for use in constructing a system description for the config(1) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /var/log/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system and network related information. The networking support is introduced in netintro(4).

X68k DEVICE SUPPORT

This section describes the hardware supported on the x68k platform. Software support for these devices comes in two forms. A hardware device may be supported with a character or block *device driver*, or it may be used within the networking subsystem and have a *network interface driver*. Block and character devices are accessed through files in the file system of a special type; see mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device and, if found, enable the software support for it. If a device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a device which did not autoconfigure, the system will have to be rebooted.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO

config(1), autoconf(4), intio(4), mfp(4), neptune(4), vs(4)

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. Not all supported devices are listed.

intio	Internal I/O virtual device
mfp	MC68901 MFP (Multi-Function Peripheral)
spc	Sharp genuine MB89352 SCSI host adaptor
mha	MK-HA1 Mankai-Seisakusho Mach-2 SCSI host adaptor
neptune	Neptune-X Ethernet interface
fdc	Built-in floppy disk controller device
fd	Floppy disk device
grf	Built-in frame buffer
kmem	kernel virtual memory
ite	x68k Internal Terminal Emulator
mem	physical memory
par	Centronics printer interface
com	NS16450 serial interface
zs	Z8530 built-in serial interface
kbd	x68k Keyboard device
ms	x68k mouse / trackball
bell	The keyboard bell emulator

pow	The power switch
vs	MSM6258V built-in voice synthesizer

HISTORY

The x68k **intro** man page first appeared in NetBSD 1.3.

NAME

aac — Adaptec AdvancedRAID Controller driver

SYNOPSIS

```
aac* at pci? dev ? function ?
ld* at aac? unit ?
```

DESCRIPTION

The **aac** driver provides support for the Adaptec AAC family of SCSI and SATA RAID controllers. These controllers support RAID 0, 1, 5, 10, and volume sets. They have four channels in the add-in version or 1-2 channels in the motherboard integrated version, and are most often found rebadged by Dell, Hewlett-Packard or IBM. Supported controllers include:

- Adaptec AAC-364
- Adaptec SCSI RAID 2120S
- Adaptec SCSI RAID 2200S
- Adaptec SATA RAID 2410SA
- Adaptec SCSI RAID 5400S
- Dell PERC 2/Si
- Dell PERC 2/QC
- Dell PERC 3/Di
- Dell PERC 3/Si
- Dell PERC 320/DC
- Dell CERC SATA RAID 1.5/6ch
- HP NetRAID 4M
- HP ML110 G2 (Adaptec SATA RAID 2610SA)
- IBM ServeRAID 8k

Access to RAID containers is available via the **ld** device driver. Individual drives cannot be accessed unless they are part of a container or volume set, and non-fixed disks cannot be accessed. Containers can be configured by using the on-board BIOS utility of the card.

DIAGNOSTICS

The adapter can send status and alert messages asynchronously to the driver. These messages are printed on the system console.

SEE ALSO

intro(4), ld(4)

HISTORY

The **aac** driver first appeared in NetBSD 1.6, and was based on the FreeBSD driver of the same name.

BUGS

This driver is not compatible with controllers that have version 1.x firmware. The firmware version is the same as the kernel version printed in the BIOS POST and driver attach messages.

NAME

ac97 — generic AC97 codec driver

DESCRIPTION

AC97 codecs contain the analog-to-digital (A/D), digital-to-analog (D/A), and mixing circuitry of many modern sound cards. AC97 codecs, for the most part, do not talk to host busses like the PCI bus directly. Instead, they communicate through an interface chip, called the host controller. The Ensoniq AudioPCI 97 (see eap(4)) is an example of such a host controller.

Unlike many drivers, the **ac97** driver does not appear in the configuration file. Instead, the driver is automatically attached by the drivers that require it.

SEE ALSO

auacer(4), auich(4), auixp(4), autri(4), auvia(4), clcs(4), clct(4), eap(4), emuxki(4), esa(4), esm(4), fms(4), neo(4), yds(4)

BUGS

The **ac97** driver does not keep track of the current user settings and instead relies on the hardware to do this.

The **ac97** driver could do more to detect mixer channels that don't work and cull them from the list.

NAME

acardide — acard IDE disk controllers driver

SYNOPSIS

acardide* **at pci? dev ? function ? flags 0x0000**

DESCRIPTION

The **acardide** driver supports the following IDE controllers:

Acard ATP850U Ultra33

Acard ATP860 Ultra66

Acard ATP860-A Ultra66

Acard ATP865-A Ultra100

It provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **acardide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

The timings used for the PIO and DMA modes for controllers listed above are for a PCI bus running at 30 or 33 MHz. This driver may not work properly on overclocked systems.

NAME

acc — ACC LH/DH IMP network interface

SYNOPSIS

```
pseudo-device imp acc0 at uba0 csr 167600 vector accrint accxint
```

DESCRIPTION

NOTE: At the moment NetBSD does not support IMP, so this manual page is not relevant.

The **acc** device provides a Local Host/Distant Host interface to an IMP. It is normally used when participating in the DARPA Internet. The controller itself is not accessible to users, but instead provides the hardware support to the IMP interface described in imp(4). The configuration entry for the imp(4) must also include the *pseudo-device* as shown above.

DIAGNOSTICS

acc%d: not alive. The initialization routine was entered even though the device did not autoconfigure. This indicates a system problem.

acc%d: can't initialize. Insufficient UNIBUS resources existed to initialize the device. This is likely to occur when the device is run on a buffered data path on an 11/750 and other network interfaces are also configured to use buffered data paths, or when it is configured to use buffered data paths on an 11/730 (which has none).

acc%d: imp doesn't respond, icsr=%b. The driver attempted to initialize the device, but the IMP failed to respond after 500 tries. Check the cabling.

acc%d: stray xmit interrupt, csr=%b. An interrupt occurred when no output had previously been started.

acc%d: output error, ocsr=%b, icsr=%b. The device indicated a problem sending data on output.

acc%d: input error, csr=%b. The device indicated a problem receiving data on input.

acc%d: bad length=%d. An input operation resulted in a data transfer of less than 0 or more than 1008 bytes of data into memory (according to the word count register). This should never happen as the maximum size of a host-IMP message is 1008 bytes.

SEE ALSO

netintro(4)

HISTORY

The **acc** interface appeared in 4.2BSD.

NAME

aceride — PCI IDE disk controllers driver

SYNOPSIS

```
aceride* at pci? dev ? function ? flags 0x0000
```

DESCRIPTION

The **aceride** driver supports the Acer Labs M5229 IDE controllers, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **aceride** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

The timings used for the PIO and DMA modes for controllers listed above are for a PCI bus running at 30 or 33 MHz. This driver may not work properly on overclocked systems.

NAME

acphy — Driver for Altima AC101, AC101L and AMD Am79c874 10/100 Ethernet PHYs

SYNOPSIS

acphy* **at mii? phy ?**

DESCRIPTION

The **acphy** driver supports the Altima AC101, AC101L and AMD Am79c874 NetPHY-1LP 10/100 Ethernet PHYs. These PHYs are often found on low-power Ethernet interfaces, such as MiniPCI interfaces found in laptops and embedded systems.

The AMD 79c874 is a work-alike (most likely an OEM of the core) of the Altima part.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

acpi — Advanced Configuration and Power Interface

SYNOPSIS

```
acpi0     at mainbus0
acpiacad* at acpi?
acpibat*  at acpi?
acpibut*  at acpi?
acpiec*   at acpi?
acpilid*  at acpi?
acpitz*   at acpi?
attimer*  at acpi?
aiboost*  at acpi?
com*      at acpi?
fdc*      at acpi?
hpet*     at acpi?
joy*      at acpi?
lpt*      at acpi?
mpu*      at acpi?
npx*      at acpi?
pckbc*    at acpi?
pcppi*    at acpi?
sony*     at acpi?
spic*     at acpi?
ug*       at acpi?
vald*     at acpi?
wss*      at acpi?
ym*       at acpi?

options   ACPI_DEBUG
options   ACPIVERBOSE
options   ACPI_ACTIVATE_DEV
options   ACPICA_PEDANTIC
options   ACPI_DSDT_OVERRIDE
```

DESCRIPTION

NetBSD provides machine-independent bus support for ACPI devices and includes several ACPI device drivers.

The NetBSD implementation of ACPI integrates Intel's ACPI Component Architecture (aka ACPI-CA) for the OS-independent part. The ACPI-CA provides OS-neutral ACPI functionalities such as ACPI BIOS table support, an ACPI event framework and an AML (ACPI Machine Language) interpreter.

Options:

`ACPI_DEBUG`	Enable ACPI debugging message outputs.
`ACPI_ACTIVATE_DEV`	Determine if the ACPI driver should attempt to activate inactive devices. The default is off.
`ACPICA_PEDANTIC`	Force strict conformance to the ACPI specification in the ACPI-CA. Do not enable this option unless you are debugging.

`ACPI_DSDT_OVERRIDE`	Force a given DSDT instead of the BIOS-supplied version. Use `ACPI_DSDT_FILE` to specify a DSDT.
`ACPI_DSDT_FILE="filename"`	If `ACPI_DSDT_FILE` is not specified, default to “dsdt.hex” in the build directory.

SUPPORTED DEVICES

NetBSD ACPI supports several machine-dependent and machine-independent devices:

machine-independent devices

acpiacad	ACPI AC adapters.
acpibat	ACPI Control Method Batteries.
acpibut	ACPI power and sleep buttons.
acpiec	ACPI Embedded Controllers.
acpilid	ACPI lid switches.
acpitz	ACPI thermal zones.
attimer	AT Timer.
aiboost	ASUS AI Booster Hardware monitor.
com	NS8250-, NS16450-, and NS16550-based serial ports.
fdc	Floppy disk controllers.
hpet	High Precision Event Timer.
joy	Joystick/Game port interface.
lpt	Standard ISA parallel port interface.
mpu	Roland MPU-401 (compatible) MIDI UART.
pcppi	AT-style speaker sound.
ug	Abit uGuru Hardware monitor.
wss	Windows Sound System-compatible sound cards based on the AD1848 and compatible chips.
ym	Yamaha OPL3-SA2 and OPL3-SA3 audio device driver.

i386-dependent devices

npx	i386 numeric processing extension coprocessor.
pckbc	PC keyboard controllers.
sony	Sony Miscellaneous Controller
spic	Sony programmable I/O controller.
vald	Toshiba Libretto device.

SEE ALSO

`acpiacad`(4), `acpibat`(4), `acpibut`(4), `acpiec`(4), `acpilid`(4), `acpitz`(4), `aiboost`(4), `apm`(4), `attimer`(4), `com`(4), `fdc`(4), `joy`(4), `lpt`(4), `mpu`(4), `npx`(4), `pci`(4), `pckbc`(4), `pcppi`(4), `sony`(4), `spic`(4), `ug`(4), `vald`(4), `wss`(4), `ym`(4), `acpidump`(8), `amldb`(8)

ACPI specification, http://www.acpi.info/.

Intel ACPI CA (Component Architecture), http://developer.intel.com/technology/iapc/acpi/.

HISTORY

The **acpi** driver appeared in NetBSD 1.6.

BUGS

Most of the ACPI power management functionalities are not implemented.

NAME

acpiacad — ACPI AC Adapter

SYNOPSIS

acpiacad* **at acpi?**

DESCRIPTION

The **acpiacad** driver supports ACPI AC Adapters.

The status (connected or disconnected) can be monitored by the envsys(4) API or the envstat(8) command. Actions against AC adapter online/offline events can be configured using the powerd(8) daemon.

SEE ALSO

acpi(4), envsys(4), envstat(8), powerd(8)

HISTORY

The **acpiacad** driver appeared in NetBSD 1.6.

NAME

acpibat — ACPI Control Method Battery

SYNOPSIS

acpibat* at acpi?

DESCRIPTION

The **acpibat** driver supports ACPI Control Method Battery.

The status can be monitored by envsys(4) API or envstat(8) command.

EVENTS

The **acpibat** driver is able to send events to powerd(8) when capacity state has been changed. The new state will be reported as the *fourth* argument to the */etc/powerd/scripts/sensor_battery* script. If a custom capacity limit was set via envstat(8), the **acpibat** driver will report a *user-capacity* event to the same script when current capacity limit has been reached.

SEE ALSO

acpi(4), envsys(4), envstat(8), powerd(8)

HISTORY

The **acpibat** driver appeared in NetBSD 1.6.

BUGS

Smart Battery Specification-based batteries are not supported.

NAME

acpibut — ACPI Button

SYNOPSIS

acpibut* at acpi?

DESCRIPTION

The **acpibut** driver supports ACPI fixed-feature or control method power/sleep buttons. Actions against these buttons can be configured using the powerd(8) daemon.

SEE ALSO

acpi(4), powerd(8)

HISTORY

The **acpibut** driver appeared in NetBSD 1.6.

NAME

acpidalb — Direct Application Launch Buttons

SYNOPSIS

acpidalb* **at acpi?**

DESCRIPTION

This driver provides support for PNP0C32 ACPI hotkeys aka “The Direct Application Launch Buttons”.

These are recognized on startup from system-wake or at runtime. Behaviour may differ from the standard specification in relation to the ACPI implementation.

The hotkeys are reported to the power daemon as `hotkey_button`.

SEE ALSO

`acpi`(4), `powerd`(8)

STANDARDS

This drivers conforms to the PNP0C32 specification provided at

http://www.microsoft.com/whdc/system/vista/DirAppLaunch.mspx

HISTORY

The **acpidalb** driver appeared in NetBSD 5.0.

AUTHORS

This driver was written for NetBSD by Christoph Egger.

NAME

acpiec — ACPI Embedded Controller

SYNOPSIS

```
acpiec* at acpi?
acpiecdt* at acpi?
```

DESCRIPTION

The **acpiec** driver supports ACPI Embedded Controllers.

It provides embedded controller access for other ACPI devices. On many systems which have an **acpiec** device, other ACPI devices such as acpiacad(4), acpibat(4), or acpitz(4) implicitly depend on the **acpiec** device. Therefore it is a good idea to enable **acpiec** with other ACPI devices unless you are sure the system does not have an **acpiec** device.

SEE ALSO

acpi(4)

HISTORY

The **acpiec** driver appeared in NetBSD 1.6.

NAME

acpilid — ACPI Lid Switch

SYNOPSIS

```
acpilid* at acpi?
```

DESCRIPTION

The **acpilid** driver supports ACPI Lid switches. Actions against lid open/close events can be configured using the powerd(8) daemon.

SEE ALSO

acpi(4), powerd(8)

HISTORY

The **acpilid** driver appeared in NetBSD 1.6.

NAME

acpitz — ACPI Thermal Zone

SYNOPSIS

acpitz* **at acpi?**

DESCRIPTION

The **acpitz** driver supports ACPI Thermal Zones.

The temperature can be monitored by the envsys(4) API or the envstat(8) command.

EVENTS

The **acpitz** driver is able to send events to powerd(8) when sensor's state has changed. When a Thermal Zone is critical, a *critical* event will be sent or if the Thermal Zone is hot, a *warning-over* event will be sent to the /etc/powerd/scripts/sensor_temperature script.

SEE ALSO

acpi(4), envsys(4), envstat(8), powerd(8)

HISTORY

The **acpitz** driver appeared in NetBSD 2.0.

BUGS

Any policy control such as fan control is not implemented.

NAME

ad — Data Translation A/D converter

SYNOPSIS

```
ad0 at uba0 csr 0170400 vector adintr
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **ad** driver provides an interface to the Data Translation A/D converter. This is *not* a real-time driver, but merely allows the user process to sample the board's channels one at a time. Each minor device selects a different A/D board.

The driver communicates to a user process by means of `ioctl`(2)s. The `AD_CHAN` `ioctl`(2) selects which channel of the board to read. For example,

```
chan = 5;
ioctl(fd, AD_CHAN, &chan);
```

selects channel 5. The `AD_READ` `ioctl`(2) actually reads the data and returns it to the user process. An example is

```
ioctl(fd, AD_READ, &data);
```

FILES

`/dev/ad`

DIAGNOSTICS

None.

HISTORY

The **ad** driver appeared in 4.1BSD.

NAME

adb — Apple Desktop Bus driver

SYNOPSIS

```
adb* at obio?

options MRG_ADB

#include <machine/adbsys.h>
```

DESCRIPTION

The Apple Desktop Bus (ADB) is the single-master, multiple-slave, low-speed serial bus interface used by Macintosh computers to connect input devices such as keyboards, mice, trackpads, trackballs, and graphics tablets to the machine. NetBSD provides support for the Apple Desktop Bus as found on all supported mac68k models, as well as macppc models with on-board ADB (PowerBooks and "Old World" models).

The **adb** driver accesses the ADB controller using the so-called "HWDIRECT" method. This method of access bypasses the Macintosh ROM and uses only NetBSD routines for ADB access. This is the only method supported on macppc and is the default for mac68k systems.

On mac68k systems there is an alternate method of accessing the ADB controller. With the Macintosh ROM Glue (MRG) method, the routines written for MacOS are used. To enable this method of ADB access, uncomment the line:

```
options MRG_ADB
```

in your kernel configuration file.

The `ioctl`(2) call is used to control the ADB event device. The following is a list of available `ioctl`(2) commands:

`ADBIOC_DEVSINFO` Get ADB Device Info

The **adb** event device will return an array of information containing an entry for each device connected to the bus. Each entry contains the current address, default address, and handler ID for the corresponding ADB device.

`ADBIOC_GETREPEAT` Get Keyboard Repeat Info

Returns a structure containing the current keyboard repeat delay and keyboard repeat interval.

`ADBIOC_SETREPEAT` Set Keyboard Repeat Rate

Sets the keyboard repeat delay and interval to the values specified by *`argp`*.

`ADBIOC_RESET` ADB Reset

Perform a reset of the ADB which will reinitialize all of the devices attached to the bus.

`ADBIOC_LISTENCMD` ADB Listen Command

Send data to the register of the ADB device specified by *`argp`*. This command is not fully implemented at this time.

SUPPORTED DEVICES

NetBSD includes support for the following ADB devices, sorted by driver name:

abtn ADB mouse button?

aed ADB event device

akbd ADB keyboard

ams ADB mouse

apm APM emulation

FILES

`/dev/adb` The ADB event device.

DIAGNOSTICS

aed0 at adb0 addr 0: ADB Event device This is a normal autoconfiguration message noting the presence of the **adb** event device.

adb0 at obio0 offset 0x16000 irq 18: 2 targets A standard autoconfiguration message indicating the initialization of the ADB subsystem.

adb: no devices found. No ADB devices were found to be connected to the bus during autoconfiguration.

adb: using %s series hardware support. Indicates the class of ADB hardware support the machine uses.

adb: hardware type unknown for this machine. The ADB hardware in this machine is currently unsupported.

adb: no ROM ADB driver in this kernel for this machine. The kernel lacks the necessary Macintosh ROM Glue (MRG) support for accessing the ADB hardware on this machine.

adb: using serial console. A serial console will be used for user input rather than the ADB event device.

adb: %s at %d. An ADB device of the type specified by *%s* has been found at location *%d*.

SEE ALSO

`aed`(4), `akbd`(4), `ams`(4), `apm`(4)

HISTORY

The **adb** interface first appeared in NetBSD 0.9. It has been under development ever since.

AUTHORS

Bradley A. Grantham wrote the original **adb** driver, including the MRG support. The hardware direct interface was written by John P. Wittkowski. The PowerManager interface was written by Takashi Hamada.

BUGS

- Not every class of ADB hardware is supported yet.
- The talk command is currently unimplemented.
- The listen command is not implemented yet.
- Not all multi-button mice are currently supported.
- Only mapped and relative-position ADB devices (i.e. keyboards and mice) are supported. Thus absolute-position and other exotic devices will not work.
- Some of the diagnostic messages in this man page need to be updated.

Some mac68k machines contain so-called dirty ROM. These machines are the: Mac SE/30, Mac II, Mac IIx, and Mac IIcx. Machines with dirty ROM may experience trouble booting if the MRG code is used, espe-

cially under the following conditions:

- Both a keyboard and a mouse are not attached to the computer.
- An extended keyboard is attached to the computer.

On (some) machines with dirty ROM, the ROM indicates the presence of a “ghost” keyboard or mouse. When this non-existant device is probed for, the result is an infinite loop. This is believed to be triggered by the **adb** driver probing for extended mice, and non-EMP Logitech mice.

NAME

adbbt — support for ADB hotkey devices found in some Apple laptops

SYNOPSIS

```
adbbt* at nadb?
wskbd* at adbbt?
```

DESCRIPTION

The **adbbt** driver handles all ADB hotkey devices within the wscons(4) framework. So far it only translates button events back to their corresponding function key codes.

SEE ALSO

nadb(4), wskbd(4), wsconsctl(8)

BUGS

Actually send hotkey events at least for the classes we can handle like display brightness.

NAME

adbkbd — support for ADB keyboards

SYNOPSIS

```
adbkbd* at nadb?
wskbd* at adbkbd? console ? mux 1
wsmouse* at adbkbd?
```

DESCRIPTION

The **adbkbd** driver handles most ADB keyboards within the wscons(4) framework. It also provides an interface to translate key strokes to mouse button events.

Which keys are translated to mouse button events can be configured for each individual keyboard via sysctl(8):

machdep.adbkbd0.middle
: Controls which scan code is used for middle mouse button events. Default is 103, which corresponds to F11.

machdep.adbkbd0.right
: Controls which scan code is used for right mouse button events. Default is 111, which corresponds to F12.

SEE ALSO

nadb(4), wskbd(4), wsmouse(4), wsconsctl(8), wskbd(9)

NAME

adbms — support for ADB mice, trackballs, and touchpads

SYNOPSIS

```
adbms* at nadb?
wsmouse* at adbms?
```

DESCRIPTION

The **adbms** driver handles most relative ADB pointing devices within the wscons(4) framework. For touchpads it also provides support for translating tapping the pad to mouse button events.

Tapping support can be turned on or off on a per-device basis using sysctl(8):

machdep.adbms0.tapping
: 0 disables tapping, 1 enables it.

SEE ALSO

nadb(4), wsmouse(4), wsconsctl(8)

NAME

adc — SuperH on-chip analog/digital converter

SYNOPSIS

adc* **at shb?**

DESCRIPTION

The **adc** driver provides support for a 10-bit successive-approximation A/D converter with a selection of up to eight analog input channels. ADC is an on-chip module found in some SuperH microprocessors (SH7709 and others).

SEE ALSO

shb(4)

HISTORY

The **adc** driver first appeared in NetBSD 2.0.

NAME

adt7467c — Analog Devices ADT7467 and ADM1030 hardware monitors and fan controllers

SYNOPSIS

```
adt7467c0 at ki2c?
```

DESCRIPTION

The **adt7467c** driver provides support for the Analog Devices ADT7467 and ADM1030 hardware monitor chips to be used with the envsys(4) API.

The ADT7467 supports five sensors:

Sensor	Units	Typical Use
temp0	uK	chip temperature
temp1	uK	CPU temperature
temp2	uK	GPU temperature
voltage0	uV DC	CPU Vcore
fan0	RPM	Chassis Fan

The ADM1030 supports three sensors:

Sensor	Units	Typical Use
temp0	uK	chip temperature
temp1	uK	CPU temperature
fan0	RPM	Chassis Fan

Due to hardware limitations, fresh sensor data is only available every 2 seconds.

Both controllers support fan speed control based on temperature thresholds - the fan will spin up when any thermal sensor reaches its configured threshold, it will go faster with higher temperature and slow down when temperature sinks. The fan will be turned off when the sensor(s) that triggered it report a temperature about 5C below threshold. All thresholds are configurable via sysctl:

```
machdep.adt7467c0.temp0 = 56
machdep.adt7467c0.temp1 = 85
machdep.adt7467c0.temp2 = 76
```

Every threshold corresponds to a temperature sensor, so with the ADM1030 there will be only two of them. Both chips use degree Celsius to specify temperature thresholds so that's what the sysctl interface uses too. Configuring a threshold below room temperature will essentially turn on the fan permanently, values above 85C will be rejected.

HISTORY

The **adt7467c** device appeared in NetBSD 4.0.

BUGS

The drivers have been tested with iBooks only so far. The ADT7467 supports more fan speed sensors but these are unused in iBooks.

NAME

adv — ConnectCom Solutions AdvanSys SCSI adapter driver

SYNOPSIS

```
adv* at pci? dev ? function ?
adv0 at isa? port ? irq ? drq ?
adv* at cardbus? function ?
scsibus* at adv?
```

DESCRIPTION

The **adv** driver supports the following AdvanSys SCSI host adapters

PCI bus

Connectivity Products:

ABP920	Bus-Master PCI (16 CDB)
ABP930	Bus-Master PCI (16 CDB) (note 1)
ABP930U	Bus-Master PCI Ultra (16 CDB)
ABP930UA	Bus-Master PCI Ultra (16 CDB)
ABP960	Bus-Master PCI MAC/PC (16 CDB) (note 2)
ABP960U	Bus-Master PCI MAC/PC Ultra (16 CDB) (note 2)

Notes:

1. This board has been sold by SIIG as the Fast SCSI Pro PCI.
2. This board has been sold by Iomega as a Jaz Jet PCI adapter.

Single Channel Products:

ABP940	Bus-Master PCI (240 CDB)
ABP940U	Bus-Master PCI Ultra (240 CDB)
ABP970	Bus-Master PCI MAC/PC (240 CDB)
ABP970U	Bus-Master PCI MAC/PC Ultra (240 CDB)
ABP940UW	Bus-Master PCI Ultra-Wide (240 CDB)

Multi Channel Products:

ABP950	Dual Channel Bus-Master PCI (240 CDB Per Channel)
ABP980	Four Channel Bus-Master PCI (240 CDB Per Channel)
ABP980U	Four Channel Bus-Master PCI Ultra (240 CDB Per Channel)

ISA bus

Connectivity Products:

ABP510/5150	Bus-Master ISA (240 CDB) (note 1)
ABP5140	Bus-Master ISA (16 CDB) (note 1) (note 2)
ABP5142	Bus-Master ISA with floppy (16 CDB) (note 3)

Notes:

1. This board has been shipped by HP with the 4020i CD-R drive. The board has no BIOS so it cannot control a boot device, but it can control any secondary SCSI device.
2. This board has been sold by SIIG as the i540 SpeedMaster.
3. This board has been sold by SIIG as the i542 SpeedMaster.

Single Channel Products:

ABP542	Bus-Master ISA with floppy (240 CDB)
ABP842	Bus-Master VL (240 CDB)

Dual Channel Products:

ABP852 Dual Channel Bus-Master VL (240 CDB Per Channel)

SEE ALSO

cd(4), ch(4), intro(4), isa(4), pci(4), scsi(4), sd(4), st(4)

http://www.connectcom.net/

http://www.siig.com/

HISTORY

The **adv** device driver appeared in NetBSD 1.4.

AUTHORS

Baldassare Dante Profeta ⟨dante@mclink.it⟩

NAME

adw — ConnectCom Solutions AdvanSys PCI Ultra Wide SCSI host adapter driver

SYNOPSIS

```
adw* at pci? dev ? function ?
scsibus* at adw?

options FAILSAFE
options SCSI_ADW_WDTR_DISABLE=mask
options SCSI_ADW_SDTR_DISABLE=mask
options SCSI_ADW_TAGQ_DISABLE=mask
```

DESCRIPTION

The **adw** driver provides support for the ADW (AdvanSys) ABP-940UW, ASB-3940UW, ASB-3940U2W SCSI host adapters.

The following kernel configuration options are available:

options FAILSAFE
Disables tagged command queuing, wide data transfers and synchronous data transfers for all SCSI devices controlled by the **adw** driver. By default, tagged command queuing, wide data transfers and synchronous data transfers are used if the SCSI devices support them.

The following options use a *mask* to specify which SCSI peripherals the option applies to. The *mask* is a 16 bit bitfield value. Each bit corresponds to a peripheral ID. The LSB (bit 0) corresponds to the peripheral with ID 0. The MSB (bit 15) corresponds to the peripheral with ID 15. The following features cannot be disabled for the host adapter, which by default has ID 7.

options SCSI_ADW_WDTR_DISABLE=mask
Disable WIDE data transfer for the peripherals specified by the mask value.

options SCSI_ADW_SDTR_DISABLE=mask
Disable SYNCHRONOUS data transfer for the peripherals specified by the mask value.

options SCSI_ADW_TAGQ_DISABLE=mask
Disable TAGGED COMMAND QUEUING for the peripherals specified by the mask value.

SEE ALSO

cd(4), ch(4), intro(4), scsi(4), sd(4), st(4), uk(4)

HISTORY

The **adw** device driver appeared in NetBSD 1.4.

AUTHORS

Baldassare Dante Profeta ⟨dante@NetBSD.org⟩.

NAME

ae — Ethernet driver for DP8390-based NuBus Ethernet boards

SYNOPSIS

ae* **at nubus?**

DESCRIPTION

The **ae** interface provides access to a 10 Mb/s Ethernet network via the National Semiconductor DP8390 Ethernet chip set. This set includes the DP83902.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **ae** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

HARDWARE

The **ae** interface is currently known to support the following NuBus cards:

Asante MacCon Series
Asante MacCon-i Series
Asante MacCON-III+ PDS card
Farallon EtherMac Series
Farallon EtherWave Series
Apple EtherTalk NB
3Com EtherLink NB

DIAGNOSTICS

ae%d at nubus%d: address %s, type %s %dKB memory. This is a normal autoconfiguration message noting the 6 byte physical Ethernet address of the adapter, its manufacturer, and how much buffer memory it has.

SEE ALSO

`arp`(4), `inet`(4), `netintro`(4), `sn`(4), `ifconfig`(8)

HISTORY

The **ae** interface first appeared in NetBSD 1.0.

NAME

afsc — A4091 low level SCSI interface

SYNOPSIS

```
afsc0 at zbus0
```

DESCRIPTION

The Amiga architecture uses a common machine independent scsi sub-system provided in the kernel source. The machine independent drivers that use this code access the hardware through a common interface. (see scsibus(4)) This common interface interacts with a machine dependent interface, such as **afsc**, which then handles the hardware specific issues.

The **afsc** interface handles things such as DMA and interrupts as well as actually sending commands, negotiating synchronous or asynchronous transfers and handling disconnect/reconnect of SCSI targets. The hardware that **afsc** uses is based on the NCR53c710 SCSI chip.

HARDWARE

The **afsc** interface supports the following Zorro III expansion cards:

A4091 Commodore SCSI adapter, manufacturer 514, product 84

DIAGNOSTICS

afsc%s: abort %s: dstat %02x, sstat0 %02x sbcl %02x The scsi operation %s was aborted due to error. Dstat, sstat and sbcl are registers within the NCR53c710 SCSI chip.

siop id %d reset0 The NCR53c710 SCSI chip has been reset and configure at id %d.

SIOP interrupt: %x sts %x msg %x sbcl %x The NCR53c710 SCSI chip has interrupted unexpectedly.

SIOP: SCSI Gross Error The NCR53c710 SCSI chip has indicated that it is confused.

SIOP: Parity Error The NCR53c710 SCSI chip has indicated that it has detected a parity error on the SCSI bus.

SEE ALSO

scsibus(4)

HISTORY

The **afsc** interface first appeared in NetBSD 1.0

NAME

agp — accelerated graphics port driver

SYNOPSIS

agp* **at pchb?**

DESCRIPTION

The **agp** driver provides machine-independent support for the accelerated graphics port (AGP) found on many PC-based and PCI systems. The AGP specification was designed by Intel.

The AGP chipset is positioned between the PCI-Host bridge and the graphics accelerator to provide a high-performance dedicated graphics bus for moving large amounts of data directly from host memory to the graphics accelerator. The specification currently supports a peak bandwidth of 528 MB/s. AGP uses a Graphics Address Remapping Table (GART) to provide a physically-contiguous view of scattered pages in host memory for DMA transfers.

The **agp** driver supports the following chipsets:

- ALI M1541 host-to-AGP bridge
- AMD 751 and 761 host-to-AGP bridges
- Intel 82810, 82810-DC100, 82810E, and 82815 SVGA controllers
- SiS 5591 host-to-AGP bridge
- VIA

The **agp** driver also provides an interface to user processes for use by X servers. A user process communicates to the device initially by means of `ioctl`(2) calls. The calls supported are:

`AGPIOC_INFO`
Get AGP information, setting the members in the *agp_info* structure as defined in <sys/agpio.h>:

```
typedef struct _agp_info {
        agp_version version;    /* version of the driver        */
        uint32_t bridge_id;    /* bridge vendor/device         */
        uint32_t agp_mode;     /* mode info of bridge          */
        off_t aper_base;        /* base of aperture             */
        size_t aper_size;       /* size of aperture             */
        size_t pg_total;        /* max pages (swap + system)    */
        size_t pg_system;       /* max pages (system)           */
        size_t pg_used;         /* current pages used           */
} agp_info;
```

`AGPIOC_ACQUIRE`
Acquire AGP.

`AGPIOC_RELEASE`
Release AGP.

`AGPIOC_SETUP`
Set up AGP, using the members in the *agp_setup* structure as defined in <sys/agpio.h>:

```
typedef struct _agp_setup {
        uint32_t agp_mode;     /* mode info of bridge          */
} agp_setup;
```

`AGPIOC_ALLOCATE`
Allocate AGP space, using and setting the members in the *agp_allocate* structure as defined in <sys/agpio.h>:

```
typedef struct _agp_allocate {
        int key;                /* tag of allocation           */
        size_t pg_count;        /* number of pages             */
        uint32_t type;         /* 0 == normal, other devspec   */
        paddr_t physical;       /* device specific (some devices
                                 * need a phys address of the
                                 * actual page behind the gatt
                                 * table)                      */
} agp_allocate;
```

`AGPIOC_DEALLOCATE`
Deallocate AGP space.

`AGPIOC_BIND`
Bind AGP space, using the members in the *agp_bind* structure as defined in <sys/agpio.h>:

```
typedef struct _agp_bind {
        int key;                /* tag of allocation           */
        off_t pg_start;         /* starting page to populate   */
} agp_bind;
```

`AGPIOC_UNBIND`
Unbind AGP space, using the members in the *agp_unbind* structure as defined in <sys/agpio.h>:

```
typedef struct _agp_unbind {
        int key;                /* tag of allocation           */
        uint32_t priority;     /* priority for paging out      */
} agp_unbind;
```

FILES

`/dev/agp?` AGP GART device special files
`/dev/agpgart` AGP GART device special file

EXAMPLES

This short code fragment is an example of opening the AGP device and performing some basic operations:

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/agpio.h>
#include <fcntl.h>
#include <err.h>

int
main(int argc, char **argv)
{
        int fd;
        agp_info info;
        agp_allocate alloc;
        agp_setup setup;
        agp_bind bind;
        agp_unbind unbind;

        fd = open("/dev/agp0", O_RDWR);
        if (fd < 0)
                err(1, "open");
```

```
if (ioctl(fd, AGPIOC_INFO, &info) < 0)
        err(2, "ioctl AGPIOC_INFO");

printf("version:      %u.%u\n", info.version.major,
    info.version.minor);

printf("id:           %x\n", info.bridge_id);
printf("mode:         %x\n", info.agp_mode);
printf("base:         %x\n", info.aper_base);
printf("size:         %uM\n", info.aper_size);
printf("total mem:    %u\n", info.pg_total);
printf("system mem:   %u\n", info.pg_system);
printf("used mem:     %u\n\n", info.pg_used);

setup.agp_mode = info.agp_mode;

if (ioctl(fd, AGPIOC_SETUP, &setup) < 0)
        err(3, "ioctl AGPIOC_SETUP");

if (ioctl(fd, AGPIOC_ACQUIRE, 0) < 0)
        err(3, "ioctl AGPIOC_ACQUIRE");

alloc.type = 0;
alloc.pg_count = 64;

if (ioctl(fd, AGPIOC_ALLOCATE, &alloc) < 0)
        err(4, "ioctl AGPIOC_ALLOCATE");

printf("alloc key %d, paddr %x\n", alloc.key, alloc.physical);
if (ioctl(fd, AGPIOC_INFO, &info) < 0)
        err(5, "ioctl AGPIOC_INFO");

bind.key = alloc.key;
bind.pg_start = 0x1000;

if (ioctl(fd, AGPIOC_BIND, &bind) < 0)
        err(6, "ioctl AGPIOC_BIND");

printf("used mem now:  %u\n\n", info.pg_used);

unbind.key = alloc.key;
unbind.priority = 0;

if (ioctl(fd, AGPIOC_UNBIND, &unbind) < 0)
        err(6, "ioctl AGPIOC_BIND");

if (ioctl(fd, AGPIOC_DEALLOCATE, &alloc.key) < 0)
        err(6, "ioctl AGPIOC_DEALLOCATE");

if (ioctl(fd, AGPIOC_RELEASE, 0) < 0)
        err(7, "ioctl AGPIOC_RELEASE");
```

```
        close(fd);

        printf("agp test successful\n");

        return 0;
}
```

SEE ALSO

`ioctl`(2), `pci`(4)

HISTORY

The **agp** driver first appeared in FreeBSD 4.1. It was adopted in NetBSD 1.6.

NAME

agr — link aggregation pseudo network interface driver

SYNOPSIS

```
pseudo-device agr
```

DESCRIPTION

The **agr** driver provides link aggregation functionality (a.k.a. L2 trunking or bonding).

It supports the IEEE 802.3ad Link Aggregation Control Protocol (LACP) and the Marker Protocol.

The **agr** driver supports the following link specific flags for `ifconfig`(8):

link0 Use the round-robin distribution algorithm. Don't use it unless you're really sure, because it violates the frame ordering rule.

-link0 Use the default distribution algorithm, which is based on the hash of DA/SA, TCI, and, if available, some upper layer protocol information like `ip`(4) DA/SA.

EXAMPLES

Create an **agr** interface, **agr0**, and attach **re0** and **re1** to it. (In other words, aggregate **re0** and **re1** so that they can be used as a single interface, **agr0**)

```
ifconfig agr0 create
ifconfig agr0 agrport re0
ifconfig agr0 agrport re1
```

Destroy an interface created in the above example.

```
ifconfig agr0 -agrport re0
ifconfig agr0 -agrport re1
ifconfig agr0 destroy
```

SEE ALSO

`ifconfig`(8)

STANDARDS

IEEE 802.3ad Aggregation of Multiple Link Segments

HISTORY

The **agr** driver first appeared in NetBSD 4.0.

AUTHORS

The **agr** driver was written by YAMAMOTO Takashi.

BUGS

The current implementation of the **agr** driver always attempts automatic configuration via LACP. There is no way to configure statically or use non IEEE 802.3 devices.

There is no way to configure LACP administrative variables, including system and port priorities. The current implementation of the **agr** driver always performs active-mode LACP and uses 0x8000 as system and port priorities.

The **agr** driver uses the MAC address of the first-added physical interface as the MAC address of the **agr** interface itself. Thus, removing the physical interface and using it for another purpose can result in non-unique MAC addresses.

The current implementation of the **agr** driver doesn't prevent unsafe operations like some ioctls against underlying physical interfaces. Such operations can result in unexpected behaviors, and are strongly discouraged.

There is no way to configure **agr** interfaces without attaching physical interfaces.

Physical interfaces being added to the **agr** interface shouldn't have any addresses except for link level address. Otherwise, the attempt will fail with EBUSY. Note that it includes an automatically assigned IPv6 link-local address.

NAME

aha — Adaptec 154x SCSI adapter driver

SYNOPSIS

```
aha0 at isa? port 0x330 irq ? drq ?
aha* at isapnp?
aha* at mca? slot ?
scsibus* at aha?
```

DESCRIPTION

The **aha** driver provides support for the following SCSI adapters:

Adaptec AHA-154xA
Adaptec AHA-154xB
Adaptec AHA-154xC
Adaptec AHA-154xCF
Buslogic BT-54x
Adaptec AHA-1640 (MCA)

SEE ALSO

cd(4), ch(4), intro(4), mca(4), scsi(4), sd(4), st(4)

NAME

ahb — Adaptec 1742 SCSI adapter driver

SYNOPSIS

```
ahb0 at eisa? slot ? irq ?
scsibus* at ahb?
```

DESCRIPTION

The **ahb** driver implements support for the following card:

Adaptec AHA-1742 EISA SCSI adaptor

SEE ALSO

cd(4), ch(4), intro(4), scsi(4), sd(4), st(4)

NAME

ahc — Adaptec VL/EISA/PCI/CardBus SCSI host adapter driver

SYNOPSIS

For VL cards:
ahc0 at isa? port ? irq ?

For EISA cards:
ahc* at eisa? slot ?

For PCI cards:
ahc* at pci? dev ? function ?

For CardBus cards:
ahc* at cardbus? function ?

To allow PCI adapters to use memory mapped I/O if enabled:
options AHC_ALLOW_MEMIO

Disable tagged queuing (avoids hangs on some hardware under load)
options AHC_NO_TAGS

Change the default SCSI id for cards without a SEEPROM (default 7):
options AHC_CARDBUS_DEFAULT_SCSI_ID=integer

For SCSI buses:
scsibus* at ahc?

DESCRIPTION

The **ahc** device driver supports SCSI controllers based on Adaptec AIC77xx and AIC78xx SCSI host adapter chips found on many motherboards as well as Adaptec SCSI controller cards.

Driver features include support for twin and wide buses, fast, ultra or ultra2 synchronous transfers depending on controller type, tagged queuing and SCB paging.

Memory mapped I/O can be enabled for PCI devices with the "AHC_ALLOW_MEMIO" configuration option. Memory mapped I/O is more efficient than the alternative, programmed I/O. Most PCI BIOSes will map devices so that either technique for communicating with the card is available. In some cases, usually when the PCI device is sitting behind a PCI->PCI bridge, the BIOS may fail to properly initialize the chip for memory mapped I/O. The typical symptom of this problem is a system hang if memory mapped I/O is attempted. Most modern motherboards perform the initialization correctly and work fine with this option enabled.

Per target configuration performed in the SCSI-Select menu, accessible at boot in non-EISA models, or through an EISA configuration utility for EISA models, is honored by this driver. This includes synchronous/asynchronous transfers, maximum synchronous negotiation rate, wide transfers, disconnection, the host adapter's SCSI ID, and, in the case of EISA Twin Channel controllers, the primary channel selection. For systems that store non-volatile settings in a system specific manner rather than a serial EEPROM directly connected to the aic7xxx controller, the BIOS must be enabled for the driver to access this information. This restriction applies to all EISA and many motherboard configurations.

Note that I/O addresses are determined automatically by the probe routines, but care should be taken when using a 284x (VESA local bus controller) in an EISA system. The jumpers setting the I/O area for the 284x should match the EISA slot into which the card is inserted to prevent conflicts with other EISA cards.

Performance and feature sets vary throughout the aic7xxx product line. The following table provides a comparison of the different chips supported by the **ahc** driver. Note that wide and twin channel features, although always supported by a particular chip, may be disabled in a particular motherboard or card design.

Chip	*MIPS*	*Bus*	*MaxSync*	*MaxWidth*	*SCBs*	*Features*
aic7770	10	EISA/VL	10MHz	16Bit	4	1
aic7850	10	PCI/32	10MHz	8Bit	3	
aic7860	10	PCI/32	20MHz	8Bit	3	
aic7870	10	PCI/32	10MHz	16Bit	16	
aic7880	10	PCI/32	20MHz	16Bit	16	
aic7890	20	PCI/32	40MHz	16Bit	16	3 4 5 6 7 8
aic7891	20	PCI/64	40MHz	16Bit	16	3 4 5 6 7 8
aic7892	20	PCI/64	80MHz	16Bit	16	3 4 5 6 7 8
aic7895	15	PCI/32	20MHz	16Bit	16	2 3 4 5
aic7895C	15	PCI/32	20MHz	16Bit	16	2 3 4 5 8
aic7896	20	PCI/32	40MHz	16Bit	16	2 3 4 5 6 7 8
aic7897	20	PCI/64	40MHz	16Bit	16	2 3 4 5 6 7 8
aic7899	20	PCI/64	80MHz	16Bit	16	2 3 4 5 6 7 8

1. Multiplexed Twin Channel Device - One controller servicing two buses.
2. Multi-function Twin Channel Device - Two controllers on one chip.
3. Command Channel Secondary DMA Engine - Allows scatter gather list and SCB prefetch.
4. 64 Byte SCB Support - SCSI CDB is embedded in the SCB to eliminate an extra DMA.
5. Block Move Instruction Support - Doubles the speed of certain sequencer operations.
6. ‘Bayonet’ style Scatter Gather Engine - Improves S/G prefetch performance.
7. Queuing Registers - Allows queuing of new transactions without pausing the sequencer.
8. Multiple Target IDs - Allows the controller to respond to selection as a target on multiple SCSI IDs.

HARDWARE

Supported SCSI controllers include:

Adaptec AHA-2742W EISA Fast Wide SCSI adapter

Adaptec AHA-274xAT EISA dual channel Fast SCSI adapter

Adaptec AHA-284x VL Fast SCSI adapter

Adaptec AHA-2910 PCI Fast SCSI adapter (no SCSI BIOS)

Adaptec AHA-2915 PCI Fast SCSI adapter (no SCSI BIOS)

Adaptec AHA-2920C PCI Fast SCSI adapter

Note: Adaptec AHA-2920/A which use the Future Domain’s chips are not supported by this driver.

Adaptec AHA-2930C PCI Ultra SCSI adapter

Adaptec AHA-2930U2 PCI Ultra2 Wide LVD SCSI adapter

Adaptec AHA-2940 PCI Fast SCSI adapter

Adaptec AHA-2940U PCI Ultra SCSI adapter

Adaptec AHA-2940AU PCI Ultra SCSI adapter

Adaptec AHA-2940UW PCI Ultra Wide SCSI adapter

Adaptec AHA-2940UW Dual PCI dual channel Ultra Wide SCSI adapter

Adaptec AHA-2940UW Pro PCI Ultra Wide SCSI adapter

Adaptec AHA-2940U2W PCI Ultra2 Wide LVD SCSI adapter

Adaptec AHA-2940U2B PCI Ultra2 Wide LVD SCSI adapter

Adaptec AHA-2944W PCI Fast Wide Differential SCSI adapter

Adaptec AHA-2944UW PCI Ultra Wide Differential SCSI adapter

Adaptec AHA-2950U2W

Adaptec AHA-2950U2B 64bit PCI Ultra2 Wide LVD SCSI adapter

Adaptec AHA-19160B PCI Ultra160 Wide LVD SCSI adapter

Adaptec ASC-29160 PCI Ultra160 Wide LVD SCSI adapter

Adaptec AHA-29160N PCI Ultra160 Wide LVD SCSI adapter

Adaptec AHA-29160B 64bit PCI Ultra160 Wide LVD SCSI adapter

Adaptec AHA-3940 PCI dual channel Fast SCSI adapter

Adaptec AHA-3940U PCI dual channel Ultra SCSI adapter

Adaptec AHA-3940AU PCI dual channel Ultra SCSI adapter

Adaptec AHA-3940UW PCI dual channel Ultra Wide SCSI adapter

Adaptec AHA-3940AUW PCI dual channel Ultra Wide SCSI adapter

Adaptec AHA-3940U2W PCI dual channel Ultra2 Wide LVD SCSI adapter

Adaptec AHA-3950U2 64bit PCI dual channel Ultra2 Wide LVD SCSI adapter

Adaptec AHA-3960 64bit PCI dual channel Ultra160 Wide LVD SCSI adapter

Adaptec AHA-3985 PCI dual channel Fast SCSI RAID adapter

Adaptec AHA-39160 64bit PCI dual channel Ultra160 Wide LVD SCSI adapter

Adaptec AHA-4944UW PCI quad channel PCI Ultra Wide Differential SCSI adapter

Other SCSI controllers based on the Adaptec AIC7770, AIC7850, AIC7860, AIC7870, AIC7880, AIC7890, AIC7891, AIC7892, AIC7895, AIC7896, AIC7897 and AIC7899 SCSI host adapter chips.

SCSI CONTROL BLOCKS (SCBs)

Every transaction sent to a device on the SCSI bus is assigned a ‘SCSI Control Block’ (SCB). The SCB contains all of the information required by the controller to process a transaction. The chip feature table lists the number of SCBs that can be stored in on-chip memory. All chips with model numbers greater than or equal to 7870 allow for the on chip SCB space to be augmented with external SRAM up to a maximum of 255 SCBs. Very few Adaptec controller configurations have external SRAM.

If external SRAM is not available, SCBs are a limited resource. Using the SCBs in a straight forward manner would only allow the driver to handle as many concurrent transactions as there are physical SCBs. To fully use the SCSI bus and the devices on it, requires much more concurrency. The solution to this problem is *SCB Paging*, a concept similar to memory paging. SCB paging takes advantage of the fact that devices usually disconnect from the SCSI bus for long periods of time without talking to the controller. The SCBs for disconnected transactions are only of use to the controller when the transfer is resumed. When the host queues another transaction for the controller to execute, the controller firmware will use a free SCB if one is available. Otherwise, the state of the most recently disconnected (and therefor most likely to stay disconnected) SCB is saved, via DMA, to host memory, and the local SCB reused to start the new transaction. This allows the controller to queue up to 255 transactions regardless of the amount of SCB space. Since the local SCB space serves as a cache for disconnected transactions, the more SCB space available, the less host bus

traffic consumed saving and restoring SCB data.

SEE ALSO

aha(4), ahb(4), ahd(4), cd(4), ch(4), intro(4), scsi(4), sd(4), st(4)

HISTORY

The **ahc** driver appeared in FreeBSD 2.0 and NetBSD 1.1.

AUTHORS

The **ahc** driver, the AIC7xxx sequencer-code assembler, and the firmware running on the aic7xxx chips was written by Justin T. Gibbs. NetBSD porting is done by Stefan Grefen, Charles M. Hannum, Michael Graff, Jason R. Thorpe, Pete Bentley, Frank van der Linden and Noriyuki Soda.

BUGS

Some Quantum drives (at least the Empire 2100 and 1080s) will not run on an AIC7870 Rev B in synchronous mode at 10MHz. Controllers with this problem have a 42 MHz clock crystal on them and run slightly above 10MHz. This confuses the drive and hangs the bus. Setting a maximum synchronous negotiation rate of 8MHz in the SCSI-Select utility will allow normal operation.

Target mode is not supported on NetBSD version of this driver.

NAME

ahcisata — AHCI 1.0 and 1.1 compliant SATA controllers driver

SYNOPSIS

ahcisata* **at pci? dev ? function ? flags 0x0000**

DESCRIPTION

The **ahcisata** driver supports the SATA controllers compliants to the Serial ATA Advanced Host Controller Interface Revision 1.0 or 1.1 specification, and provides the interface with the hardware for the ata(4) driver.

The **ahcisata** driver will only attach if the contoller has been put in AHCI mode by the BIOS; if the controller is in pciide-compatible mode, it will be handled by the appropriate driver (piixide(4) for Intel AHCI controllers).

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

Native Command Queueing is not yet supported.

NAME

ahd — Adaptec PCI/PCI-X Ultra320 SCSI host adapter driver

SYNOPSIS

For one or more PCI/PCI-X cards:

```
ahd* at pci? dev ? function ?
```

To compile in debugging code:

```
options AHD_DEBUG
options AHD_DEBUG_OPTS=<bitmask of options>
options AHD_REG_PRETTY_PRINT
```

For SCSI busses:

```
scsibus* at ahd?
```

DESCRIPTION

This driver provides access to the SCSI bus(ses) connected to Adaptec AIC79xx host adapter chips.

Driver features include support for narrow and wide busses, fast, ultra, ultra2, ultra160, and ultra320 synchronous transfers, packetized transfers, tagged queueing, and 512 SCBs.

The `AHD_DEBUG_OPTS` option is used to control which diagnostic messages are printed to the console when `AHD_DEBUG` is enabled. Logically OR the following bits together:

Value	*Function*
0x0001	Show miscellaneous information
0x0002	Show sense data
0x0004	Show Serial EEPROM contents
0x0008	Show bus termination settings
0x0010	Show host memory usage
0x0020	Show SCSI protocol messages
0x0040	Show mode pointer of the chip register window
0x0080	Show selection timeouts
0x0100	Show FIFO usage messages
0x0200	Show Queue Full status
0x0400	Show SCB queue status
0x0800	Show inbound packet information
0x1000	Show S/G list information
0x2000	Enable extra diagnostic code in the firmware

The `AHD_REG_PRETTY_PRINT` option compiles in support for human-readable bit definitions for each register that is printed by the debugging code. However, it also bloats the compiled size of the driver by approximately 215KB.

HARDWARE

The **ahd** driver supports the following:

- Adaptec AIC7901 host adapter chip
- Adaptec AIC7901A host adapter chip
- Adaptec AIC7902 host adapter chip
- Adaptec 29320 host adapter
- Adaptec 39320 host adapter
- Many motherboards with on-board SCSI support

SEE ALSO

ahc(4), cd(4), ch(4), intro(4), scsi(4) sd(4), ses(4), st(4)

HISTORY

The **ahd** driver first appeared in FreeBSD 4.7 and NetBSD 2.0.

AUTHORS

The **ahd** driver, the AIC7xxx sequencer-code assembler, and the firmware running on the aic79xx chips was written by Justin T. Gibbs. NetBSD porting is done by Pascal Renauld, Frank van der Linden, Jason Thorpe, and Allen Briggs. This manual page is based on the ahc(4) manual page.

NAME

ahsc — A3000 low level SCSI interface

SYNOPSIS

```
ahsc0 at mainbus0
```

DESCRIPTION

The Amiga architecture uses a common machine independent scsi sub-system provided in the kernel source. The machine independent drivers that use this code access the hardware through a common interface. (see scsibus(4)) This common interface interacts with a machine dependent interface, such as **ahsc**, which then handles the hardware specific issues.

The **ahsc** interface handles things such as DMA and interrupts as well as actually sending commands, negotiating synchronous or asynchronous transfers and handling disconnect/reconnect of SCSI targets. The hardware that **ahsc** uses is based on the WD33c93 SCSI chip.

DIAGNOSTICS

sbicwait TIMEO @%d with asr=x%x csr=x%x The 33c93 code (sbic) has been waiting too long for a SCSI chip operation to complete. %d is the line in the source file amiga/dev/sbic.c at which the SCSI chip timed-out. Asr and csr are status registers within the SCSI chip.

ahsc%d: abort %s: csr = 0x%02x, asr = 0x%02x A SCSI operation %s was aborted due to an error.

ahsc%d: csr == 0x%02i A error has occurred within the SCSI chip code.

ahsc%d: unexpected phase %d in icmd from %d The target described by ‘from %d’ has taken the SCSI bus into a phase which is not expected during polled IO.

ahsc%d: unexpected phase %d in icmd from %d The target described by ‘from %d’ has taken the SCSI bus into a phase which is not expected during DMA IO setup.

SEE ALSO

scsibus(4)

HISTORY

The **ahsc** interface first appeared in NetBSD 1.0

NAME

ai — AT&T StarLAN Ethernet interface driver

SYNOPSIS

```
ai0 at isa? port 0x360 iomem 0xd0000 irq 7
```

DESCRIPTION

The **ai** driver supports the following ISA bus NICs:

AT&T StarLAN 10
AT&T StarLAN Fiber

These cards are based on the Intel 82586 Ethernet controller chip.

SEE ALSO

ef(4), elmc(4), ifmedia(4), intro(4), isa(4), ix(4), ifconfig(8)

AUTHORS

Rafal K. Boni

NAME

aiboost — ASUS AI Booster hardware monitor

SYNOPSIS

aiboost* at acpi?

DESCRIPTION

The **aiboost** driver provides support for monitoring the hardware sensors in recent ASUS motherboards. The driver uses ACPI as the backend to fetch sensor values and descriptions and provides its data via the envsys(4) interface.

The **aiboost** driver typically has 7 sensors, depending on the motherboard and chipset:

Sensor	Units	Typical Use
CPU	uK	CPU Temperature
MB	uK	MB Temperature
VCC	uV DC	Core Voltage
+3.3V	uV DC	+3.3 Voltage
+5V	uV DC	+5 Voltage
+12V	uV DC	+12 Voltage
CPU	RPM	CPU Fan

SEE ALSO

envsys(4), envstat(8)

HISTORY

The **aiboost** driver first appeared in FreeBSD and then it was ported to NetBSD 5.0.

AUTHORS

The **aiboost** driver was written by Takanori Watanabe and Juan Romero Pardines, who adapted the code for NetBSD.

BUGS

It's possible to modify voltages via the ACPI methods in the DSDT, but for now the driver only reports the sensors' values. CPU Q-Fan is another thing that is typically specified in the ACPI namespace, and in the future we should handle this feature (to enable or disable automatic/manual fan mode).

NAME

aic — Adaptec AIC-6260 and AIC-6360 SCSI driver

SYNOPSIS

```
aic0 at isa? port 0x340 irq 12
aic* at isapnp?
aic* at pcmcia? function ?
scsibus* at aic?
```

DESCRIPTION

The **aic** driver provides support for the Adaptec AIC-6260 and AIC-6360 SCSI controller chips.

The PCMCIA SCSI host adapters and many ISA cards do not include boot ROMs and therefore cannot be used to connect the boot device.

HARDWARE

Cards supported by the **aic** driver include:

Adaptec 1502 ISA SCSI host adaptor

Adaptec 152x ISA SCSI host adaptor

Adaptec AHA-1520B ISAPNP SCSI host adaptor

Adaptec APA-1460 PCMCIA SCSI host adaptor

Creative Labs SoundBlaster ISA SCSI host adaptor, and compatibles

SEE ALSO

cd(4), ch(4), intro(4), scsi(4), sd(4), st(4)

NAME

aica — Yamaha AICA sound system audio device driver

SYNOPSIS

```
aica* at g2bus?
audio* at aica?
```

DESCRIPTION

The **aica** driver provides support for the Yamaha AICA sound system (64 channel PCM sound) chip.

SEE ALSO

g2bus(4)

HISTORY

The **aica** device driver appeared in NetBSD 2.0.

BUGS

AICA has 64 PCM channels, but this driver support only 2 channels (left & right). The remaining 62 channels are idle.

NAME

akbd — Apple Desktop Bus keyboard driver for wscons

SYNOPSIS

```
akbd* at obio?
wskbd* at akbd? console ?

options ALTXBUTTONS
options CAPS_IS_CONTROL
options FORCE_FUNCTION_KEYS
```

DESCRIPTION

This driver provides the wscons(4) driver with support for Apple Desktop Bus keyboards.

To work around the limited number of buttons found on most ADB mice, one can map key sequences to trigger mouse button events. To map Option+1, Option+2, Option+3 to mouse buttons 1, 2, and 3 respectively, add the following line to your kernel configuration file:

```
options ALTXBUTTONS
```

On macppc systems it is possible to tweak the keyboard driver to treat the caps lock key on an ADB keyboard as a control key. This requires special remapping because of ADB's strange emulation of a mechanically-locked key. To enable this code add the following line to your kernel configuration file:

```
options CAPS_IS_CONTROL
```

On macppc PowerBooks, several function keys double as "hot keys" (brightness, volume, eject) when the Fn modifier is held down. Mac OS X likes to reprogram the keyboard controller to send hot key events when Fn is not held down and send function key events when it is. To transform the non-keyboard "button" events back into function key events, place the following line in your kernel configuration file:

```
options FORCE_FUNCTION_KEYS
```

SUPPORTED HARDWARE

NetBSD is known to support the following ADB keyboards:

- On-board keyboards on PowerBook models
- Apple Standard Keyboard
- Apple Keyboard II
- Apple Extended Keyboard
- Apple Extended Keyboard II
- Apple Adjustable Keyboard
- Most third-party ADB keyboards are supported

SEE ALSO

adb(4), wscons(4), wskbd(4), wsconsctl(8)

BUGS

The number pad on extended keyboards does not send out the proper key codes for many applications.

The LEDs on extended keyboards are not functional under NetBSD.

NAME

amdpm — AMD768 Power Management Controller and AMD8111 System Management Controller

SYNOPSIS

amdpm* **at pci? dev ? function ?**

DESCRIPTION

The **amdpm** provides support for the AMD768 Power Management Controller and for the AMD8111 System Management Controller.

SEE ALSO

pci(4), rnd(4)

HISTORY

The **amdpm** driver appeared in NetBSD 2.0.

BUGS

Currently, this driver does nothing except collecting random numbers.

NAME

amdtemp — AMD CPU on-die digital thermal sensor

SYNOPSIS

```
amdtemp* at pci? dev ? function ?
```

DESCRIPTION

The **amdtemp** driver provides support for the on-die digital thermal sensor present on AMD K8, AMD Barcelona, AMD Phenoms and AMD Griffin CPUs.

These sensors were officially introduced in AMD K8 Revision F processors, and provide 0.5 degC accuracy. Precision was improved in Revision G chips, which provide two more bits for 0.25 degC steppings. Each core has two temperature sensors, and there are be up to two cores per CPU socket.

AMD Barcelona, AMD Phenom and AMD Griffin provide 0.125 degC accuracy and provide one temperature sensor for each CPU socket.

The **amdtemp** driver reports temperatures through the envsys(4) API.

Sensor	Units	Typical Use
CPUN sensor0	uK	cpuN temperature

SEE ALSO

envstat(4), powerd(8)

HISTORY

The **amdtemp** driver first appeared in OpenBSD 4.4 named kate(4). And then was ported to NetBSD 5.0. The driver has been renamed with support for newer AMD CPUs.

AUTHORS

The **amdtemp** driver was written by Constantine A. Murenin <cnst@openbsd.org> whilst at the University of Waterloo. It was adapted to NetBSD by Christoph Egger.

NAME

amhphy — Driver for the AMD 79c901 10BASE-T PHY

SYNOPSIS

```
amhphy* at mii? phy ?
```

DESCRIPTION

The **amhphy** driver supports the 10BASE-T portion of the AMD 79c901 HomePNA/10BASE-T PHY.

SEE ALSO

`ifmedia`(4), `intro`(4), `mii`(4), `ifconfig`(8)

NAME

amidisplaycc — wscons interface to amiga custom chips drivers

SYNOPSIS

```
amidisplaycc0 at mainbus0
wsdisplay0 at amidisplaycc0
```

DESCRIPTION

This device acts as an adapter between the wscons(4) framework and the Amiga custom chip driver functions. It exports the internal wsdisplay(4) interface and contains the necessary rendering functions to operate a text terminal with virtual screens. It uses the Amiga abstract graphic driver (grfabs) functions for the low-level display management.

Currently it does not support running X. It can however coexist well enough with grf0 to make possible running X the old way, but be warned, you cannot switch screens while in X and when quitting it, it seems to hang. Switching a screen then will bring up the text console. As always, we apologise for the inconvenience.

What it does support is hilite (bold), underline, reverse and foreground/background colors.

Virtual terminals and screen types

The number of virtual screens is limited only by the available chip memory.

Each virtual screen can have a different screen type. A screen type defines the following things: height and width in pixels, number of colors, and font size. The supported screen types are listed below.

- 80x64 - display size 640x512 with 8 colors, font size 8x8
- 80x51 - display size 640x510 with 8 colors, font size 8x10
- 80x50 - display size 640x400 with 8 colors, font size 8x8
- 80x40 - display size 640x400 with 8 colors, font size 8x10
- 80x32 - display size 640x512 with 8 colors, font size 8x16
- 80x31 - display size 640x248 with 8 colors, font size 8x8
- 80x25 - display size 640x400 with 8 colors, font size 8x16
- 80x24 - display size 640x192 with 8 colors, font size 8x8
- default - same as either 80x64 or 80x50, depending on the presence of GRF_NTSC and GRF_PAL in the kernel configuration.

The grfabs code determines the actual screen mode that is used. The config options GRF_NTSC, GRF_PAL, GRF_AGA, etc. determine what kind of chipsets/modes are available.

Fonts

Fonts of width 8 and any height are supported. Fonts can be compiled into the kernel by specifying "options FONT_[fontname]" in the configuration file, or loaded with the wsfontload(8) utility runtime.

SEE ALSO

wscons(4), wsdisplay(4), wsfontload(8), wsfont(9)

NAME

amr — AMI MegaRAID PCI-SCSI RAID driver

SYNOPSIS

```
amr* at pci? dev ? function ?
scsibus* at amr?
```

DESCRIPTION

The **amr** driver provides support for LSI (formerly American Megatrends) MegaRAID Express, Elite and Enterprise family RAID controllers for SCSI and SATA, including models relabeled and sold by Dell, Hewlett-Packard, and Intel. Supported controllers include:

- MegaRAID 320-1
- MegaRAID 320-2
- MegaRAID Series 418
- MegaRAID Enterprise 1200 (Series 428)
- MegaRAID Enterprise 1300 (Series 434)
- MegaRAID Enterprise 1400 (Series 438)
- MegaRAID Enterprise 1500 (Series 467)
- MegaRAID Enterprise 1600 (Series 471)
- MegaRAID Elite 1500 (Series 467)
- MegaRAID Elite 1600 (Series 493)
- MegaRAID Express 100 (Series 466WS)
- MegaRAID Express 200 (Series 466)
- MegaRAID Express 300 (Series 490)
- LSI MegaRAID SCSI 320-0X, 320-2X, 320-4X
- LSI MegaRAID SCSI 320-1E, 320-2E
- LSI MegaRAID SATA 300-6x, 300-8x
- Dell PERC
- Dell PERC 2/SC
- Dell PERC 2/DC
- Dell PERC 4/Di
- Dell PERC 4/SC
- Dell PERC 4e/Si
- HP NetRAID-1/Si
- HP NetRAID-3/Si
- HP Embedded NetRAID
- Intel SRCU42X
- Intel SRCU42E
- Intel SRMOBU42E
- Intel SRCS28X

DIAGNOSTICS

Driver initialisation/shutdown phase

amr%d: memory window not available
amr%d: I/O window not available

The PCI BIOS did not allocate resources necessary for the correct operation of the controller. The driver cannot attach to this controller.

amr%d: busmaster bit not set, enabling

The PCI BIOS did not enable busmaster DMA, which is required for the correct operation of the controller. The driver has enabled this bit and initialisation will proceed.

amr%d: can't allocate register window
amr%d: can't allocate interrupt
amr%d: can't set up interrupt
amr%d: can't allocate parent DMA tag
amr%d: can't allocate buffer DMA tag
amr%d: can't allocate scatter/gather DMA tag
amr%d: can't allocate s/g table
amr%d: can't allocate mailbox tag
amr%d: can't allocate mailbox memory

A resource allocation error occurred while initialising the driver; initialisation has failed and the driver will not attach to this controller.

amr%d: can't obtain configuration data from controller
amr%d: can't obtain product data from controller

The driver was unable to obtain vital configuration data from the controller. Initialisation has failed and the driver will not attach to this controller.

amr%d: can't establish configuration hook
amr%d: can't scan controller for drives

The scan for logical drives managed by the controller failed. No drives will be attached.

amr%d: device_add_child failed
amr%d: bus_generic_attach returned %d

Creation of the logical drive instances failed; attachment of one or more logical drives may have been aborted.

amr%d: flushing cache...

The controller cache is being flushed prior to shutdown or detach.

Operational diagnostics

amr%d: I/O beyond end of unit (%u,%d > %u)

A partitioning error or disk corruption has caused an I/O request beyond the end of the logical drive. This may also occur if FlexRAID Virtual Sizing is enabled and an I/O operation is attempted on a portion of the virtual drive beyond the actual capacity available.

amr%d: polled command timeout

An initialisation command timed out. The initialisation process may fail as a result.

amr%d: bad slot %d completed

The controller reported completion of a command that the driver did not issue. This may result in data corruption, and suggests a hardware or firmware problem with the system or controller.

amr%d: I/O error - %x

An I/O error has occurred.

SEE ALSO

`cd`(4), `ch`(4), `intro`(4), `pci`(4), `scsi`(4), `sd`(4), `st`(4), `amrctl`(8)

NAME

ams — Apple Desktop Bus mouse driver for wscons

SYNOPSIS

```
ams* at obio?
wsmouse* at ams?
```

DESCRIPTION

This driver provides the wscons(4) driver with support for Apple Desktop Bus mice.

SUPPORTED HARDWARE

NetBSD is known to support the following ADB mice:

- On-board trackpads and trackballs in PowerBook models
- Apple Desktop Bus Mouse
- Apple Desktop Bus Mouse II
- Interex ADB Mouse
- Logitech TrackMan
- Logitech MouseMan
- Microspeed Mouse Deluxe
- Mouse Systems A3 Mouse
- Most third-party ADB mice, trackballs, and trackpads are supported

DIAGNOSTICS

ams0 at adb0 addr 3: 1-button, 100 dpi mouse This is a typical autoconfiguration message noting the presence of the **ams** mouse.

SEE ALSO

adb(4), wscons(4), wsmouse(4), wsconsctl(8)

NAME

an — Aironet 4500/4800 and Cisco 340/350 series wireless network driver

SYNOPSIS

```
an* at pcmcia? function ?
an* at pci? dev ? function ?
an* at isapnp?
```

DESCRIPTION

The **an** driver provides support for Aironet Communications 4500/4800 and Cisco Aironet 340/350 series wireless network adapters. This includes the ISA, PCI and PCMCIA varieties. The 4500 series adapters operate at 1 and 2Mbps while the 4800 series and 340/350 series can operate at 1, 2, 5.5 and 11Mbps. The ISA, PCI and PCMCIA devices are all based on the same core PCMCIA modules and all have the same programming interface, however unlike the Lucent WaveLAN/IEEE cards, the ISA and PCI cards appear to the host as normal ISA and PCI devices and do not require any PCMCIA support.

The PCMCIA Aironet cards require PCMCIA support. ISA cards can either be configured to use ISA Plug and Play or to use a particular I/O address and IRQ by properly setting the DIP switches on the board. (The default switch setting is for plug and play.) The **an** driver has Plug and Play support and will work in either configuration, however when using a hard-wired I/O address and IRQ, the driver configuration and the NIC's switch settings must agree. PCI cards require no switch settings of any kind and will be automatically probed and attached.

All host/device interaction with the Aironet cards is via programmed I/O. The Aironet devices support 802.11 and 802.3 frames, power management, BSS (infrastructure) and IBSS (ad-hoc) operation modes. The **an** driver encapsulates all IP and ARP traffic as 802.11 frames, however it can receive either 802.11 or 802.3 frames. Transmit speed is selectable between 1Mbps, 2Mbps, 5.5Mbps, 11Mbps, or "auto" (the NIC automatically chooses the best speed).

By default, the **an** driver configures the Aironet card to join an access point with an SSID of null string. For ad-hoc mode, in which stations can communicate among each other without the aid of an access point, the driver must be set using `ifconfig`(8).

For more information on configuring this device, see `ifconfig`(8) and `ifmedia`(4).

HARDWARE

Cards supported by the **an** driver include:

Aironet 4500 Series

Aironet 4800 Series

Cisco Aironet 340 Series

Cisco Aironet 350 Series

DIAGNOSTICS

an%d: init failed The Aironet card failed to come ready after an initialization command was issued.

an%d: failed to allocate %d bytes on NIC The driver was unable to allocate memory for transmit frames in the NIC's on-board RAM.

an%d: device timeout The Aironet card failed to generate an interrupt to acknowledge a transmit command.

SEE ALSO

`arp`(4), `ifmedia`(4), `netintro`(4), `ifconfig`(8)

HISTORY

The **an** device driver first appeared in FreeBSD 4.0, and then in NetBSD 1.6.

AUTHORS

The **an** driver was written by Bill Paul ⟨wpaul@ee.columbia.edu⟩.

NAME

apecs — DECchip 21072/21071 Core Logic chipset

SYNOPSIS

```
apecs* at mainbus?
pci* at apecs?
```

DESCRIPTION

The **apecs** driver provides support for the DECchip 21072/21071 Core Logic chipset (PCI controller) found on the AlphaStation 200/250/255/400 systems, EB64+-family systems, AlphaServer 800/1000A systems and AlphaServer 1000 systems.

SEE ALSO

intro(4), mainbus(4), pci(4)

NAME

apm — Advanced Power Management pseudo-device driver

SYNOPSIS

```
apm0 at mainbus0
#include <machine/apmvar.h>
/dev/apm
```

DESCRIPTION

The **apm** driver provides support for the Advanced Power Management features of some i386 system BIOSes. The driver supports the Advanced Power Management (APM) BIOS Interface Specification (revision 1.2), published jointly by the Intel Corporation and the Microsoft Corporation.

The APM driver's behavior may be adjusted by specifying any of the following kernel configuration options:

`APM_NO_IDLE`
: Do not call the BIOS CPU idle function from the system idle loop. (Some systems will hang on certain device accesses, such as sound cards or floppy diskette drives, without this option)

`APM_V10_ONLY`
: Use only the APM revision 1.0 specification calls. (Some systems do not implement APM v1.1 very well, and generate weird events instead of the expected events when the system suspend key is pressed.)

`APM_NO_V12`
: Don't attach to the BIOS as APM v1.2 compliant device. (In case there are problems with v1.2 support.)

`APM_NO_STANDBY`
: Do not attempt to put the system into standby mode.

`APM_NO_POWEROFF`
: Do not attempt to turn off power when halting the system.

`APM_FORCE_64K_SEGMENTS`
: Force the length of the APM BIOS code and data segments to 64KB.

`APM_ALLOW_BOGUS_SEGMENTS`
: Allow the use of data segments which are in unexpected locations.

`APMDEBUG`
: Enable kernel printout of events received from the APM BIOS.

`APMCALLDEBUG`
: Enable kernel printout of every call to the APM BIOS (this is very noisy).

`APM_POWER_PRINT`
: Print power state on console at `APM_POWER_CHANGE` events. (Since it increases `syslogd`(8)'s activity, it may consume increased battery power. Some systems generate the events too frequently, and printing the status may disturb single-user operations.)

`APM_DISABLE_INTERRUPTS`
: Set this to zero if you don't want the kernel to disable interrupts before calling the BIOS. This is required for most IBM ThinkPads, and some other newer laptops. A good indication that you need this is that the machine hangs just after resuming from suspended state. It's unclear if doing this has negative effects on older BIOS, therefore it defaults to one (i.e interrupts are disabled).

If no processes are holding open file descriptors to the APM device, the driver will process the APM BIOS events itself. If a process has the device open for write, the driver defers all suspend and standby processing to the user process as long as there is sufficient queue space to store the event for the process. If the device is only open for read, the driver will report events but handle them itself.

The APM device may be opened by multiple readers but only one writer. Multiple readers may fetch the status with `ioctl`(2) without worrying about interference, but they must cooperate to share events as only a

single event queue is provided. The device may only be `select`(2)ed or manipulated with `ioctl`(2); `read`(2) and `write`(2) are not supported. The `ioctl`(2) calls supported are:

`APM_IOC_SUSPEND`
: Initiate an APM suspend mode. This is a deep sleep mode which powers down most devices. The device must be open for writing for this command to succeed.

`APM_IOC_STANDBY`
: Initiate an APM standby mode. This is a light sleep mode from which the system can quickly restore normal operation. The device must be open for writing for this command to succeed.

`APM_IOC_GETPOWER`
: Fetch the current power status into an *apm_power_info* structure.

```
struct apm_power_info {
        u_char battery_state;
        u_char ac_state;
        u_char battery_life;
        u_char spare1;
        u_int minutes_left;           /* estimate */
        u_int nbattery;
        u_int batteryid;
        u_int spare2[4];
};
```

The structure should be zeroed (except for *batteryid*) before being passed.

battery_state is one of `APM_BATT_HIGH`, `APM_BATT_LOW`, `APM_BATT_CRITICAL`, `APM_BATT_CHARGING`, or `APM_BATT_UNKNOWN`.

ac_state is one of `APM_AC_OFF`, `APM_AC_ON`, `APM_AC_BACKUP`, or `APM_AC_UNKNOWN`.

battery_life is the percentage estimated remaining normal battery life (or 0 if the BIOS cannot provide an estimate).

minutes_left is an estimated remaining lifetime (or 0 if the BIOS cannot provide an estimate).

nbattery is the number of batteries in the system. If the system is using APM v1.1 or earlier, nbattery will always return 0.

Batteries are numbered from a base of 1. If the passed value of *batteryid* is 0, the returned values will reflect the percentage remaining, minutes left, etc. of all of the system's batteries taken together. If the passed value of *batteryid* is nonzero, the return values will reflect the indicated battery's percentage remaining, minutes left, etc. It is an error to set *batteryid* to a value greater than that returned by *nbattery*. If the system is using APM v1.1 or earlier, individual batteries cannot be queried, and *nbattery* will always return 0. *batteryid* is always set to the passed value upon return.

`APM_IOC_NEXTEVENT`
: Fetch the next event from the APM BIOS into an *apm_event_info* structure. If no more events are ready, this will return `EAGAIN`.

```
struct apm_event_info {
      u_int type;
      u_int index;
      u_int spare[8];
};
```

type is one of the APM event types (APM_STANDBY_REQ through APM_SYS_STANDBY_RESUME). *index* is the ordinal event sequence number.

SEE ALSO

apmd(8)

REFERENCES

Advanced Power Management (APM) BIOS Interface Specification (Revision 1.1), Intel Corporation and Microsoft Corporation. Intel order number 241704-001; Microsoft part number 781-110-X01.

HISTORY

The **apm** pseudo-device driver appeared in NetBSD 1.3.

NAME

aps — ThinkPad Active Protection System accelerometer

SYNOPSIS

```
aps0 at isa? port 0x1600
```

DESCRIPTION

The **aps** driver provides support for several sensors found in some ThinkPad laptops.

The sensors currently exposed via the envsys(4) interface are:

Sensor	Units	Typical Use
X_ACCEL	Integer	X Acceleration
Y_ACCEL	Integer	Y Acceleration
X_VAR	Integer	Weighted X Acceleration?
Y_VAR	Integer	Weighted Y Acceleration?
Temp1	uK	Unknown
Temp2	uK	Unknown
Keyboard Active	Boolean	Keyboard activity
Mouse Active	Boolean	Mouse activity
Lid Open	Boolean	Lid state

SEE ALSO

envsys(4), envstat(8)

HISTORY

The **aps** driver first appeared in OpenBSD 3.8 and then was ported to NetBSD 5.0.

AUTHORS

The **aps** driver was written by Jonathan Gray ⟨jsg@openbsd.org⟩.

CAVEATS

The **aps** driver does not yet maintain state and subsequently does not take evasive action when it thinks the hard drive is in danger.

The Y axis on X40 and possibly other models seems to be inverted. It is unknown how to distinguish between different versions of the accelerometer to compensate for this in the driver at this time.

As IBM provides no documentation, it is not known what all the available sensors are used for.

NAME

arckbd, **arcwskbd**, **arcwsmouse** — Archimedes keyboard/mouse driver

SYNOPSIS

```
arckbd0         at ioc0 bank 0 offset 0x04
arcwskbd0       at arckbd0
wskbd0          at arcwskbd0
arcwsmouse0     at arckbd0
wsmouse0        at arcwsmouse0
```

DESCRIPTION

The **arckbd** driver controls the keyboard on systems supported by NetBSD/acorn26 and interfaces it to the **wscons** system. All communication between user programs and the **arckbd** driver goes through the **wskbd** and **wsmouse** drivers.

Because of the architecture of **wscons**, it is impossible to have a device at which both **wskbd** and **wsmouse** devices attach. The **arckbd** driver works around this by having dummy **arcwskbd** and **arcwsmouse** drivers which interpose between itself and the **wscons** drivers.

SEE ALSO

wscons(4), wskbd(4), wsmouse(4)

BUGS

The **arckbd** driver is capable of determining the layout of the keyboard, and noticing when it changes (if a different keyboard is connected, for instance), but doesn't know how to pass this information on to **wscons**.

NAME

arcmsr — Areca Technology Corporation SATA/SAS RAID controller

SYNOPSIS

arcmsr* **at pci? dev ? function ?**

DESCRIPTION

The **arcmsr** driver provides support for the PCI-X and PCI Express RAID controllers from Areca Technology Corporation:

- ARC-1110 PCI-X 4 Port SATA RAID Controller
- ARC-1110ML PCI-X 4 Port SATA RAID Controller
- ARC-1120 PCI-X 8 Port SATA RAID Controller
- ARC-1120ML PCI-X 8 Port SATA RAID Controller
- ARC-1130 PCI-X 12 Port SATA RAID Controller
- ARC-1130ML PCI-X 12 Port SATA RAID Controller
- ARC-1160 PCI-X 16 Port SATA RAID Controller
- ARC-1160ML PCI-X 16 Port SATA RAID Controller
- ARC-1170 PCI-X 24 Port SATA RAID Controller
- ARC-1200 Rev A PCI Express 2 Port SATA RAID Controller
- ARC-1202 PCI Express 2 Port SATA RAID Controller
- ARC-1210 PCI Express 4 Port SATA RAID Controller
- ARC-1220 PCI Express 8 Port SATA RAID Controller
- ARC-1230 PCI Express 12 Port SATA RAID Controller
- ARC-1230ML PCI Express 12 Port SATA RAID Controller
- ARC-1231ML PCI Express 12 Port SATA RAID Controller
- ARC-1260 PCI Express 16 Port SATA RAID Controller
- ARC-1260ML PCI Express 16 Port SATA RAID Controller
- ARC-1261ML PCI Express 16 Port SATA RAID Controller
- ARC-1280 PCI Express 24 Port SATA RAID Controller
- ARC-1280ML PCI Express 24 Port SATA RAID Controller
- ARC-1680 PCI Express 8 Port SAS RAID Controller
- ARC-1680LP PCI Express 8 Port SAS RAID Controller
- ARC-1680i PCI Express 8 Port SAS RAID Controller
- ARC-1680x PCI Express 8 Port SAS RAID Controller
- ARC-1681 PCI-X 8 Port SAS RAID Controller

These controllers support RAID levels 0, 1, 1E, 3, 5, 6, and JBOD using either SAS or SATA II drives.

arcmsr supports management and monitoring of the controller through the `bioctl`(8) and `envstat`(8) commands.

Please note, however, that to use some features that require special privileges, such as creating/removing hotspares, pass-through disks or RAID volumes will require to have the *password* disabled in the firmware; otherwise a *Permission denied* error will be reported by `bioctl`(8).

When a RAID 1 or 1+0 volume is created, either through the `bioctl`(8) command or controller's firmware, the volume won't be accessible until the initialization is done. A way to get access to the `sd`(4) device that corresponds to that volume without rebooting, is to issue the following command (once the initialization is finished):

```
$ scsictl scsibus0 scan any any
```

The **arcmsr** driver will also report to the kernel log buffer any error that might appear when handling firmware commands, such as used by the `bioctl`(8) command.

EVENTS

The **arcmsr** driver is able to send events to powerd(8) if a volume or any drive connected to the volume is not online. The *state-changed* event will be sent to the /etc/powerd/scripts/sensor_drive script when such condition happens.

SEE ALSO

intro(4), pci(4), scsi(4), sd(4), bioctl(8), envstat(8), powerd(8), scsictl(8)

HISTORY

The **arcmsr** driver first appeared in NetBSD 5.0.

AUTHORS

The **arcmsr** driver was originally written for OpenBSD by David Gwynne. It was ported to NetBSD and extended by Juan Romero Pardines.

NAME

aria — Sierra's Aria chipset audio device driver

SYNOPSIS

```
aria0 at isa? port 0xPPP irq I
aria0 at isa? port 0xPPP irq I flags 1
audio* at audiobus?
```

DESCRIPTION

The **aria** driver provides support for Sierra's Aria chipset, which is the basis of such cards as the Prometheus Aria 16, the Genoa AudioBahn 16 Pro, and some Diamond Sonic Sounds (Rev A5 and Rev B2).

The Sierra Aria chipset is full-duplex and is capable of 8- and 16- bit audio sample recording and playback at 7875 Hz, 11025 Hz, 15750 Hz, 22050 Hz, 31500 Hz, and 44100 Hz.

Valid I/O addresses are 0x280, 0x290, 0x2A0, and 0x2B0. The IRQ may be set to 10, 11, or 12.

The flags setting is necessary for the Prometheus Aria 16, as it needs to be specially configured at each cold boot by twiddling with the joystick port.

SEE ALSO

audio(4), isa(4)

HISTORY

The **aria** device driver appeared in NetBSD 1.4.

BUGS

DMA is not yet supported.

The flags option should not be necessary.

It is necessary to configure the port and irq.

NAME

arp — Address Resolution Protocol

SYNOPSIS

```
#include <netinet/if_ether.h>
```

DESCRIPTION

The Address Resolution Protocol (ARP) is a protocol used to dynamically map between Internet host addresses and Ethernet addresses. It is used by all the Ethernet interface drivers. It is not specific to Internet protocols or to Ethernet, but this implementation currently supports only that combination.

ARP caches Internet-Ethernet address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message which requires the mapping and broadcasts a message on the associated network requesting the address mapping. If a response is provided, the new mapping is cached and any pending message is transmitted. ARP will queue at most one packet while waiting for a response to a mapping request; only the most recently ‘‘transmitted’’ packet is kept. If the target host does not respond after several requests, the host is considered to be down for a short period (normally 20 seconds), allowing an error to be returned to transmission attempts during this interval. The error is `EHOSTDOWN` for a non-responding destination host, and `EHOSTUNREACH` for a non-responding router.

The ARP cache is stored in the system routing table as dynamically-created host routes. The route to a directly-attached Ethernet network is installed as a “cloning” route (one with the `RTF_CLONING` flag set), causing routes to individual hosts on that network to be created on demand. These routes time out periodically (normally 20 minutes after validated; entries are not validated when not in use). An entry for a host which is not responding is a “reject” route (one with the `RTF_REJECT` flag set).

ARP entries may be added, deleted or changed with the `arp`(8) utility. Manually-added entries may be temporary or permanent, and may be “published”, in which case the system will respond to ARP requests for that host as if it were the target of the request.

In the past, ARP was used to negotiate the use of a trailer encapsulation. This is no longer supported.

ARP watches passively for hosts impersonating the local host (i.e. a host which responds to an ARP mapping request for the local host’s address).

DIAGNOSTICS

duplicate IP address %x sent from ethernet address %x:%x:%x:%x:%x:%x. ARP has discovered another host on the local network which responds to mapping requests for its own Internet address with a different Ethernet address, generally indicating that two hosts are attempting to use the same Internet address.

SEE ALSO

`inet`(4), `route`(4), `arp`(8), `ifconfig`(8), `route`(8)

Plummer, D., "RFC 826", *An Ethernet Address Resolution Protocol*.

Leffler, S.J. and Karels, M.J., "RFC 893", *Trailer Encapsulations*.

NAME

artsata — Intel i31244 Serial ATA disk controller driver

SYNOPSIS

```
artsata* at pci? dev ? function ? flags 0x0000
options PCIIDE_I31244_DISABLEDMA
```

DESCRIPTION

The **artsata** driver supports the Intel i31244 Serial ATA and controllers, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **artsata** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the SATA controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

Early samples of the Intel i31244 Serial ATA controller revision 0 had a bug affecting DMA data transfers. Full production samples have been fixed, but have the same revision number. The PCIIDE_I31244_DISABLEDMA option can be used to disable DMA on the buggy revisions.

NAME

asc — Acorn SCSI I Card device interface

SYNOPSIS

```
asc0 at podulebus?
```

DESCRIPTION

The **asc** interface provides access to Acorn SCSI Card interfaces.

SEE ALSO

cosc(4), csc(4), oak(4), ptsc(4)

NAME

asc — TURBOchannel single-channel SCSI adapter

SYNOPSIS

```
asc* at ioasic? offset ?
asc* at tc? slot ? offset ?
asc* at tcds? chip?
scsibus* at asc?
```

DESCRIPTION

The asc driver provides support for the NCR 53c94-based SCSI host adapter and related TURBOchannel SCSI adapter option boards.

SEE ALSO

cd(4), ch(4), intro(4), ioasic(4), scsi(4), sd(4), st(4), tc(4), tcds(4)

NAME

asc — TURBOchannel single-channel SCSI adapter

SYNOPSIS

```
asc* at ioasic? offset ?
asc* at tc? slot ? offset ?
asc* at tcds? chip?
scsibus* at asc?
```

DESCRIPTION

The **asc** driver provides support for the NCR 53c94-based SCSI host adapter on the DECstation 5000 series, and for the PMAZ-AA and related TURBOchannel SCSI-adapter option boards. The **asc** is a medium-performance implementation of the SCSI-I common command set supporting synchronous and asynchronous SCSI devices. The driver provides no support for targets with multiple Logical Unit Numbers (LUNs).

SEE ALSO

cd(4), ch(4), intro(4), ioasic(4), scsi(4), sd(4), st(4), tc(4), tcds(4)

HISTORY

The **asc** driver first appeared in 4.4BSD.

NAME

asc — VAXstation 4000 SCSI controller

SYNOPSIS

```
asc* at vsbus? csr 0x200c0080  ( VS-4000/60 or VLC )
asc* at vsbus? csr 0x25000080  ( VS-4000/9x )
asc* at tc?  ( Not yet supported )
scsibus* at asc?
```

DESCRIPTION

The **asc** driver provides support for the NCR 53c94-based SCSI host adapter on the VAXstation 4000 series, and for the PMAZ-AA and related TURBOchannel SCSI adapter option boards (also on the VAXstation 4000 series models which include support for the TURBOchannel adapter).

CONFIGURATION

The **asc** driver supports the following **flags** for use in `config`(1) files:

bits 0-7: disable disconnect/reselect for the corresponding SCSI target
bits 8-15: disable synchronous negotiation for SCSI target

"Target" is synonymous with SCSI ID number.

Note that SCSI tape drives should be allowed to perform disconnect/reselect or performance will suffer.

SEE ALSO

`cd`(4), `ch`(4), `esp`(4), `intro`(4), `scsi`(4), `sd`(4), `st`(4)

HISTORY

The **asc** driver first appeared in NetBSD 1.5.

NAME

asp — first generation I/O subsystem

SYNOPSIS

```
asp*    at mainbus0
gsc*    at asp?
```

DESCRIPTION

Core bus controller and I/O subsystem as present on older HP 9000/700 workstations. The supported core bus controllers are those used in conjunction with PA7000, PA7100, and PA7150 CPUs and include:

- Core bus controller
- System Clock
- Interrupt Controller
- DMA Controller
- Real Time Clock Interface
- RAM and EEPROM controllers

MACHINES

An incomplete list of machines that use the ASP bus controller:

- 705, 710
- 715/{33,50,75}
- 725/{50,75}
- 720, 730, 750
- 735/*
- 742i
- 745i/{50,75}
- 747i/{50,75}
- 755/*

SEE ALSO

gsc(4), intro(4), io(4)

Hardball I/O Subsystem ERS, Revision 1.1, Hewlett-Packard, 30 September 1991.

HISTORY

The **asp** driver appeared in OpenBSD 2.4. It was ported to NetBSD 1.6 by Matthew Fredette.

NAME

ast — multiplexing serial communications interface

SYNOPSIS

```
ast0 at isa? port 0x1a0 irq 5
com2 at ast? slave ?
com3 at ast? slave ?
com4 at ast? slave ?
com5 at ast? slave ?
```

DESCRIPTION

The **ast** driver provides support for boards that multiplex together up to four EIA RS-232C (CCITT V.28) communications interfaces. Apparently the original maker of hardware using this multiplexing protocol was AST.

Each **ast** device is the master device for up to four **com** devices. The kernel configuration specifies these **com** devices as slave devices of the **ast** device, as shown in the synopsis. The slave ID given for each **com** device determines which bit in the interrupt multiplexing register is tested to find interrupts for that device. The port specification for the **ast** device is used to compute the base addresses for the **com** subdevices and the port for the interrupt multiplexing register.

FILES

`/dev/tty0?`

SEE ALSO

com(4)

HISTORY

The **ast** driver was written by Roland McGrath and placed into the public domain.

NAME

ata, **atabus** — AT attachment (ATA) bus driver

SYNOPSIS

```
atabus* at wdc? channel ?
atabus* at pciide? channel ?
```

DESCRIPTION

The **ata** driver provides basic low-level functions for the wd(4) and atapi(4) drivers, for hardware which provides direct access to the ATA registers.

SEE ALSO

atapi(4), intro(4), pciide(4), wd(4), wdc(4)

NAME

atalk — AppleTalk Protocol Family

SYNOPSIS

```
#include <sys/types.h>
#include <netatalk/at.h>
```

DESCRIPTION

The AppleTalk Protocol Family provides presentation layer support for the AppleTalk Datagram Delivery Protocol (DDP), using the SOCK_DGRAM socket type. In addition, access to in-kernel AppleTalk routing tables and network interface configurations is provided.

The AppleTalk Protocol Suite provides support for five kinds of physical media: LocalTalk (230kbps wire-or'd serial), Ethernet, FDDI, Token Ring, and asynchronous serial connections (using either AppleTalk Remote Access (ARA) or PPP). Currently, NetBSD's AppleTalk implementation supports Ethernet, FDDI, and Token Ring.

AppleTalk packets are encapsulated on the Ethernet using the EtherTalk Link Access Protocol (ELAP). Local network address resolution is handled using the AppleTalk Address Resolution Protocol (AARP). Neither of these protocols is exposed to user-mode applications.

ADDRESSING

AppleTalk addresses are three byte quantities, stored in network byte order. The include file ⟨netatalk/at.h⟩ defines the AppleTalk address format.

Sockets in the AppleTalk protocol family use the following address structure:

```
struct sockaddr_at {
        uint8_t sat_len;
        sa_family_t    sat_family;
        uint8_t sat_port;
        struct at_addr sat_addr;
        union {
                struct netrange r_netrange;
                char           r_zero[8];
        } sat_range;
};
```

The port of a socket may be set with bind(2). The node for bind(2) must always be ATADDR_ANYNODE: "this node". The net must be ATADDR_ANYNET. ATADDR_ANYNET corresponds to the machine's "primary" address (the first configured). The port of a socket and the primary address are returned with getsockname(2).

PROTOCOLS

The AppleTalk protocol family comprises the DDP datagram delivery protocol, AppleTalk Data Stream Protocol (ADSP), AppleTalk Echo Protocol (AEP), AppleTalk Filing Protocol (AFP), AppleTalk Session Protocol (ASP), AppleTalk Transaction Protocol (ATP), Name Binding Protocol (NBP), Printer Access Protocol (PAP), and Zone Information Protocol (ZIP).

DDP is implemented in the kernel as SOCK_DGRAM sockets in the AF_APPLETALK address family. NetBSD implements all other AppleTalk protocols using the Netatalk package. Netatalk implements all functions except for ADSP and an AFP client. AEP, NBP, and ZIP services are provided by the atalkd daemon. ASP and ATP services are provided by a user library. PAP and AFP services are provided by user programs and daemons.

SEE ALSO

`bind`(2), `getsockname`(2), `options`(4)

Gursharan S. Sidhu, Richard F. Andrews, and Alan B. Oppenheimer, *Inside AppleTalk, second edition*.

NAME

ataraid — software BIOS RAID

SYNOPSIS

```
pseudo-device ataraid
```

DESCRIPTION

The **ataraid** driver provides support for BIOS-based software RAID controllers. These are devices which have some simple support for several basic RAID levels (often RAID 0 and RAID 1), but which require software support to actually perform the RAID function. The BIOS support is largely just to create and recognize the array so that it may be a boot device.

The driver currently supports RAID formats from:

- Promise FastTrak
- Adaptec HostRAID (found in Intel 6300ESB)
- Via V-RAID (found in many VIA-based motherboards)

SEE ALSO

ld(4)

HISTORY

The **ataraid** driver first appeared in NetBSD 2.0.

AUTHORS

The **ataraid** driver was originally adapted from FreeBSD by Jason Thorpe ⟨thorpej@NetBSD.org⟩.

BUGS

Not all features of the software RAID are currently recognized or supported. For example, the Adaptec support doesn't recognize when a RAID 1 should be in a "building" state, and it does not do the right thing.

At least part of the reason for this is that the publically-available information on these formats is quite limited.

NAME

ate — Fujitsu MB86965A based Allied-Telesis Ethernet cards driver

SYNOPSIS

```
ate0 at isa? port 0x2a0 irq ?
ate* at mca? slot ?
```

DESCRIPTION

The **ate** driver supports Allied-Telesis ISA and MCA bus Ethernet adapters based on the Fujitsu MB86965A Ethernet controller. Supported boards include:

Allied-Telesis AT1700T/AT1700BT/AT1700FT/AT1700AT

Allied-Telesis AT1720T/AT1720BT/AT1720FT/AT1720AT

Allied-Telesis RE2001/RE2003/RE2005/RE2009

SEE ALSO

fmv(4), ifmedia(4), intro(4), isa(4), mbe(4), mca(4), ifconfig(8)

HISTORY

The **ate** driver appeared in NetBSD 1.4.

NAME

atf-test-case — generic description of test cases

DESCRIPTION

A *test case* is a piece of code that stress-tests a specific feature of the software. This feature is typically self-contained enough, either in the amount of code that implements it or in the general idea that describes it, to warrant its independent testing. Given this, test cases are very fine-grained, but they attempt to group similar smaller tests which are semantically related.

A test case is defined by three components regardless of the language it is implemented in: a header, a body and a cleanup routine. The *header* is, basically, a declarative piece of code that defines several properties to describe what the test case does and how it behaves. In other words: it defines the test case's *meta-data*, further described in the **Meta-data** section. The *body* is the test case itself. It executes all actions needed to reproduce the test, and checks for failures. This body is only executed if the abstract conditions specified by the header are met. The *cleanup routine* routine is a piece of code always executed after the body, regardless of the exit status of the test case. It can be used to undo side-effects of the test case. Note that almost all side-effects of a test case are automatically cleaned up by the library; this is explained in more detail in the rest of this document.

It is extremely important to keep the separation between a test case's header and body well-defined, because the header is *always* parsed, whereas the body is only executed when the conditions defined in the header are met and when the user specifies that test case.

At last, test cases are always contained into test programs. The test programs act as a front-end to them, providing a consistent interface to the user and several APIs to ease their implementation.

Results

A test case always exits with one of the following results:

passed — The test case was executed successfully.

skipped — The test case could not be executed because some preconditions were not met. This is not a failure because it can typically be resolved by adjusting the system to meet the necessary conditions. This is always accompanied by a *reason*, a message describing why the test was skipped.

failed — An error appeared during the execution of the test case. This is always accompanied by a *reason*, a message describing why the test failed.

Input/output

Test cases are free to print whatever they want to their `stdout`(4) and `stderr`(4) file descriptors. They are, in fact, encouraged to print status information as they execute to keep the user informed of their actions. This is specially important for long test cases.

Test cases will log their results to an auxiliary file, which is then collected by the test program they are contained in. The developer need not care about this as long as he uses the correct APIs to implement the test cases.

Meta-data

The following list describes all meta-data properties interpreted internally by ATF. You are free to define new properties in your test cases and use them as you wish.

descr — Type: textual. Required.

A brief textual description of the test case's purpose. Will be shown to the user in reports. Also good for documentation purposes.

ident — Type: textual. Required.

The test case's identifier. Must be unique inside the test program and should be short but descriptive.

require.arch — Type textual. Optional. Pp. A whitespace separated list of architectures that the test case can be run under without causing errors due to an architecture mismatch.

require.config — Type: textual. Optional.

A whitespace separated list of configuration variables that must be defined to execute the test case. If any of the required variables is not defined, the test case is *skipped*.

require.machine — Type textual. Optional. Pp. A whitespace separated list of machine types that the test case can be run under without causing errors due to a machine type mismatch.

require.progs — Type: textual. Optional.

A whitespace separated list of programs that must be present to execute the test case. These can be given as plain names, in which case they are looked in the user's PATH, or as absolute paths. If any of the required programs is not found, the test case is *skipped*.

require.user — Type: textual. Optional.

The required privileges to execute the test case. Can be one of 'root' or 'unprivileged'. If the requested privileges do not match the current user, the test case is *skipped*.

NOTE: In the future, it is expected that the test case will attempt to gain the necessary privileges on its own before failing. At the very least, lowering the privileges from the super-user to an unprivileged user will be supported.

timeout — Type: integral. Required; defaults to '300'. Specifies the maximum amount of time the test case can run. This is particularly useful because some tests can stall either because they are incorrectly coded or because they trigger an anomalous behavior of the program. It is not acceptable for these tests to stall the whole execution of the test program.

Can optionally be set to zero, in which case the test case has no run-time limit. This is discouraged.

Environment

Every time a test case is executed, several environment variables are cleared or reseted to sane values to ensure they do not make the test fail due to unexpected conditions. These variables are:

HOME — Set to the work directory's path.

LANG — Undefined.

LC_ALL — Undefined.

LC_COLLATE — Undefined.

LC_CTYPE — Undefined.

LC_MESSAGES — Undefined.

`LC_MONETARY`	Undefined.
`LC_NUMERIC`	Undefined.
`LC_TIME`	Undefined.
`TZ`	Undefined.

Work directories

The test program always creates a temporary directory and switches to it before running the test case's body. This way the test case is free to modify its current directory as it wishes, and the test program will be able to clean it up later on in a safe way, removing any traces of its execution from the system.

File creation mode mask (umask)

Test cases are always executed with a file creation mode mask (umask) of '0022'. The test case's code is free to change this during execution.

SEE ALSO

`atf-test-program`(1), `atf-formats`(5), `atf`(7)

NAME

ath — Atheros IEEE 802.11 driver

SYNOPSIS

```
ath* at pci? dev ? function ?
ath* at cardbus? function ?
```

DESCRIPTION

The **ath** driver provides support for wireless network adapters based on the Atheros AR5210, AR5211, AR5212, and AR5213 chips. Chip-specific support is provided by the Atheros Hardware Access Layer (HAL), which is currently available only in binary form for selected architectures.

Supported features include 802.11 and 802.3 frames, power management, BSS, IBSS, and host-based access point operation modes. All host/device interaction is via DMA.

The **ath** driver encapsulates all IP and ARP traffic as 802.11 frames, however it can receive either 802.11 or 802.3 frames. Transmit speed and operating mode is selectable depending on your hardware.

AR5210-based devices support 802.11a operation with transmit speeds of 6 Mbps, 9 Mbps, 12 Mbps, 18 Mbps, 24 Mbps, 36 Mbps, 48 Mbps, and 54 Mbps.

AR5211-based devices support 802.11a and 802.11b operation with transmit speeds as above for 802.11a operation and 1Mbps, 2Mbps, 5.5 Mbps and 11Mbps for 802.11b operation.

AR5212-based and AR5213-based devices support 802.11a, 802.11b, and 802.11g operation with transmit speeds appropriate to each.

All chips also support an Atheros Turbo Mode (TM) that operates in the 802.11a frequency range with 2x the transmit speeds. (This mode is, however, only interoperable with other Atheros-based devices.)

The actual transmit speed used is dependent on signal quality and the “rate control” algorithm employed by the driver. All chips support WEP encryption. AR5211 and AR5212 support the AES, TKIP, and Michael cryptographic operations required for WPA but at this time the driver does not support them. To enable encryption, use `ifconfig`(8).

By default, the **ath** driver configures the card for BSS operation (aka infrastructure mode). This mode requires the use of an access point (base station).

The **ath** driver also supports the standard IBSS point-to-point mode where stations can communicate amongst themselves without the aid of an access point.

The driver may also be configured to operate in hostap mode. In this mode a host may function as an access point (base station). Access points are different than operating in IBSS mode. They operate in BSS mode. They allow for easier roaming and bridge all Ethernet traffic such that machines connected via an access point appear to be on the local Ethernet segment.

The mode of operation is chosen by specifying the appropriate mediaopt value to ifconfig. The **-m** flag to ifconfig will list the available options.

For more information on configuring this device, see `ifconfig`(8).

Devices supported by the **ath** driver come in either CardBus or mini-PCI packages. Wireless cards in CardBus slots may be inserted and ejected on the fly.

The following cards are among those supported by the **ath** driver:

Card	*Chip*	*Bus*	*Standard*
3Com 3CRPAG175	AR5212	CardBus	a/b/g
Airlink AWLH4030	AR5212	PCI	b/g

Aztech WL830PC	AR5212	CardBus	b/g
Belkin F6D3000	AR5212	PCI	a/b/g
D-Link DWL-A520	AR5210	PCI	a
D-Link DWL-A650	AR5210	CardBus	a
D-Link DWL-AB650	AR5211	CardBus	a/b
D-Link DWL-AG520	AR5212	PCI	a/b/g
D-Link DWL-AG650	AR5212	CardBus	a/b/g
D-Link DWL-AG660	AR521?	CardBus	a/b/g
D-Link DWL-G520	AR5212	PCI	b/g
D-Link DWL-G650B	AR5212	CardBus	b/g
Elecom LD-WL54	AR5211	CardBus	a
Elecom LD-WL54AG	AR5212	CardBus	a/b/g
Fujitsu E5454	AR5212	CardBus	a/b/g
Fujitsu E5454	AR5212	CardBus	a/b/g
Fujitsu FMV-JW481	AR5212	CardBus	a/b/g
HP NC4000	AR5212	PCI	a/b/g
I/O Data WN-A54	AR5212	CardBus	a
I/O Data WN-AB	AR5212	CardBus	a/b
I/O Data WN-AG	AR5212	CardBus	a/b/g
Linksys WMP55AG	AR5212	PCI	a/b/g
Linksys WPC51AB	AR5211	CardBus	a/b
Linksys WPC55AG	AR5212	CardBus	a/b/g
NEC PA-WL/54AG	AR5212	CardBus	a/b/g
Netgear WAB501	AR5211	CardBus	a/b
Netgear WAG311	AR5212	PCI	a/b/g
Netgear WAG511	AR5212	CardBus	a/b/g
Netgear WG311	AR5212	PCI	b/g
Netgear WG511T	AR5212	CardBus	b/g
Orinoco 8470WD	AR5212	CardBus	a/b/g
Orinoco 8480	AR5212	CardBus	a/b/g
Planex GW-NS54AG	AR5212	CardBus	a/b/g
Proxim Skyline 4030	AR5210	CardBus	a
Proxim Skyline 4032	AR5210	PCI	a
Samsung SWL-5200N	AR5212	CardBus	a/b/g
SMC SMC2735W	AR5210	CardBus	a
Sony PCWA-C300S	AR5212	CardBus	b/g
Sony PCWA-C500	AR5210	CardBus	a
Sony PCWA-C700	AR5212	CardBus	a/b
Ubiquiti SRC	AR5213	CardBus	a/b/g

An up to date list can be found at `http://customerproducts.atheros.com/customerproducts`.

DIAGNOSTICS

ath%d: unable to attach hardware; HAL status %u The Atheros Hardware Access Layer was unable to configure the hardware as requested. The status code is explained in the HAL include file `contrib/sys/dev/ic/athhal.h`.

ath%d: failed to allocate descriptors: %d The driver was unable to allocate contiguous memory for the transmit and receive descriptors. This usually indicates system memory is scarce and/or fragmented.

ath%d: unable to setup a data xmit queue! The request to the HAL to setup the transmit queue for normal data frames failed. This should not happen.

ath%d: unable to setup a beacon xmit queue! The request to the HAL to setup the transmit queue for 802.11 beacon frames failed. This should not happen.

ath%d: 802.11 address: %s The MAC address programmed in the EEPROM is displayed.

ath%d: hardware error; resetting An unrecoverable error in the hardware occurred. Errors of this sort include unrecoverable DMA errors. The driver will reset the hardware and continue.

ath%d: rx FIFO overrun; resetting The receive FIFO in the hardware overflowed before the data could be transferred to the host. This typically occurs because the hardware ran short of receive descriptors and had no place to transfer received data. The driver will reset the hardware and continue.

ath%d: unable to reset hardware; hal status %u The Atheros Hardware Access Layer was unable to reset the hardware as requested. The status code is explained in the HAL include file `contrib/sys/dev/ic/athhal.h`. This should not happen.

ath%d: unable to start recv logic The driver was unable to restart frame reception. This should not happen.

ath%d: device timeout A frame dispatched to the hardware for transmission did not complete in time. The driver will reset the hardware and continue. This should not happen.

ath%d: bogus xmit rate 0x%x An invalid transmit rate was specified for an outgoing frame. The frame is discarded. This should not happen.

ath%d: ath_chan_set: unable to reset channel %u (%u MHz) The Atheros Hardware Access Layer was unable to reset the hardware when switching channels during scanning. This should not happen.

ath%d: unable to allocate channel table The driver was unable to allocate memory for the table used to hold the set of available channels.

ath%d: unable to collect channel list from hal A problem occurred while querying the HAL to find the set of available channels for the device. This should not happen.

ath%d: %s: %dM -> %dM (%d ok, %d err, %d retr) The driver's rate control algorithm changed the current rate for transmitting frames. This message is temporarily enabled for normal use to help in diagnosing and improving the rate control algorithm. The message indicates the new and old transmit rates and the statistics it used to decide on this change.

ath%d: failed to enable memory mapping The driver was unable to enable memory-mapped I/O to the PCI device registers. This should not happen.

ath%d: failed to enable bus mastering The driver was unable to enable the device as a PCI bus master for doing DMA. This should not happen.

ath%d: cannot map register space The driver was unable to map the device registers into the host address space. This should not happen.

ath%d: could not map interrupt The driver was unable to allocate an IRQ for the device interrupt. This should not happen.

ath%d: could not establish interrupt The driver was unable to install the device interrupt handler. This should not happen.

SEE ALSO

`arp`(4), `cardbus`(4), `ifmedia`(4), `netintro`(4), `pci`(4), `ifconfig`(8)

HISTORY

The **ath** device driver first appeared in FreeBSD 5.2. It was ported to NetBSD 2.0.

AUTHORS

The **ath** driver was originally written by Sam Leffler, and was ported to NetBSD by David Young.

CAVEATS

Different regulatory domains have different default channels for adhoc mode. See `ifconfig`(8) for information on how to change the channel. Different regulatory domains may not be able to communicate with each other with 802.11a as different regulatory domains do not necessarily have overlapping channels.

Revision A1 of the D-LINK DWL-G520 and DWL-G650 are based on an Intersil PrismGT chip and are not supported by this driver.

Revision v2 of the Netgear WG311 is based on a Texas Instruments ACX111 and is not supported by this driver.

Revision v3 of the Netgear WG311 is based on a Marvell Libertas 88W8335 and is not supported by this driver.

The HAL module is constructed from a binary component and operating system-dependent source code. Redistribution and use in source and binary forms, without modification, are permitted provided that the conditions set forth in `src/contrib/sys/dev/ic/athhal-COPYRIGHT` are observed.

BUGS

Performance in lossy environments is suboptimal. The algorithm used to select the rate for transmitted packets is very simplistic. There is no software retransmit; only hardware retransmit is used. Contributors are encouraged to replace the existing rate control algorithm with a better one (hint: all the information needed is available to the driver).

The driver does not fully enable power-save operation of the chip; consequently power use is suboptimal.

NAME

atppc — driver for AT-style parallel port chip sets

SYNOPSIS

```
atppc* at acpi?
atppc* at isa? port 0x378 irq 7 drq 3 flags 0x00
atppc* at isapnp?
atppc* at ofisa?
atppc* at pnpbios? index ?
atppc* at puc? port ?
options ATPPC_VERBOSE
options ATPPC_DEBUG
```

DESCRIPTION

atppc supports parallel ports and provides the low level support needed by higher level drivers such as ppbus(4). This driver attaches where the traditional NetBSD lpt(4) driver would ordinarily. It provides the data transport and chip set manipulation needed by higher driver layers, such as ppbus(4) and lpt(4). This driver is designed to be one of many possible implementations supporting machine independent parallel device support via ppbus(4).

IEEE 1284 support

atppc is intended to provide to data-link like services to higher level IEEE 1284 device drivers (such as ppbus(4)). **atppc** does not directly support IEEE 1284 features such as mode negotiation but rather provides the necessary infrastructure to allow a higher level driver to provide these services.

atppc does provide chip set manipulation, device handshakes (where appropriate), low-level error detection, and data transfer.

Supported data transfer modes

atppc supports the following data transfer modes: Centronics Compatible (Standard), Nibble, Byte (PS2), Fast Centronics, ECP, and EPP. Standard and Fast Centronics modes are write only, Nibble and Byte modes are read only, and ECP and EPP modes are bidirectional.

SEE ALSO

acpi(4), i386/pnpbios(4), isa(4), isapnp(4), lpt(4), ofisa(4), ppbus(4), puc(4)

HISTORY

The **atppc** driver is based on the **ppc** driver, which originally appeared in FreeBSD. The driver was ported over in NetBSD 2.0.

AUTHORS

This manual page is based on the FreeBSD **ppc** manual page. The information has been updated for the NetBSD port by Gary Thorpe.

BUGS

The FreeBSD driver includes support for some specific chip sets, specifically detection of some non-standard device I/O locations on the ISA bus. This support was not ported over to the NetBSD version of the driver yet.

NAME

attimer — AT Timer (8253) driver

SYNOPSIS

```
attimer*    at acpi?
attimer0    at isa?
```

DESCRIPTION

The **attimer** driver handles the so-called AT Timer device, initially found as chip model 8253. It is used as the main counter for the clock on the i386 port, but also offers control over the pitch of the PC speaker.

The **attimer** driver currently only implements the access to the ISA register "TIMER1" which controls the pitch of the PC speaker, and should be configured along with pcppi(4) to be of any actual use.

SEE ALSO

acpi(4), isa(4), pcppi(4)

NAME

atu — Atmel at76c50x 802.11B wireless network interfaces

SYNOPSIS

atu* **at uhub? port ?**

DESCRIPTION

The **atu** driver provides support for wireless network adapters based around the Atmel at76c503, at76c503a, at76c505, and at76c505a USB chipsets.

Supported features include 802.11 and 802.3 frames, power management, BSS, IBSS, ad-hoc, and host-based access point mode.

The **atu** driver encapsulates all IP and ARP traffic as 802.11 frames, however it can receive either 802.11 or 802.3 frames. Transmit speed is selectable between 1Mbps fixed, 2Mbps fixed, 2Mbps with auto fallback, 5.5Mbps, 8Mbps, or 11Mbps depending on your hardware.

Four different radio chipsets are used along with the device, each requiring a different firmware.

By default, the **atu** driver configures the card for BSS operation (aka infrastructure mode). This mode requires the use of an access point (base station).

For more information on configuring this device, see `ifconfig`(8).

The following devices are among those supported by the **atu** driver:

Acer Peripherals AWL400
AcerP AWL-300
Aincomm AWU2000B
Atmel 2662W-V4
Atmel BW002
Atmel DWL-120
Atmel WL-1330
Belkin F5D6050
Geowave GW-US11S
Linksys WUSB11
Linksys WUSB11-V28
Ovislink AirLive
SMC 2662W-AR

SEE ALSO

`arp`(4), `ifmedia`(4), `intro`(4), `netintro`(4), `usb`(4), `ifconfig`(8), `wiconfig`(8)

AUTHORS

The **atu** driver was written by Daan Vreeken and ported to OpenBSD by Theo de Raadt and David Gwynne. The OpenBSD driver was then ported to NetBSD by Jesse Off ⟨joff@NetBSD.org⟩.

NAME

atw — ADMtek ADM8211 802.11 wireless network driver

SYNOPSIS

```
atw* at cardbus? function ?
atw* at pci? dev ? function ?
```

DESCRIPTION

The **atw** driver supports PCI/CardBus 802.11b wireless adapters based on the ADMtek ADM8211.

The ADM8211 is a bus-mastering 802.11 Media Access Controller (MAC) which is derived from ADMtek's Tulip clones (see tlp(4)). It supports contention-free traffic (with an 802.11 Point Coordinator), 64/128-bit WEP encryption, and 802.11 power-saving. The ADM8211 integrates an RF3000 baseband processor (BBP) by RF Microdevices.

In a typical application, the ADM8211 is coupled with an RF front-end by RFMD and a Silicon Laboratories Si4126 RF/IF synthesizer.

With the ADM8211, the division of labor between the host and NIC is different than with firmware-based NICs such as an(4), awi(4), and wi(4). The ADM8211 is still responsible for real-time 802.11 functions such as sending ACK/RTS/CTS/ATIM frames, sending beacons, and answering CF polls from the access point, but the host takes responsibility for providing 802.11 functions such as scanning, association, and authentication. The host is also responsible for programming both the BBP and the RF/IF synthesizer.

atw contains incomplete support for the ADM8211's WEP encryption/decryption engine. **atw** does not yet support hardware WEP decryption, however, it will use the ADM8211's crypto engine to encrypt transmitted frames. Documentation from ADMtek claims that, in addition to the 4 128-bit shared WEP keys, the ADM8211 will store WEP key pairs for up to 20 peers. The documentation provides no details, hence **atw** does not support the 20 key-pairs.

The ADM8211 operates in 802.11 infrastructure mode (with an access point) and in 802.11 ad hoc mode (without an access point) at 1, 2, 5.5, and 11Mbps. ADMtek says that the ADM8211 cannot operate as an access point.

The operating mode is selected using the ifconfig(8) utility. For more information on configuring this device, see ifconfig(8) and ifmedia(4).

HARDWARE

Cards supported by the **atw** driver include:

D-Link DWL-650 Rev. ?? CardBus card
D-Link DWL-520 Rev. C1 PCI card
LanReady WP2000 PCI card
TrendNet TEW-221PC CardBus card
Xterasys XN2511B PCI card

DIAGNOSTICS

atw0: failed to tune channel %d The driver failed to tune the radio to a new channel. The radio remains tuned to the old channel.

atw0: atw_si4136_write wrote %08x, SYNCTL still busy The driver waited 100ms without seeing an indication that the ADM8211 had finished writing a register on the Si4126 RF/IF synthesizer.

atw0: device timeout The ADM8211 failed to generate an interrupt to acknowledge a transmit command.

SEE ALSO

`arp`(4), `cardbus`(4), `ifmedia`(4), `netintro`(4), `pci`(4), `ifconfig`(8)

ADMtek, http://www.admtek.com.tw.

Silicon Laboratories, http://www.silabs.com.

RF Microdevicdes, http://www.rfmd.com.

HISTORY

The **`atw`** device driver first appeared in NetBSD 2.0.

AUTHORS

The **`atw`** driver was written by David Young ⟨dyoung@NetBSD.org⟩. For features which the ADM8211 has in common with the DECchip 21x4x, code was liberally borrowed from `tlp`(4) by Jason Thorpe ⟨thorpej@NetBSD.org⟩.

BUGS

The author does not fully understand what processing the duration fields for the PLCP header and the 802.11 header undergo before they are applied to a transmitted frame. If the duration fields in transmitted frames are incorrect, the performance of your network may suffer.

The driver does not provide rate control when the media type is set to autoselect.

The driver lets you change to hostap mode, but it does not work and it probably never will.

The driver will sometimes complain that it cannot re-tune the radio because the transmit process has not gone idle. The author is investigating.

Many features are still missing, especially WEP decryption and 802.11 power-saving.

The ad hoc mode has not been rigorously tested. IBSSs with the same SSID may not coalesce, but this should not matter for most applications.

The driver is untested in the ad-hoc demo mode of Lucent WaveLAN cards.

The ADM8211 supports 802.11 power-saving, however, **`atw`** does not support it yet. For time-bounded service, the ADM8211 will interoperate with an access point which implements the 802.11 Point Coordination Function, however, this is also not supported.

Combinations of an ADM8211 with either an Intersil or a Marvell RF front-end are not supported.

NAME

atzsc — A2091 low level SCSI interface

SYNOPSIS

```
atzsc0 at zbus0
```

DESCRIPTION

The Amiga architecture uses a common machine independent scsi sub-system provided in the kernel source. The machine independent drivers that use this code access the hardware through a common interface. (see scsibus(4)) This common interface interacts with a machine dependent interface, such as **atzsc**, which then handles the hardware specific issues.

The **atzsc** interface handles things such as DMA and interrupts as well as actually sending commands, negotiating synchronous or asynchronous transfers and handling disconnect/reconnect of SCSI targets. The hardware that **atzsc** uses is based on the WD33c93 SCSI chip.

HARDWARE

The **atzsc** interface supports the following Zorro II expansion cards:

A2091 Commodore SCSI adapter, manufacturer 514, product 2 or product 3

DIAGNOSTICS

sbicwait TIMEO @%d with asr=x%x csr=x%x The 33c93 code (sbic) has been waiting too long for a SCSI chip operation to complete. %d is the line in the source file amiga/dev/sbic.c at which the SCSI chip timed-out. Asr and csr are status registers within the SCSI chip.

atzsc%d: abort %s: csr = 0x%02x, asr = 0x%02x A SCSI operation %s was aborted due to an error.

atzsc%d: csr == 0x%02i A error has occurred within the SCSI chip code.

atzsc%d: unexpected phase %d in icmd from %d The target described by ‘from %d’ has taken the SCSI bus into a phase which is not expected during polled IO.

atzsc%d: unexpected phase %d in icmd from %d The target described by ‘from %d’ has taken the SCSI bus into a phase which is not expected during DMA IO setup.

SEE ALSO

scsibus(4)

HISTORY

The **atzsc** interface first appeared in NetBSD 1.0

NAME

auacer — Acer Labs I/O Controller Hub integrated AC’97 audio device driver

SYNOPSIS

```
auacer* at pci? dev ? function ?
audio* at audiobus?
```

DESCRIPTION

The **auacer** device driver supports the M5455 integrated AC’97 audio controller of some Acer Labs I/O Controller Hub.

SEE ALSO

ac97(4), audio(4), pci(4)

HISTORY

The **auacer** device driver first appeared in NetBSD 3.0.

BUGS

No input supported (yet).

NAME

audio — device-independent audio driver layer

SYNOPSIS

```
#include <sys/audioio.h>
```

DESCRIPTION

The **audio** driver provides support for various audio peripherals. It provides a uniform programming interface layer above different underlying audio hardware drivers. The audio layer provides full-duplex operation if the underlying hardware configuration supports it.

There are four device files available for audio operation: `/dev/audio`, `/dev/sound`, `/dev/audioctl`, and `/dev/mixer`.

`/dev/audio` and `/dev/sound` are used for recording or playback of digital samples.

`/dev/mixer` is used to manipulate volume, recording source, or other audio mixer functions.

`/dev/audioctl` accepts the same `ioctl`(2) operations as `/dev/sound`, but no other operations.

In contrast to `/dev/sound` which has the exclusive open property `/dev/audioctl` can be opened at any time and can be used to manipulate the audio device while it is in use.

SAMPLING DEVICES

When `/dev/audio` is opened, it automatically directs the underlying driver to manipulate monaural 8-bit mu-law samples. In addition, if it is opened read-only (write-only) the device is set to half-duplex record (play) mode with recording (playing) unpaused and playing (recording) paused. When `/dev/sound` is opened, it maintains the previous audio sample mode and record/playback mode. In all other respects `/dev/audio` and `/dev/sound` are identical.

Only one process may hold open a sampling device at a given time (although file descriptors may be shared between processes once the first open completes).

On a half-duplex device, writes while recording is in progress will be immediately discarded. Similarly, reads while playback is in progress will be filled with silence but delayed to return at the current sampling rate. If both playback and recording are requested on a half-duplex device, playback mode takes precedence and recordings will get silence.

On a full-duplex device, reads and writes may operate concurrently without interference. If a full-duplex capable audio device is opened for both reading and writing it will start in half-duplex play mode; full-duplex mode has to be set explicitly.

On either type of device, if the playback mode is paused then silence is played instead of the provided samples, and if recording is paused then the process blocks in `read`(2) until recording is unpaused.

If a writing process does not call `write`(2) frequently enough to provide samples at the pace the hardware consumes them silence is inserted. If the `AUMODE_PLAY_ALL` mode is not set the writing process must provide enough data via subsequent write calls to “catch up” in time to the current audio block before any more process-provided samples will be played. If a reading process does not call `read`(2) frequently enough, it will simply miss samples.

The audio device is normally accessed with `read`(2) or `write`(2) calls, but it can also be mapped into user memory with `mmap`(2) (when supported by the device). Once the device has been mapped it can no longer be accessed by read or write; all access is by reading and writing to the mapped memory. The device appears as a block of memory of size *buffersize* (as available via `AUDIO_GETINFO` or `AUDIO_GETBUFINFO`). The device driver will continuously move data from this buffer from/to the audio hardware, wrapping around at the end of the buffer. To find out where the hardware is currently accessing data in the buffer the

AUDIO_GETIOFFS and AUDIO_GETOOFFS calls can be used. The playing and recording buffers are distinct and must be mapped separately if both are to be used. Only encodings that are not emulated (i.e. where AUDIO_ENCODINGFLAG_EMULATED is not set) work properly for a mapped device.

The audio device, like most devices, can be used in *select*, can be set in non-blocking mode and can be set (with a FIOASYNC ioctl) to send a SIGIO when I/O is possible. The mixer device can be set to generate a SIGIO whenever a mixer value is changed.

The following ioctl(2) commands are supported on the sample devices:

AUDIO_FLUSH
: This command stops all playback and recording, clears all queued buffers, resets error counters, and restarts recording and playback as appropriate for the current sampling mode.

AUDIO_RERROR (int)
: This command fetches the count of dropped input samples into its integer argument. There is no information regarding when in the sample stream they were dropped.

AUDIO_WSEEK (int)
: This command fetches the count of samples that are queued ahead of the first sample in the most recent sample block written into its integer argument.

AUDIO_DRAIN
: This command suspends the calling process until all queued playback samples have been played by the hardware.

AUDIO_GETDEV (audio_device_t)
: This command fetches the current hardware device information into the audio_device_t argument.

```
typedef struct audio_device {
        char name[MAX_AUDIO_DEV_LEN];
        char version[MAX_AUDIO_DEV_LEN];
        char config[MAX_AUDIO_DEV_LEN];
} audio_device_t;
```

AUDIO_GETFD (int)
: The command returns the current setting of the full duplex mode.

AUDIO_GETENC (audio_encoding_t)
: This command is used iteratively to fetch sample encoding names and format_ids into the input/output audio_encoding_t argument.

```
typedef struct audio_encoding {
       int index;      /* input: nth encoding */
       char name[MAX_AUDIO_DEV_LEN]; /* name of encoding */
       int encoding;   /* value for encoding parameter */
       int precision;  /* value for precision parameter */
       int flags;
#define AUDIO_ENCODINGFLAG_EMULATED 1 /* software emulation mode */
} audio_encoding_t;
```

To query all the supported encodings, start with an index field of 0 and continue with successive encodings (1, 2, ...) until the command returns an error.

AUDIO_SETFD (int)
: This command sets the device into full-duplex operation if its integer argument has a non-zero value, or into half-duplex operation if it contains a zero value. If the device does not support full-duplex operation, attempting to set full-duplex mode returns an error.

`AUDIO_GETPROPS (int)`

This command gets a bit set of hardware properties. If the hardware has a certain property the corresponding bit is set, otherwise it is not. The properties can have the following values:

`AUDIO_PROP_FULLDUPLEX`	the device admits full duplex operation.
`AUDIO_PROP_MMAP`	the device can be used with `mmap`(2).
`AUDIO_PROP_INDEPENDENT`	the device can set the playing and recording encoding parameters independently.

`AUDIO_GETIOFFS (audio_offset_t)`

`AUDIO_GETOOFFS (audio_offset_t)`

This command fetches the current offset in the input(output) buffer where the audio hardware's DMA engine will be putting(getting) data. It mostly useful when the device buffer is available in user space via the `mmap`(2) call. The information is returned in the audio_offset structure.

```
typedef struct audio_offset {
        u_int   samples;   /* Total number of bytes transferred */
        u_int   deltablks; /* Blocks transferred since last checked */
        u_int   offset;    /* Physical transfer offset in buffer */
} audio_offset_t;
```

`AUDIO_GETINFO (audio_info_t)`

`AUDIO_GETBUFINFO (audio_info_t)`

`AUDIO_SETINFO (audio_info_t)`

Get or set audio information as encoded in the audio_info structure.

```
typedef struct audio_info {
        struct  audio_prinfo play;   /* info for play (output) side */
        struct  audio_prinfo record; /* info for record (input) side */
         u_int  monitor_gain;                /* input to output mix */
        /* BSD extensions */
        u_int   blocksize;     /* H/W read/write block size */
        u_int   hiwat;         /* output high water mark */
        u_int   lowat;         /* output low water mark */
        u_int   _ispare1;
        u_int   mode;          /* current device mode */
#define AUMODE_PLAY   0x01
#define AUMODE_RECORD  0x02
#define AUMODE_PLAY_ALL 0x04  /* do not do real-time correction */
} audio_info_t;
```

When setting the current state with `AUDIO_SETINFO`, the audio_info structure should first be initialized with `AUDIO_INITINFO` (&info) and then the particular values to be changed should be set. This allows the audio driver to only set those things that you wish to change and eliminates the need to query the device with `AUDIO_GETINFO` or `AUDIO_GETBUFINFO` first.

The *mode* field should be set to `AUMODE_PLAY`, `AUMODE_RECORD`, `AUMODE_PLAY_ALL`, or a bitwise OR combination of the three. Only full-duplex audio devices support simultaneous record and playback.

hiwat and *lowat* are used to control write behavior. Writes to the audio devices will queue up blocks until the high-water mark is reached, at which point any more write calls will block until the queue is drained to the low-water mark. *hiwat* and *lowat* set those high- and low-water marks (in audio blocks). The default for *hiwat* is the maximum value and for *lowat* 75 % of *hiwat*.

blocksize sets the current audio blocksize. The generic audio driver layer and the hardware driver have the opportunity to adjust this block size to get it within implementation-required limits. Upon return from an AUDIO_SETINFO call, the actual blocksize set is returned in this field. Normally the *blocksize* is calculated to correspond to 50ms of sound and it is recalculated when the encoding parameter changes, but if the *blocksize* is set explicitly this value becomes sticky, i.e., it remains even when the encoding is changed. The stickiness can be cleared by reopening the device or setting the *blocksize* to 0.

```
struct audio_prinfo {
        u_int   sample_rate;    /* sample rate in samples/s */
        u_int   channels;       /* number of channels, usually 1 or 2 */
        u_int   precision;      /* number of bits/sample */
        u_int   encoding;       /* data encoding (AUDIO_ENCODING_* below) */
        u_int   gain;           /* volume level */
        u_int   port;           /* selected I/O port */
        u_long  seek;           /* BSD extension */
        u_int   avail_ports;    /* available I/O ports */
        u_int   buffer_size;    /* total size audio buffer */
        u_int   _ispare[1];
        /* Current state of device: */
        u_int   samples;        /* number of samples */
        u_int   eof;            /* End Of File (zero-size writes) counter */
        u_char  pause;          /* non-zero if paused, zero to resume */
        u_char  error;          /* non-zero if underflow/overflow occurred */
        u_char  waiting;        /* non-zero if another process hangs in open */
        u_char  balance;        /* stereo channel balance */
        u_char  cspare[2];
        u_char  open;           /* non-zero if currently open */
        u_char  active;         /* non-zero if I/O is currently active */
};
```

Note: many hardware audio drivers require identical playback and recording sample rates, sample encodings, and channel counts. The playing information is always set last and will prevail on such hardware. If the hardware can handle different settings the AUDIO_PROP_INDEPENDENT property is set.

The encoding parameter can have the following values:

AUDIO_ENCODING_ULAW	mu-law encoding, 8 bits/sample
AUDIO_ENCODING_ALAW	A-law encoding, 8 bits/sample
AUDIO_ENCODING_SLINEAR	two's complement signed linear encoding with the platform byte order
AUDIO_ENCODING_ULINEAR	unsigned linear encoding with the platform byte order
AUDIO_ENCODING_ADPCM	ADPCM encoding, 8 bits/sample
AUDIO_ENCODING_SLINEAR_LE	two's complement signed linear encoding with little endian byte order
AUDIO_ENCODING_SLINEAR_BE	two's complement signed linear encoding with big endian byte order
AUDIO_ENCODING_ULINEAR_LE	unsigned linear encoding with little endian byte order
AUDIO_ENCODING_ULINEAR_BE	unsigned linear encoding with big endian byte order

The *gain*, *port* and *balance* settings provide simple shortcuts to the richer mixer interface described below and are not obtained by AUDIO_GETBUFINFO. The gain should be in the range [AUDIO_MIN_GAIN, AUDIO_MAX_GAIN] and the balance in the range

[AUDIO_LEFT_BALANCE, AUDIO_RIGHT_BALANCE] with the normal setting at AUDIO_MID_BALANCE.

The input port should be a combination of:

AUDIO_MICROPHONE to select microphone input.
AUDIO_LINE_IN to select line input.
AUDIO_CD to select CD input.

The output port should be a combination of:

AUDIO_SPEAKER to select speaker output.
AUDIO_HEADPHONE to select headphone output.
AUDIO_LINE_OUT to select line output.

The available ports can be found in *avail_ports* (AUDIO_GETBUFINFO only).

buffer_size is the total size of the audio buffer. The buffer size divided by the *blocksize* gives the maximum value for *hiwat*. Currently the *buffer_size* can only be read and not set.

The *seek* and *samples* fields are only used by AUDIO_GETINFO and AUDIO_GETBUFINFO. *seek* represents the count of samples pending; *samples* represents the total number of bytes recorded or played, less those that were dropped due to inadequate consumption/production rates.

pause returns the current pause/unpause state for recording or playback. For AUDIO_SETINFO, if the pause value is specified it will either pause or unpause the particular direction.

MIXER DEVICE

The mixer device, /dev/mixer, may be manipulated with ioctl(2) but does not support read(2) or write(2). It supports the following ioctl(2) commands:

AUDIO_GETDEV (audio_device_t)
This command is the same as described above for the sampling devices.

AUDIO_MIXER_READ (mixer_ctrl_t)

AUDIO_MIXER_WRITE (mixer_ctrl_t)
These commands read the current mixer state or set new mixer state for the specified device *dev*. *type* identifies which type of value is supplied in the *mixer_ctrl_t* argument.

```
#define AUDIO_MIXER_CLASS  0
#define AUDIO_MIXER_ENUM   1
#define AUDIO_MIXER_SET    2
#define AUDIO_MIXER_VALUE  3
typedef struct mixer_ctrl {
        int dev;                        /* input: nth device */
        int type;
        union {
                int ord;                /* enum */
                int mask;               /* set */
                mixer_level_t value;    /* value */
        } un;
} mixer_ctrl_t;

#define AUDIO_MIN_GAIN  0
#define AUDIO_MAX_GAIN  255
typedef struct mixer_level {
        int num_channels;
```

```
        u_char level[8];                /* [num_channels] */
} mixer_level_t;
#define AUDIO_MIXER_LEVEL_MONO  0
#define AUDIO_MIXER_LEVEL_LEFT  0
#define AUDIO_MIXER_LEVEL_RIGHT 1
```

For a mixer value, the *value* field specifies both the number of channels and the values for each channel. If the channel count does not match the current channel count, the attempt to change the setting may fail (depending on the hardware device driver implementation). For an enumeration value, the *ord* field should be set to one of the possible values as returned by a prior AUDIO_MIXER_DEVINFO command. The type AUDIO_MIXER_CLASS is only used for classifying particular mixer device types and is not used for AUDIO_MIXER_READ or AUDIO_MIXER_WRITE.

AUDIO_MIXER_DEVINFO (mixer_devinfo_t)

This command is used iteratively to fetch audio mixer device information into the input/output mixer_devinfo_t argument. To query all the supported devices, start with an index field of 0 and continue with successive devices (1, 2, ...) until the command returns an error.

```
typedef struct mixer_devinfo {
        int index;              /* input: nth mixer device */
        audio_mixer_name_t label;
        int type;
        int mixer_class;
        int next, prev;
#define AUDIO_MIXER_LAST      -1
        union {
                struct audio_mixer_enum {
                        int num_mem;
                        struct {
                                audio_mixer_name_t label;
                                int ord;
                        } member[32];
                } e;
                struct audio_mixer_set {
                        int num_mem;
                        struct {
                                audio_mixer_name_t label;
                                int mask;
                        } member[32];
                } s;
                struct audio_mixer_value {
                        audio_mixer_name_t units;
                        int num_channels;
                        int delta;
                } v;
        } un;
} mixer_devinfo_t;
```

The *label* field identifies the name of this particular mixer control. The *index* field may be used as the *dev* field in AUDIO_MIXER_READ and AUDIO_MIXER_WRITE commands. The *type* field identifies the type of this mixer control. Enumeration types are typically used for on/off style controls (e.g. a mute control) or for input/output device selection (e.g. select recording input source from CD, line in, or microphone). Set types are similar to enumeration types but any combination

of the mask bits can be used.

The *mixer_class* field identifies what class of control this is. The (arbitrary) value set by the hardware driver may be determined by examining the *mixer_class* field of the class itself, a mixer of type `AUDIO_MIXER_CLASS`. For example, a mixer controlling the input gain on the line in circuit would have a *mixer_class* that matches an input class device with the name "inputs" (`AudioCinputs`), and would have a *label* of "line" (`AudioNline`). Mixer controls which control audio circuitry for a particular audio source (e.g. line-in, CD in, DAC output) are collected under the input class, while those which control all audio sources (e.g. master volume, equalization controls) are under the output class. Hardware devices capable of recording typically also have a record class, for controls that only affect recording, and also a monitor class.

The *next* and *prev* may be used by the hardware device driver to provide hints for the next and previous devices in a related set (for example, the line in level control would have the line in mute as its "next" value). If there is no relevant next or previous value, `AUDIO_MIXER_LAST` is specified.

For `AUDIO_MIXER_ENUM` mixer control types, the enumeration values and their corresponding names are filled in. For example, a mute control would return appropriate values paired with `AudioNon` and `AudioNoff`. For `AUDIO_MIXER_VALUE` and `AUDIO_MIXER_SET` mixer control types, the channel count is returned; the units name specifies what the level controls (typical values are `AudioNvolume`, `AudioNtreble`, `AudioNbass`).

By convention, all the mixer devices can be distinguished from other mixer controls because they use a name from one of the `AudioC`∗ string values.

FILES

`/dev/audio`
`/dev/audioctl`
`/dev/sound`
`/dev/mixer`

SEE ALSO

`audioctl`(1), `mixerctl`(1), `ioctl`(2), `ossaudio`(3), `midi`(4), `radio`(4)

ISA bus

`aria`(4), `ess`(4), `gus`(4), `guspnp`(4), `pas`(4), `sb`(4), `wss`(4), `ym`(4)

PCI bus

`auacer`(4), `auich`(4), `auixp`(4), `autri`(4), `auvia`(4), `azalia`(4), `clcs`(4), `clct`(4), `cmpci`(4), `eap`(4), `emuxki`(4), `esa`(4), `esm`(4), `eso`(4), `fms`(4), `neo`(4), `sv`(4), `yds`(4)

PCMCIA

`esl`(4)

TURBOchannel

`bba`(4)

USB

`uaudio`(4)

BUGS

If the device is used in `mmap`(2) it is currently always mapped for writing (playing) due to VM system weirdness.

NAME

audioamd — baseboard audio device driver

SYNOPSIS

```
audioamd0   at mainbus0
audioamd0   at obio0
audioamd0   at sbus0 slot ? offset ?
audio* at audioamd0
```

DESCRIPTION

The **audioamd** driver provides support for the baseboard audio found on sun4c and sun4m systems. The baseboard audio driver is based on the AMD 79c30 ISDN and audio interface. The interface is only capable of playing and recording 8kHz mu-law audio.

SEE ALSO

audio(4), sbus(4)

NAME

audiocs — Crystal Semiconductor CS4231 audio device driver

SYNOPSIS

```
audiocs0 at sbus0 slot ? offset ?
audiocs0 at ebus?
audio*   at audiobus?
```

DESCRIPTION

The **audiocs** driver provides support for the CS4231 audio devices on the EBus and SBus buses in sparc and sparc64 machines.

SEE ALSO

audio(4), ebus(4), sbus(4)

BUGS

Mixer “outputs” class (AudioCoutputs) is not yet supported.

NAME

aue — ADMtek AN986 and AN8511 Pegasus USB Ethernet driver

SYNOPSIS

```
aue*   at uhub?
ukphy* at mii?
```

HARDWARE

The **aue** driver supports the following adapters:

@Home USB 10/100
Abocom DSB650TX
Billionton Systems USB100
Compaq HNE-200
Corega FEther USB-TX
Corega FEther USB-TXS
D-Link DSB-650
D-Link DSB-650TX
D-Link DSB-650TX-PNA
I/O DATA USB ETTX
Hawking UF100
Kingston KNU101TX
LinkSys USB100TX
LinkSys USB100H1
LinkSys USB10TA
Melco Inc. LU-ATX
Microsoft MN110
SOHOware NUB100
SMC 2202USB
SMC 2206USB/ETH

DESCRIPTION

The **aue** driver provides support for USB Ethernet adapters based on the ADMtek AN986 Pegasus and AN8511 Pegasus II chipsets.

The Pegasus contains a 10/100 Ethernet MAC with MII interface and is designed to work with both Ethernet and HomePNA transceivers. Although designed to interface with 100Mbps peripherals, the existing USB standard specifies a maximum transfer speed of 12Mbps. Users should therefore not expect to actually achieve 100Mbps speeds with these devices.

The Pegasus supports a 64-bit multicast hash table, single perfect filter entry for the station address and promiscuous mode. Packets are received and transmitted over separate USB bulk transfer endpoints.

The **aue** driver supports the following media types:

autoselect — Enable automatic selection of the media type and options. The user can manually override the automatically selected mode by adding media options to the `/etc/rc.conf` file.

10baseT/UTP — Set 10Mbps operation. The *mediaopt* option can also be used to enable *full-duplex* operation. Not specifying *full duplex* implies *half-duplex* mode.

100baseTX — Set 100Mbps (fast Ethernet) operation. The *mediaopt* option can also be used to enable *full-duplex* operation. Not specifying *full duplex* implies *half-duplex* mode.

The **aue** driver supports the following media options:

full-duplex — Force full duplex operation. The interface will operate in half duplex mode if this media option is not specified.

For more information on configuring this device, see ifconfig(8).

DIAGNOSTICS

aue%d: watchdog timeout A packet was queued for transmission and a transmit command was issued, however the device failed to acknowledge the transmission before a timeout expired.

aue%d: no memory for rx list The driver failed to allocate an mbuf for the receiver ring.

SEE ALSO

arp(4), netintro(4), usb(4), ifconfig(8)

ADMtek AN986 data sheet, http://www.admtek.com.tw.

HISTORY

The **aue** device driver first appeared in FreeBSD 4.0, and in NetBSD 1.5.

AUTHORS

The **aue** driver was written by Bill Paul ⟨wpaul@ee.columbia.edu⟩.

NAME

auich — Intel I/O Controller Hub integrated AC'97 audio device driver

SYNOPSIS

```
auich* at pci? dev ? function ?
audio* at audiobus?
```

DESCRIPTION

The **auich** device driver supports the integrated AC'97 audio controller of the Intel I/O Controller Hub. Supported chipsets include the i82801AA (ICH), i82801AB (ICH0), i82801BA (ICH2), i82440MX, i82801CA (ICH3), i82801DB (ICH4), i82801EB (ICH5), i82801FB (ICH6), i82801GB/GR (ICH7), and Intel 6300ESB. The driver also supports SiS 7012, nForce MCP, nForce2 MCP-T, nForce3 MCP-T, nForce3 250 MCP-T, nForce4, and AMD 8111.

SEE ALSO

ac97(4), audio(4), pci(4)

HISTORY

The **auich** device driver appeared in NetBSD 1.6.

BUGS

The 'microphone' input DMA channel is not currently supported.

NAME

auixp — ATI IXP series integrated AC'97 audio device driver

SYNOPSIS

```
auixp* at pci? dev ? function ?
audio* at audiobus?
```

DESCRIPTION

The **auixp** device driver supports the integrated AC'97 audio controller of the ATP IXP series I/O controller hub. Supported models include all the chips that have IXP-200 base functionality for codec communication and DAC/ADC DMA support.

SEE ALSO

ac97(4), audio(4), pci(4)

HISTORY

The **auixp** device driver appeared in NetBSD 2.1.

BUGS

The 'SPDIF' support is still rudimentary and not supported other than through the codec. 'Quadrophonic' and 'Dolby 5.1' audio are supported but untested.

NAME

aupci — PCI bridge

SYNOPSIS

```
aupci* at aubus0 addr 0x14005000
pci* at aupci?
```

DESCRIPTION

The **aupci** driver provides support for the on-chip PCI bridge found in Alchemy Au1500 and Au1550 processors.

SEE ALSO

intro(4), pci(4)

HISTORY

The **aupci** driver first appeared in NetBSD 4.0.

BUGS

Additional board-specific support in the driver is required, so it might not be supported on all Au1500 and Au1550 designs.

DMA is not supported while the PCI is running in big endian mode.

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel. Devices which exist in the machine but are not configured into the kernel are not detected.

DIAGNOSTICS

CPU class not configured. You tried to boot NetBSD on a class of CPU type which it doesn't (or at least this compiled version of NetBSD doesn't) understand.

SEE ALSO

config(1), intro(4), boot(8)

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel. Devices which exist in the machine but are not configured into the kernel are not detected.

DIAGNOSTICS

Support for system type %d is not present in this kernel. You tried to boot NetBSD on a class of CPU type which it doesn't (or at least this compiled version of NetBSD doesn't) understand.

NetBSD does not yet support system type %d (%s). You tried to boot NetBSD on an unsupported system.

WARNING: can't figure what device matches %s. Unknown device.

PALcode not valid You tried to boot NetBSD on a system with an unknown version of the PALcode.

SEE ALSO

config(1), intro(4), boot(8)

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel. Devices which exist in the machine but are not configured into the kernel are not detected.

DIAGNOSTICS

CPU class not configured. You tried to boot NetBSD on a class of CPU type which it doesn't (or at least this compiled version of NetBSD doesn't) understand.

%s at mainbus0. An Amiga internal device '%s' was configured

not configured. If this line follows the `fd0 at fdc0` configuration line, this diagnostic indicates that a second floppy drive was detected, but was not configured into the kernel.

zbus0 at mainbus0 [mem 0x%x-0x%x]. The kernel is configuring AutoConfigured expansion boards. If any Zorro II memory was detected, the virtual address of the space reserved for DMA bounce buffers is printed.

%s at zbus0: pa 0x%x man/prod %d/%d. A Zorro expansion board was configured. `pa 0x%0x` is the physical address the board was configured at. `Man/prod %d/%d` is the manufacturer/product codes.

%s at zbus0: pa 0x%x man/prod %d/%d not configured. A Zorro expansion board located at `pa 0x%0x` with a manufacturer/product code `%d/%d` was found that is not configured into the kernel.

SEE ALSO

config(1), intro(4), boot(8)

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel.

Autoconfiguration on the HP300s is similar to that on the VAX, the primary difference is in the naming conventions. On the HP300, if devices exist which are not configured they will be ignored; if devices exist of unsupported type they will be ignored.

Normally, the system uses the disk from which it was loaded as the root filesystem. If that is not possible, a generic system will use 'rd0' if it exists. If such a system is booted with the RB_ASKNAME option (see reboot(2)), then the name of the root device is read from the console terminal at boot time, and any available device may be used.

DIAGNOSTICS

CPU type not configured. You tried to boot NetBSD on a CPU type which it doesn't (or at least this compiled version of NetBSD doesn't) understand.

hpibbus%d at sc%d, ipl %d. An HP-IB was found at sc%d (the select code) with ipl%d (interrupt priority level). NetBSD will call it hpibbus%d.

%s%d: %s.
%s%d at hpibbus%d, slave %d. An HP-IB disk or tape controller was found. For disks %s%d will look like 'rd0', for tapes like 'ct0'. The '%s' in the first line will be a product type like ''7945A'' or ''9144''. The slave number comes from the address select switches on the drive.

grf0 csr 0x560000
grf%d at sc%d A bit mapped display was found either at the ''internal'' address (first case) or at some ''external'' select code (second case). If it exists, the internal display will always be unit 0.

%s%d at sc%d, ipl %d flags %d Another peripheral controller was found at the indicated select code and with indicated interrupt priority level. '%s' will be one of com(4) (single-port serial interfaces), dcm(4) (four-port serial interfaces), or le(4) (LAN cards). The slave number comes from the address select switches on the interface card.

SEE ALSO

config(1), intro(4), boot(8)

4.3BSD for the HP300, in the distribution documentation package.

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel. Devices which exist in the machine but are not configured into the kernel are not detected.

DIAGNOSTICS

CPU class not configured. You tried to boot NetBSD on a class of CPU type which it doesn't (or at least this compiled version of NetBSD doesn't) understand.

SEE ALSO

config(1), intro(4), boot(8)

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel. Devices which exist in the machine but are not configured into the kernel are usually not detected. The exception to this is the case of NuBus expansion boards. All NuBus slots are probed, and information about any cards found is reported even if the card is not currently configured.

DIAGNOSTICS

%s%d at obio0: %s. The system is configuring an onboard I/O controller.

addr %x not configured. An onboard I/O controller was encountered that is not configured into the kernel.

%s at nubus%d slot %x. The system is configuring a NuBus expansion card in slot %x.

%s at nubus%d slot %x: %s (Vendor: %s, Part: %s Type: %x %x %x %x) not configured. A NuBus expansion card was encountered that is not configured into the kernel. 'Vendor' is the manufacturer of the board, 'Part' is the name of the board, and 'Type' is the Apple-defined type.

%s: channel %d not configured. A serial device channel was encountered that is not configured into the kernel.

%s at %s not configured. A device which was configured into the kernel was unable to properly initialize itself.

SEE ALSO

config(1), intro(4), boot(8)

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel. Devices which exist in the machine but are not configured into the kernel are not detected.

SEE ALSO

config(1), intro(4), boot(8)

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel. Devices which exist in the machine but are not configured into the kernel are not detected.

DIAGNOSTICS

Sorry, the kernel isn't configured for this model. You tried to boot a NetBSD kernel which was not configured to support your MVME board.

Sorry, NetBSD doesn't support this model yet. You tried to boot NetBSD on an MVME board which is not yet supported.

SEE ALSO

config(1), intro(4), boot(8)

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel. Devices which exist in the machine but are not configured into the kernel are not detected.

DIAGNOSTICS

CPU type (0x%x) not supported. You tried to boot NetBSD on a type of CPU type which it doesn't (or at least this compiled version of NetBSD doesn't) understand.

SEE ALSO

config(1), intro(4), boot(8)

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel. Devices which exist in the machine but are not configured into the kernel are not detected.

DIAGNOSTICS

CPU class not configured. You tried to boot NetBSD on a class of CPU type which it doesn't (or at least this compiled version of NetBSD doesn't) understand.

SEE ALSO

config(1), intro(4), boot(8)

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel. Devices which exist in the machine but are not configured into the kernel are not detected.

DIAGNOSTICS

CPU class not configured. You tried to boot NetBSD on a class of CPU type which it doesn't (or at least this compiled version of NetBSD doesn't) understand.

SEE ALSO

config(1), intro(4), boot(8)

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel. Devices which exist in the machine but are not configured into the kernel are not detected.

DIAGNOSTICS

CPU class not configured. You tried to boot NetBSD on a class of CPU type which it doesn't (or at least this compiled version of NetBSD doesn't) understand.

SEE ALSO

config(1), intro(4), boot(8)

NAME

autoconf — diagnostics from the autoconfiguration code

DESCRIPTION

When NetBSD bootstraps it probes the innards of the machine on which it is running and locates controllers, drives, and other devices. Each item found is recorded on the console. This procedure is driven by a system configuration table which is processed by config(1) and compiled into each kernel.

On the VAX, devices in NEXUS slots are normally noted, thus memory controllers, UNIBUS and MASSBUS adaptors. Devices which are not supported which are found in NEXUS slots are noted also. The Q-bus on the MICROVAX is configured in the same way as the UNIBUS.

MASSBUS devices are located by a very deterministic procedure since MASSBUS space is completely probeable. If devices exist which are not configured they will be silently ignored; if devices exist of unsupported type they will be noted.

UNIBUS devices are located by probing to see if their control-status registers respond. If not, they are silently ignored. If the control status register responds but the device cannot be made to interrupt, a diagnostic warning will be printed on the console and the device will not be available to the system.

Normally, the system uses the disk from which it was loaded as the root filesystem. If that is not possible, a generic system will pick its root device as the "best" available device (MASSBUS disks are better than SMD UNIBUS disks are better than RK07s; the device must be drive 0 to be considered). If such a system is booted with the RB_ASKNAME option (see reboot(2)), then the name of the root device is read from the console terminal at boot time, and any available device may be used.

DIAGNOSTICS

cpu type %d not configured. You tried to boot NetBSD on a CPU type which it doesn't (or at least this compiled version of NetBSD doesn't) understand.

mba%d at tr%d. A MASSBUS adapter was found in tr%d (the NEXUS slot number). NetBSD will call it mba%d.

%d mba's not configured. More MASSBUS adapters were found on the machine than were declared in the machine configuration; the excess MASSBUS adapters will not be accessible.

uba%d at tr%d. A UNIBUS adapter was found in tr%d (the NEXUS slot number). NetBSD will call it uba%d.

dr32 unsupported (at tr %d). A DR32 interface was found in a NEXUS, for which NetBSD does not have a driver.

ci unsupported (at tr %d). A CI interface was found in a NEXUS, for which NetBSD does not have a driver.

mcr%d at tr%d. A memory controller was found in tr%d (the NEXUS slot number). NetBSD will call it mcr%d.

5 mcr's unsupported. NetBSD supports only 4 memory controllers per CPU.

mpm unsupported (at tr%d). Multi-port memory is unsupported in the sense that NetBSD does not know how to poll it for ECC errors.

%s%d at mba%d drive %d. A tape formatter or a disk was found on the MASSBUS; for disks %s%d will look like "hp0", for tape formatters like "ht1". The drive number comes from the unit plug on the drive or in the TM formatter (*not* on the tape drive; see below).

%s%d at %s%d slave %d. (For MASSBUS devices). Which would look like "`tu0 at ht0 slave 0`", where "`tu0`" is the name for the tape device and "`ht0`" is the name for the formatter. A tape slave was found on the tape formatter at the indicated drive number (on the front of the tape drive). UNIX will call the device, e.g., "`tu0`".

%s%d at uba%d csr %o vec %o ipl %x. The device `%s%d`, e.g. "`dz0`" was found on `uba%d` at control-status register address '`%o`' and with device vector '`%o`'. The device interrupted at priority level '`%x`'.

%s%d at uba%d csr %o zero vector. The device did not present a valid interrupt vector, rather presented 0 (a passive release condition) to the adapter.

%s%d at uba%d csr %o didn't interrupt. The device did not interrupt, likely because it is broken, hung, or not the kind of device it is advertised to be.

%s%d at %s%d slave %d. (For UNIBUS devices). Which would look like "`up0 at sc0 slave 0`", where "`up0`" is the name of a disk drive and "`sc0`" is the name of the controller. Analogous to MASSBUS case.

SEE ALSO

`config`(1), `intro`(4), `boot`(8)

HISTORY

The **`autoconf`** feature appeared in 4.1BSD.

NAME

autri — Trident 4DWAVE-DX/NX, SiS 7018, ALi M5451 audio device driver

SYNOPSIS

```
autri* at pci? dev ? function ?
audio* at audiobus?
midi*  at autri?
```

DESCRIPTION

The **autri** device driver supports the AC'97 audio controller found in Trident 4DWAVE-DX/NX, SiS 7018 and ALi M5451.

SEE ALSO

ac97(4), audio(4), midi(4), pci(4)

HISTORY

The **autri** device driver appeared in NetBSD 1.6.

NAME

auvia — VIA VT82C686A/VT8233/VT8235/VT8237 integrated AC’97 audio device driver

SYNOPSIS

```
auvia* at pci? dev ? function ?
audio* at audiobus?
```

DESCRIPTION

The **auvia** device driver supports the integrated AC’97 audio controller of the VIA Technologies VT82C686A/VT8233/VT8235/VT8237 Southbridge chip found on some motherboards.

SEE ALSO

ac97(4), audio(4), pci(4)

HISTORY

The **auvia** device driver appeared in NetBSD 1.5.

NAME

auxreg — Sun SPARC auxiliary register

SYNOPSIS

```
auxreg0 at mainbus0 # sun4c
auxreg0 at obio0    # sun4m
options BLINK
```

DESCRIPTION

The **auxreg** device contains controls for the front panel LED and various status bits for the SPARC sparc/fdc(4) floppy controller.

options BLINK Force the front panel LED to blink.

SEE ALSO

sparc/fdc(4)

HISTORY

The **auxreg** appeared in NetBSD 1.0.

NAME

awacs — Apple audio device driver

SYNOPSIS

```
awacs* at obio?
audio* at awacs?
```

DESCRIPTION

The **awacs** (audio waveform amplifier and converter for sound) driver provides support for the audio hardware found in many Apple PowerMacs.

SEE ALSO

audio(4)

HISTORY

The **awacs** device driver appeared in NetBSD 1.5.1.

AUTHORS

The **awacs** driver was written by Tsubai Masanari.

BUGS

This driver needs more testing.

This manual page needs more precision and detail.

NAME

awi — AMD PCnetMobile IEEE 802.11 PCMCIA wireless network driver

SYNOPSIS

awi* **at pcmcia? function ?**

DESCRIPTION

The **awi** driver supports various IEEE 802.11 wireless cards that run AMD PCnetMobile firmware based on the AMD 79c930 controller with the Intersil (formerly Harris) PRISM radio chipset. It provides access to 32kb of memory shared between the controller and the host. All host/device interaction is accomplished via this shared memory, which can be accessed either via PCMCIA or I/O memory spaces. The **awi** driver encapsulates all IP and ARP traffic in 802.11 frames.

The driver works both in infrastructure mode and in ad-hoc (independent BSS) mode.

In infrastructure mode, it communicates with an Access Point, which serves as a link-layer bridge between an Ethernet segment and the wireless network. An access point also provides roaming capability, which allows a wireless node to move between access points.

In ad-hoc mode, the device communicates peer to peer. Although it is more efficient to communicate between wireless nodes, the coverage is limited spatially due to the lack of roaming capability.

In addition to these two modes in the IEEE 802.11 specification, the **awi** driver also supports a variant of ad-hoc mode outside of the spec for DS radio cards. This makes it possible to communicate with the WaveLAN ad-hoc mode of `wi`(4) driver. The NWID has no effect in this mode.

Another mode added to the **awi** driver can be used with old Melco access points with 2Mbps cards. This mode actually uses the IEEE 802.11 ad-hoc mode with encapsulation of raw Ethernet packets (including headers) in 802.11 frames.

For more information on configuring this device, see `ifconfig`(8) and `ifmedia`(4).

HARDWARE

Cards supported by the **awi** driver include:

BayStack 650	1Mbps Frequency Hopping PCCARD adapter
BayStack 660	2Mbps Direct Sequence PCCARD adapter
Icom SL-200	2Mbps Direct Sequence PCCARD adapter
Melco WLI-PCM	2Mbps Direct Sequence PCCARD adapter
NEL SSMagic	2Mbps Direct Sequence PCCARD adapter
Netwave AirSurfer Plus	1Mbps Frequency Hopping PCCARD adapter
Netwave AirSurfer Pro	2Mbps Direct Sequence PCCARD adapter
Nokia C020 WLAN	2Mbps Direct Sequence PCCARD adapter
Farallon SkyLINE	2Mbps Direct Sequence PCCARD adapter
Zoom Air Model 4000	

The original Xircom Netwave AirSurfer is supported by the `cnw`(4) driver, and the PRISM-II cards are supported by the `wi`(4) driver.

MEDIA SELECTION

In addition to default *Auto* media type, the DS cards support *DS1* and *DS2* media types, while the FH cards support the *FH1* media type. For each media type, the *adhoc* mediaopt can be used to indicate to the driver to operate in ad-hoc mode. The *flag0* mediaopt should be used only with old access points, which operate in

IBSS mode. For DS radio cards, the *adhoc,flag0* mediaopt can be used for `wi`(4) compatible WaveLAN ad-hoc mode.

DIAGNOSTICS

awi0: no suitable CIS info found The device cannot be mapped due to a resource conflict. Or, the device failed to initialize its firmware.

awi0: failed to complete selftest (%s) The device failed to complete its self test. In some circumstances, resetting device after power on fails. Re-inserting the card or setting the interface up and then down again (using `ifconfig`(8)) may also be helpful.

awi0: transmit timeout The device failed to generate an interrupt to acknowledge a transmitted packet.

awi0: failed to lock interrupt The system was unable to obtain the lock to access shared memory.

awi0: command %d failed %x The device failed to complete the request from the system.

SEE ALSO

`arp`(4), `cnw`(4), `ifmedia`(4), `netintro`(4), `pcmcia`(4), `wi`(4), `ifconfig`(8), `wiconfig`(8)

Am79C930 PCnet Mobile Single-Chip Wireless LAN Media Access Controller, http://www.amd.com.

HISTORY

The **`awi`** device driver first appeared in NetBSD 1.5.

AUTHORS

The initial version of the **`awi`** driver was written by Bill Sommerfeld ⟨sommerfeld@NetBSD.org⟩. It was then completely rewritten to support cards with the DS phy and ad-hoc mode by
Atsushi Onoe ⟨onoe@NetBSD.org⟩.

NAME

axe — ASIX AX88172 USB Ethernet driver

SYNOPSIS

```
axe*   at uhub?
XXphy* at mii?
```

HARDWARE

The **axe** driver supports the following adapters:

Corega FEther USB2-TX
D-Link DUB-E100
Linksys USB 200M
Melco LUA-U2-KTX
Netgear FA120
Sitecom LN029
Systemtalks SGCX2UL

DESCRIPTION

The **axe** driver provides support for USB (2.0) Ethernet adapters based on the ASIX AX88172 chip.

The chip contains a 10/100 Ethernet MAC with MII interface and is designed to work with both Ethernet and HomePNA transceivers. The chip also supports USB 2.0, thereby accommodating 100 Mb/s data rates.

The **axe** driver supports the following media types:

autoselect	Enable automatic selection of the media type and options.
10baseT/UTP	Set 10Mbps operation. The *mediaopt* option can also be used to enable *full-duplex* operation. Not specifying *full duplex* implies *half-duplex* mode.
100baseTX	Set 100Mbps (fast Ethernet) operation. The *mediaopt* option can also be used to enable *full-duplex* operation. Not specifying *full duplex* implies *half-duplex* mode.

The **axe** driver supports the following media options:

full-duplex	Force full duplex operation. The interface will operate in half duplex mode if this media option is not specified.

For more information on configuring this device, see `ifconfig`(8).

SEE ALSO

`arp`(4), `netintro`(4), `usb`(4), `ifconfig`(8)

ASIX AX88172 data sheet, http://www.asix.com.tw.

HISTORY

The **axe** device driver first appeared in FreeBSD and was ported to NetBSD 3.0. It replaces the NetBSD uax driver.

NAME

az — Aztech/PackardBell radio card device driver

SYNOPSIS

```
az0    at isa? port 0x350
az1    at isa? port 0x358
radio* at az?
```

DESCRIPTION

The **az** driver provides support for the Aztech/PackardBell radio cards.

The Aztech/PackardBell cards are stereo FM tuners that allow tuning in the 87.5-108.0 MHz range. They are capable of reporting signal status (tuned/not tuned, stereo/mono signal) and forcing audio output to mono.

The Aztech cards use only one I/O port. The I/O port is set by the driver to the value specified in the configuration file. The I/O port must be one of 0x350 and 0x358.

SEE ALSO

isa(4), radio(4)

HISTORY

The **az** device driver appeared in OpenBSD 3.0 and NetBSD 1.6.

AUTHORS

The **az** driver was written by Vladimir Popov and Maxim Tsyplakov. The man page was written by Vladimir Popov.

BUGS

It is impossible to determine to which frequency the card is tuned. Thus, the driver will report an internally stored value even if it is not correct (changed by some program that uses direct port access).

NAME

azalia — Generic High Definition Audio device driver

SYNOPSIS

```
azalia* at pci? dev ? function ?
audio* at audiobus?

options AZALIA_DEBUG
```

DESCRIPTION

The **azalia** device driver is expected to support any PCI device which is compliant to the High Definition Audio Specification 1.0. Known supported controllers are Intel 82801F (ICH6) and Intel 82801G (ICH7). Known supported codecs are Realtek ALC260, ALC880, ALC882, Analog Devices AD1981HD, CMedia CMI9880, Sigmatel STAC9221, and STAC9221D.

SEE ALSO

audio(4), pci(4) http://www.intel.com/standards/hdaudio/

HISTORY

The **azalia** device driver appeared in NetBSD 3.0.

BUGS

Non-PCM encodings such as Float32 and AC-3 are not supported.

There is no way to use 20bit or 24bit precision because of a limitation of the MI audio framework.

NAME

bah — ARCnet network driver for SMC COM90C26 based boards

SYNOPSIS

bah* **at zbus0**

DESCRIPTION

The **bah** interface provides access to the 2.5 Mb/s ARCnet network via the SMC COM90C26 + COM90C32 ARCnet chip set.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The interface MTU is 507 for protocols that do not use link level fragmentation and 60480 bytes for the others. The routing layer may specify additional limits.
Currently supported protocols are IPv4(+ARP), and IPv6.

IP VERSION 4 CONSIDERATIONS

When the NOARP flag is set on the **bah** interface, it does not employ the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network. Instead, it uses the least significant 8 bits of the IP address as hardware address, as described in RFC 1051 and RFC 1201.

With the IFF_LINK0 flag cleared, IP and ARP encoding is done according to the deprecated, but popular among Amiga users, RFC 1051 encoding (that is, with simple header, packet type 240 / 241), and the MTU is 507.

With the IFF_LINK0 flag set, IP/ARP/RARP encoding is done according to RFC 1201 (that is, with Packet Header Definition Standard header and packet type 212/213). The MTU is normally 1500.

When switching between the two modes, do a **ifconfig interfacename down up** to switch the MTU.

When the IFF_LINK2 flag is set, ARP packets are sent with the protocol type encoded as it would be in the ARCnet header, and decoded to the right protocol encoding on reception. According to "assigned numbers", this is wrong, but some legacy software (namely, AmiTCP 3.0beta) shows this bug.

HARDWARE

The **bah** interface supports the following Zorro II expansion cards:

A2060	Commodore's ARCnet card, manufacturer 514, product 9
AMERISTAR	Ameristar's ARCnet card, manufacturer 1053, product 9

SEE ALSO

`arp`(4), `inet`(4), `intro`(4), `ifconfig`(8)

P.A. Prindeville, "Standard for the transmission of IP datagrams and ARP packets over ARCNET networks.", *RFC*, 1051, March 1988.

D. Provan, "Transmitting IP traffic over ARCNET networks.", *RFC*, 1201, February 1991.

I. Souvatzis, "Transmission of IPv6 Packets over ARCnet Networks.", *RFC*, 2497, January 1999.

ARCnet Packet Header Definition Standard, Novell Inc., 1989

STANDARDS

RFC 1051/RFC1201 with ARP, or without, using direct mapping of lower 8 IP address bits instead.

HISTORY

The Amiga **bah** interface first appeared in NetBSD 1.1. ARP support was added in NetBSD 1.3.

AUTHORS

Ignatios Souvatzis

NAME

battery — support for batteries and related functionality in Apple Powerbook 2400, 3400 and 3500 (also known as Original PowerBook G3)

SYNOPSIS

```
battery* at pmu?
```

DESCRIPTION

The **battery** driver provides support for batteries and hardware sensors found in 1st generation PCI-based PowerBooks, currently it only exports a few sensors via the envsys(4) interface, including battery charge, voltage, CPU and battery temperature and AC power availability.

SEE ALSO

envsys(4), pmu(4), envstat(8)

BUGS

There is no APM emulation right now and the battery current sensor may return misleading or wrong data. This driver is considered preliminary and may be replaced at some point.

NAME

bba — IOASIC Baseboard Audio device driver

SYNOPSIS

```
bba0   at ioasic? offset ?
audio* at audiobus?
```

DESCRIPTION

The **bba** driver provides support for the IOASIC baseboard audio found on DEC 3000/300, 3000/500 (NetBSD/alpha) and DEC Personal DECstation (NetBSD/pmax) systems. The baseboard audio driver is based on the AMD 79c30 ISDN and audio interface. The interface is only capable of playing and recording 8kHz mu-law audio.

SEE ALSO

audio(4), ioasic(4)

HISTORY

The **bba** device driver appeared in NetBSD 1.5. The name for the driver was adopted from the same driver in ULTRIX.

NAME

bce — Broadcom BCM4401 Ethernet device driver

SYNOPSIS

```
bce* at pci? dev ? function ?
```

DESCRIPTION

The **bce** provides support for the Broadcom BCM4401 10/100 Ethernet card. Other cards from the 440x series may also be supported.

SEE ALSO

bge(4), mii(4), ukphy(4)

HISTORY

The **bce** driver appeared in NetBSD 1.6.2.

AUTHORS

Cliff Wright ⟨cliff@snipe444.org⟩

BUGS

There is no VLAN support.

There is no flow control support.

Multicast is not using the packet filter and is in the accept all multicast mode.

NAME

bcsp — BlueCore Serial Protocol driver

SYNOPSIS

```
pseudo-device bcsp
```

DESCRIPTION

The **bcsp** driver provides a tty(4) line discipline to send and receive BlueCore Serial Protocol packets over a serial line, as described in the "BlueCore Serial Protocol (BCSP)" specification.

Moreover, the **bcsp** supports BCSP Link Establishment Protocol, as described in the "BCSP Link Establishment Protocol" specification.

SEE ALSO

bluetooth(4), btuart(4), btattach(8)

HISTORY

The **bcsp** device appeared in NetBSD 5.0.

BUGS

The **bcsp** not support configuration for baud rate yet.

AUTHORS

KIYOHARA Takashi ⟨kiyohara@kk.iij4u.or.jp⟩

NAME

be — SPARC Fast Ethernet interface

SYNOPSIS

```
qec* at sbus? slot ? offset ?
be* at qec?
```

DESCRIPTION

The **be** interface provides access to the 10Mb/s and 100Mb/s (half duplex only) Ethernet networks. The **be** is found on the Sun 10/100 Mbit Ethernet boards (Sun part number SUNW,501-2450).

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **be** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

The **be** is not capable of link autonegotiation, so a media type must be specified with `ifconfig`(8). The supported media types are:

media 100baseTX
: Use 100Mbps, half duplex

media 10baseT
: Use 10Mbps, half duplex

SEE ALSO

`arp`(4), `ifmedia`(4), `inet`(4), `intro`(4), `netintro`(4), `sbus`(4), `ifconfig`(8)

HISTORY

Support for the **be** first appeared in NetBSD 1.4.

NAME

bge — Broadcom BCM570x family Gigabit Ethernet driver

SYNOPSIS

bge* **at pci? dev ? function ?**

Configuration of PHYs may also be necessary. See `mii`(4).

DESCRIPTION

The **bge** device driver supports Gigabit Ethernet interfaces based on the Broadcom BCM570x family of Gigabit Ethernet chips. The interfaces supported by the **bge** driver include:

- 3Com 3c996-T (10BASE-T/100BASE-TX/1000BASE-T)
- Dell PowerEdge 2550 integrated BCM5700 NIC (10BASE-T/100BASE-TX/1000BASE-T)
- IBM x235 server integrated BCM5703x NIC (10BASE-T/100BASE-TX/1000BASE-T)
- Netgear GA302T (10BASE-T/100BASE-TX/1000BASE-T)
- SysKonnect SK-9D21 (10BASE-T/100BASE-TX/1000BASE-T)
- SysKonnect SK-9D41 (1000BASE-SX)

The BCM570x family supports IPv4/TCP/UDP checksumming in hardware. The **bge** driver supports this feature of the chip. See `ifconfig`(8) for information on how to enable this feature.

SEE ALSO

`arp`(4), `bce`(4), `ifmedia`(4), `mii`(4), `netintro`(4), `pci`(4), `ifconfig`(8)

HISTORY

The **bge** driver first appeared in NetBSD 1.6.1.

AUTHORS

The **bge** driver was written by Bill Paul ⟨wpaul@windriver.com⟩ for FreeBSD and ported to NetBSD by Frank van der Linden ⟨fvdl@wasabisystems.com⟩, Jason R. Thorpe ⟨thorpej@wasabisystems.com⟩ and Jonathan Stone ⟨jonathan@dsg.stanford.edu⟩.

NAME

bha, **bt** — Buslogic SCSI adapter driver

SYNOPSIS

```
bha0 at isa? port 0x330 irq ? drq ?
bha* at eisa? slot ?
bha* at pci? dev ? function ?
scsibus* at bha?
```

DESCRIPTION

The **bha** driver supports the following Buslogic SCSI adapters:

Buslogic ISA BT-445
Buslogic EISA BT-74x
Buslogic PCI BT-9[45][68]

SEE ALSO

cd(4), ch(4), intro(4), scsi(4), sd(4), st(4)

HISTORY

In NetBSD 1.2 and earlier, this driver was named **bt** but was renamed to **bha** in later releases.

BUGS

The Buslogic BT-930 is not supported in this driver.

The Buslogic BT-445S has a problem in early hardware and firmware revisions which prevents proper operation on a system with more than 16MB of RAM. Hardware revision D and firmware revision 3.37 should be considered minimum requirements for using this board on systems configured in this manner.

NAME

bio — Block IO ioctl tunnel pseudo-device

SYNOPSIS

```
pseudo-device bio
```

DESCRIPTION

The **bio** driver provides userland applications `ioctl`(2) access to devices otherwise not found as `/dev` nodes. The `/dev/bio` device node operates by delegating ioctl calls to a requested device driver. Only drivers which have registered with the **bio** device can be accessed via this interface.

The following device drivers register with **bio** for volume management:

`arcmsr`(4)	Areca Technology Corporation SATA RAID controller
`cac`(4)	Compaq RAID array controller
`ciss`(4)	Compaq Smart ARRAY 5/6 SAS/SATA/SCSI RAID controller
`mfi`(4)	LSI Logic & Dell MegaRAID SAS RAID controller

The following ioctl calls apply to the **bio** device:

`BIOCLOCATE` Locate a named device and give back a cookie to the application for subsequent ioctl calls. The cookie is used to tunnel further ioctls to the right device.

`BIOCINQ` Retrieve number of volumes and physical disks for a specific device.

`BIOCDISK` Retrieve detailed information for the specified physical disk. Information returned can include status, size, channel, target, lun, vendor name, serial number, and processor device (ses or safte).

`BIOCDISK_NOVOL` Is just the same as *BIOCDISK* but doesn't require the disks to be in volume sets, so this applies to any physical disk connected to the controller.

Note: this ioctl might not be supported on all hardware.

`BIOCVOL` Retrieve detailed information for the specified volume. Information returned can include status, size, RAID level, number of disks, device name association (sd?) and vendor name.

`BIOCALARM` Control the alarm beeper on the device. Supported states are: disable alarm, enable alarm, silence alarm, status and test alarm.

Note: These options might not be supported on all hardware.

`BIOCBLINK` Blink an LED of the specified physical disk. Supported blink states are: blink LED, unblink LED and blink alarm LED.

Note: This option is only supported if the disk is governed by ses(4) or safte(4) and the hardware supports hardware blinking.

`BIOCSETSTATE` Alter the state of specified physical disk. Supported states are: create/remove hotspare, create/remove pass through disk, start/stop consistency check in a volume, online disk and offline disk.

Note: These options might not be supported on all hardware.

`BIOCVOLOPS` For operations in volume sets. It's able to create and remove a volume set in a supported RAID controller.

Note: this ioctl might not be supported on all hardware.

FILES

`/dev/bio` ioctl tunnel device

SEE ALSO

`ioctl`(2), `bioctl`(8)

HISTORY

The **`bio`** driver first appeared in OpenBSD 3.2 and NetBSD 4.0.

AUTHORS

The **`bio`** driver was written by Niklas Hallqvist ⟨niklas@openbsd.org⟩. The API was written by Marco Peereboom ⟨marco@openbsd.org⟩ and was extended even more for NetBSD by Juan Romero Pardines ⟨xtraeme@netbsd.org⟩.

NAME

bktr — Brooktree 848 compatible TV card driver

SYNOPSIS

```
bktr* at pci? dev ? function ?
radio* at bktr?

#include <dev/ic/bt8xx.h>

options BKTR_OVERRIDE_CARD=n
options BKTR_OVERRIDE_TUNER=n
options BKTR_OVERRIDE_DBX=n
options BKTR_OVERRIDE_MSP=n
options BKTR_SYSTEM_DEFAULT=n
options BKTR_USE_PLL
options BKTR_GPIO_ACCESS
options BKTR_NO_MSP_RESET
```

DESCRIPTION

This driver supports video capture (frame grabber) and TV tuner cards based on the Brooktree Bt848, Bt848A, Bt849A, Bt878, and Bt879 chips.

Supported cards include most cards by AVerMedia, Hauppauge, Leadtek, Miro, Pinnacle, Pixelview, Terratec, and some other companies, especially all cards based on the Brooktree Bt848, Bt848A, Bt849A, Bt878, or Bt879 chips. A notable exception are the ATI All-in-Wonder cards.

The following kernel configuration options are available:

options BKTR_OVERRIDE_CARD=n

If the card is not recognized correctly by the auto-detection routine, it can be overridden by setting this option to the appropriate value. The following values are allowed:

1 Pinnacle Systems (Miro) TV,
2 Hauppauge WinCast/TV,
3 STB TV/PCI,
4 Intel Smart Video III and Videologic Captivator PCI,
5 IMS TV Turbo,
6 AVerMedia TV/FM,
7 MMAC Osprey,
8 NEC PK-UG-X017,
9 I/O DATA GV-BCTV2/PCI,
10 Animation Technologies FlyVideo,
11 Zoltrix TV,
12 KISS TV/FM PCI,
13 Video Highway Xtreme,
14 Askey/Dynalink Magic TView,
15 Leadtek WinFast TV 2000/VC100,
16 TerraTec TerraTV+, and
17 TerraTec TValue.

options BKTR_OVERRIDE_TUNER=n

If the TV tuner is not recognized correctly by the auto-detection routine, it can be overridden by setting this option to the appropriate value. Known values are:

1 Temic NTSC,
2 Temic PAL,
3 Temic SECAM,
4 Philips NTSC,
5 Philips PAL,
6 Philips SECAM,
7 Temic PAL I,
8 Philips PAL I,
9 Philips FR1236 NTSC FM,
10 Philips FR1216 PAL FM,
11 Philips FR1236 SECAM FM,
12 ALPS TSCH5 NTSC FM, and
13 ALPS TSBH1 NTSC.

options BKTR_OVERRIDE_DBX=n
To override detection of the BTSC (dbx) chip, set this to *1* if you have one, or *0* if not.

options BKTR_OVERRIDE_MSP=n
To override detection of the MSP 34xx chip, set this to *1* if you have one, or *0* if not.

options BKTR_SYSTEM_DEFAULT=n
If this option is set to *BROOKTREE_PAL* default to PAL, else to NTSC.

options BKTR_USE_PLL
Default to PLL instead of XTAL.

options BKTR_GPIO_ACCESS
Use **ioctl**()s for direct GPIO access.

options BKTR_NO_MSP_RESET
Skip the MSP reset. This option is handy if you initialize the MSP audio in another operating system first and then do a soft reboot.

VIDEO CAPTURE INTERFACE

The video capture interface to **bktr** is accessed through the `/dev/bktrN` devices. The following `ioctl`(2) commands are supported on the Brooktree848 video capture interface:

`METEORSFMT` *`unsigned long *`*
This command sets the video format, also sometimes referred to as the video norm. The supported formats are:

`METEOR_FMT_NTSC`	NTSC
`METEOR_FMT_PAL`	PAL
`METEOR_FMT_SECAM`	SECAM
`METEOR_FMT_AUTOMODE`	hardware default

`METEORGFMT` *`unsigned long *`*
This command retrieves the current video format to the *`unsigned long *`* argument.

`METEORSETGEO` *`struct meteor_geomet *`*
This command sets the video properties that affect the bit size of a frame throught the *`meteor_geomet *`* argument.

```
struct meteor_geomet {
        u_short         rows;    /* height in pixels*/
        u_short         columns; /* width in pixels */
        u_short         frames;
```

```
        u_long          oformat;
}
```

The *frames* field is the number of frames to buffer. Currently only 1 frame is supported for most operations.

The *oformat* field is a bit-field describing the output pixel format type and which video fields to capture. The following are supported pixel format types:
.Pp

METEOR_GEO_RGB16	16-bit RGB
METEOR_GEO_RGB24	24-bit RGB in 32 bits
METEOR_GEO_YUV_PACKED	16-bit 4:2:2 YUV
METEOR_GEO_YUV_PLANAR	16-bit 4:2:2 YUV
METEOR_GEO_YUV_UNSIGNED	unsigned UV
METEOR_GEO_YUV_422	
METEOR_GEO_YUV_12	
METEOR_GEO_YUV_9	

The following are supported field capture modes:

METEOR_GEO_ODD_ONLY	only odd fields
METEOR_GEO_EVEN_ONLY	only even fields

By default, frames will consist of both the odd and even fields.

METEORGSUPPIXFMT *struct meteor_pixfmt ∗*
: This command is used interatively to fetch descriptions of supported output pixel formats into the *meteor_pixfmt ∗* argument.

```
struct meteor_pixfmt {
        u_int           index;
        METEOR_PIXTYPE type;
        u_int           Bpp;            /∗ bytes per pixel ∗/
        u_long          masks[3];       /∗ YUV bit masks ∗/
        unsigned        swap_bytes :1;
        unsigned        swap_shorts:1;
};
```

To query all the supported formats, start with an index field of 0 and continue with successive encodings (1, 2, ...) until the command returns an error.

METEORSACTPIXFMT *int ∗*
: This command sets the active pixel format. The *int ∗* argument is the index of the pixel format as returned by METEORGSUPPIXFMT.

METEORGACTPIXFMT *int ∗*
: This command fetches the active pixel format index into the *int ∗* argument.

METEORSINPUT *unsigned long ∗*
: This command sets the input port of the Brooktree848 device. The following are supported input ports:

METEOR_INPUT_DEV0	composite (RCA)
METEOR_INPUT_DEV1	tuner
METEOR_INPUT_DEV2	composite S-video

METEOR_INPUT_DEV3	mystery device
METEOR_INPUT_DEV_RGB	rgb meteor
METEOR_INPUT_DEV_SVIDEO	S-Video

Not all devices built with Brooktree848 chips support the full list of input ports.

METEORGINPUT *unsigned long* *
: This command retrieves the current input port to the *unsigned long* * argument.

METEORSFPS *unsigned short* *
: This command sets the number of frames to grab each second. Valid frame rates are integers from 0 to 30.

METEORGFPS *unsigned short* *
: This command fetches the number of frames to grab each second into the *unsigned short* * argument.

METEORCAPTUR *int* *
: This command controls capturing of video data. The following are valid arguments:

METEOR_CAP_SINGLE	capture one frame
METEOR_CAP_CONTINOUS	continuously capture
METEOR_CAP_STOP_CONT	stop continuous capture

METEORSSIGNAL *unsigned int* *
: This command controls the signal emission properties of **bktr**. If the *unsigned int* * argument is a valid signal, then that signal will be emitted when either a frame or field capture has completed. To select between frame or field signalling, the following arguments are used:

METEOR_SIG_FRAME	signal every frame
METEOR_SIG_FIELD	signal every field

By default, signals will be generated for every frame. Generation of signals is terminated with the METEOR_SIG_MODE_MASK argument.

TUNER INTERFACE

Most cards supported by this driver feature a hardware television tuner on the I2C bus. The tuner interface to **bktr** is accessed through the /dev/tunerN devices. The following ioctl(2) commands are supported on the tuner interface:

TVTUNER_SETTYPE *unsigned int* *
: This command sets the tuner's TV channel set, also sometimes called the TV channel band. This setting is used to calculate the proper tuning frequencies. The desired channel set must be selected before attempting to set the tuner channel or frequency. The following is a list of valid channel sets:

CHNLSET_NABCST	North America broadcast
CHNLSET_CABLEIRC	North America IRC cable
CHNLSET_CABLEHRC	North America HRC cable
CHNLSET_WEUROPE	Western Europe
CHNLSET_JPNBCST	Japan broadcast
CHNLSET_JPNCABLE	Japan cable
CHNLSET_XUSSR	Russia
CHNLSET_AUSTRALIA	Australia
CHNLSET_FRANCE	France

TVTUNER_GETTYPE *unsigned int* *
: This command fetches the tuner's current channel set to the *unsigned int* * argument.

`TVTUNER_SETCHNL` *`unsigned int *`*
This command sets the tuner's frequency to a specified channel in the current channel set.

`TVTUNER_GETCHNL` *`unsigned int *`*
This command fetches the last selected channel. Note that it is not necessarily the current channel. In particular, changing the tuner's frequency by a command other than `TVTUNER_SETCHNL` will not update this setting, and it defaults to 0 on driver initialization.

`TVTUNER_SETFREQ` *`unsigned int *`*
This command sets the tuner's frequency to 1/16th the value of the *`unsigned int *`* argument, in MHz. Note that the current channelset is used to determine frequency offsets when this command is executed.

`TVTUNER_GETFREQ` *`unsigned int *`*
This command fetches the tuner's current frequency to the *`unsigned int *`* argument. Note that this value is 16 times the actual tuner frequency, in MHz.

`BT848_SAUDIO` *`int *`*
This command controls the audio input port and mute state. The following is a list of valid arguments:

`AUDIO_TUNER`	tuner audio port
`AUDIO_EXTERN`	external audio port
`AUDIO_INTERN`	internal audio port
`AUDIO_MUTE`	mute audio
`AUDIO_UNMUTE`	unmute audio

`BT848_GAUDIO` *`int *`*
This command fetches the audio input and mute state bits to the *`int *`* argument.

FILES

`/dev/bktr*` **`bktr`** driver interface device
`/dev/tuner*` **`bktr`** tuner interface device
`/dev/vbi*` teletext interface device

SEE ALSO

`options`(4), `pci`(4), `radio`(4), `pkgsrc/audio/xmradio`, `pkgsrc/multimedia/ffmpeg`, `pkgsrc/multimedia/fxtv`

HISTORY

The **`bktr`** driver appeared in FreeBSD 2.2 and NetBSD 1.5.

AUTHORS

The **`bktr`** driver was originally written by Amancio Hasty for FreeBSD and is now maintained by Roger Hardiman. NetBSD porting was done by Bernd Ernesti, Berndt Josef Wulf, Matthias Scheler, and Thomas Klausner.

NAME

bluetooth — Bluetooth Protocol Family

SYNOPSIS

```
#include <netbt/bluetooth.h>
#include <netbt/hci.h>
#include <netbt/l2cap.h>
#include <netbt/rfcomm.h>
```

DESCRIPTION

The Bluetooth Protocol Family

ADDRESSING

Bluetooth Protocol Family sockets all use a *sockaddr_bt* structure which contains a Bluetooth Device Address (BDADDR). This consists of a six byte string in least significant byte first order.

```
struct sockaddr_bt {
        uint8_t         bt_len;
        sa_family_t     bt_family;
        bdaddr_t        bt_bdaddr;
        uint16_t        bt_psm;
        uint8_t         bt_channel;
};
```

The local address used by the socket can be set with bind(2).

PROTOCOLS

Protocols included are:

BTPROTO_HCI

This gives raw access to the Host Controller Interface of local devices using the HCI protocol as described in the Bluetooth Core Specification. Any user may open an HCI socket but there are limitations on what unprivileged users can send and receive. The local address specified by bind(2) may be used to select the device that the socket will receive packets from. If BDADDR_ANY is specified then the socket will receive packets from all devices on the system. connect(2) may be used to create connections such that packets sent with send(2) will be delivered to the specified device, otherwise sendto(2) should be used.

The *bt_psm* and *bt_channel* fields in the sockaddr_bt structure are ignored by HCI protocol code and should be set to zero.

HCI socket options:

SO_HCI_EVT_FILTER [*struct hci_filter*]
: This filter controls which events will be recieved at the socket. See <netbt/hci.h> for available events. By default, Command_Complete and Command_Status events only are enabled.

SO_HCI_PKT_FILTER [*struct hci_filter*]
: This filter controls the type of packets that will be received at the socket. By default, Event packets only are enabled.

SO_HCI_DIRECTION [*int*]
: When set, this enables control messages on packets received at the socket indicating the direction of travel of the packet.

HCI `sysctl`(8) controls:

`net.bluetooth.hci.sendspace`
Default send buffer size for HCI sockets.

`net.bluetooth.hci.recvspace`
Default receive buffer size for HCI sockets

`net.bluetooth.hci.acl_expiry`
If set, this is the time in seconds after which unused ACL data connections will be expired. If zero, connections will not be closed.

`net.bluetooth.hci.memo_expiry`
Time, in seconds, that the system will keep records of Bluetooth devices in the vicinity after an Inquiry Response packet has been recieved. This information is used for routing purposes.

`net.bluetooth.hci.eventq_max`
The maximum number of packets on the low level Event queue.

`net.bluetooth.hci.aclrxq_max`
The maximum number of packets on the low level ACL queue.

`net.bluetooth.hci.scorxq_max`
The maximum number of packets on the low level SCO queue.

`BTPROTO_L2CAP`
L2CAP sockets give sequential packet access over channels to other Bluetooth devices and make use of the *`bt_psm`* field in the *`sockaddr_bt`* structure to select the Protocol/Sevice Multiplexer to specify when making connections.

L2CAP socket options:

`SO_L2CAP_IMTU` [*`uint16_t`*]
Incoming MTU

`SO_L2CAP_OMTU` [*`uint16_t`*]
Outgoing MTU (read-only)

`SO_L2CAP_LM` [*`int`*]
Link Mode. The following bits may be set:

`L2CAP_LM_AUTH`	Request authentication (pairing).
`L2CAP_LM_ENCRYPT`	Request encryption (includes auth).
`L2CAP_LM_SECURE`	Request secured link (encryption, plus change link key).

Link mode settings will be applied to the baseband link during L2CAP connection establishment. If the L2CAP connection is already established, `EINPROGRESS` may be returned, and it is not possible to guarantee that data already queued (from either end) will not be delivered. If the mode change fails, the L2CAP connection will be aborted.

L2CAP `sysctl`(8) controls:

`net.bluetooth.l2cap.sendspace`
Default send buffer size for L2CAP sockets.

`net.bluetooth.l2cap.recvspace`
Default receive buffer size for L2CAP sockets.

`net.bluetooth.l2cap.rtx`
Response Timeout eXpiry for L2CAP signals.

`net.bluetooth.l2cap.ertx`
Extended Response Timeout eXpiry for L2CAP signals.

BTPROTO_RFCOMM

RFCOMM sockets provide streamed data over Bluetooth connection and make use of the *bt_psm*, and *bt_channel* fields in the *sockaddr_bt* structure. The channel number must be between 1 and 30 inclusive except that if the special value `RFCOMM_CHANNEL_ANY` is bound, when the `listen`(2) call is made, the first unused channel for the relevant bdaddr will be allocated and may be discovered using the `getsockname`(2) call. If no PSM is specified, a default value of `L2CAP_PSM_RFCOMM` (0x0003) will be used.

RFCOMM socket options:

`SO_RFCOMM_MTU` [*uint16_t*]
Maximum Frame Size to use for this link.

`SO_RFCOMM_LM` [*int*]
Link Mode. The following bits may be set at any time:

`RFCOMM_LM_AUTH`	Request authentication (pairing).
`RFCOMM_LM_ENCRYPT`	Request encryption (includes auth).
`RFCOMM_LM_SECURE`	Request secured link (encryption, plus change link key).

Link mode settings will be applied to the baseband link during RFCOMM connection establishment. If the RFCOMM connection is already established, `EINPROGRESS` may be returned, and it is not possible to guarantee that data already queued (from either end) will not be delivered. If the mode change fails, the RFCOMM connection will be aborted.

RFCOMM `sysctl`(8) controls:

`net.bluetooth.rfcomm.sendspace`
Default send buffer size for RFCOMM sockets.

`net.bluetooth.rfcomm.recvspace`
Default receive buffer size for RFCOMM sockets.

`net.bluetooth.rfcomm.default_mtu`
Maximum Frame Size (N1)

`net.bluetooth.ack_timeout`
Acknowledgement Timer (T1)

`net.bluetooth.mcc_timeout`
Response Timer for Multiplexer Control Channel (T2)

BTPROTO_SCO

SCO sockets provide sequential packet access to time sensitive data channels over Bluetooth connections, typically used for audio data.

SCO socket options:

`SO_SCO_MTU` [*uint16_t*]
Maximum packet size for use on this link. This is read-only and will be set by the protocol code when a connection is made. Currently, due to limitations in the `ubt`(4) driver, the SCO protocol code will only accept packets with exactly this size.

`SO_SCO_HANDLE` [*uint16_t*]
Connection handle for this link. This is read-only and provided for informational purposes only.

SCO `sysctl`(8) controls:

`net.bluetooth.sco.sendspace`
Default send buffer size for SCO sockets.

`net.bluetooth.sco.recvspace`
Default receive buffer size for SCO sockets.

INFORMATION

The following `ioctl`(2) calls may be used to manipulate Bluetooth devices. The `ioctl`(2) must be made on **`BTPROTO_HCI`** sockets. All of the requests take a *`btreq`* structure defined as follows as their parameter and unless otherwise specified, use the *`btr_name`* field to identify the device.

```
struct btreq {
    char btr_name[HCI_DEVNAME_SIZE];  /* device name */

    union {
        struct {
            bdaddr_t btri_bdaddr;     /* device bdaddr */
            uint16_t btri_flags;      /* flags */
            uint16_t btri_num_cmd;    /* # of free cmd buffers */
            uint16_t btri_num_acl;    /* # of free ACL buffers */
            uint16_t btri_num_sco;    /* # of free SCO buffers */
            uint16_t btri_acl_mtu;    /* ACL mtu */
            uint16_t btri_sco_mtu;    /* SCO mtu */
            uint16_t btri_link_policy;        /* Link Policy */
            uint16_t btri_packet_type;        /* Packet Type */
        } btri;
        struct bt_stats btrs;   /* unit stats */
    } btru;
};

#define btr_flags     btru.btri.btri_flags
#define btr_bdaddr    btru.btri.btri_bdaddr
#define btr_num_cmd   btru.btri.btri_num_cmd
#define btr_num_acl   btru.btri.btri_num_acl
#define btr_num_sco   btru.btri.btri_num_sco
#define btr_acl_mtu   btru.btri.btri_acl_mtu
#define btr_sco_mtu   btru.btri.btri_sco_mtu
#define btr_link_policy btru.btri.btri_link_policy
#define btr_packet_type btru.btri.btri_packet_type
#define btr_stats     btru.btrs

/* btr_flags */
#define BTF_UP                (1<<0)  /* unit is up */
#define BTF_RUNNING           (1<<1)  /* unit is running */
#define BTF_XMIT_CMD          (1<<2)  /* transmitting CMD packets */
#define BTF_XMIT_ACL          (1<<3)  /* transmitting ACL packets */
#define BTF_XMIT_SCO          (1<<4)  /* transmitting SCO packets */
#define BTF_INIT_BDADDR               (1<<5)  /* waiting for bdaddr */
#define BTF_INIT_BUFFER_SIZE  (1<<6)  /* waiting for buffer size */
#define BTF_INIT_FEATURES     (1<<7)  /* waiting for features */
#define BTF_NOOP_ON_RESET     (1<<8)  /* wait for No-op on reset */
```

```
#define BTF_INIT_COMMANDS     (1<<9)  /* waiting for supported commands */

struct bt_stats {
        uint32_t        err_tx;
        uint32_t        err_rx;
        uint32_t        cmd_tx;
        uint32_t        evt_rx;
        uint32_t        acl_tx;
        uint32_t        acl_rx;
        uint32_t        sco_tx;
        uint32_t        sco_rx;
        uint32_t        byte_tx;
        uint32_t        byte_rx;
};
```

`SIOCGBTINFO` Get Bluetooth device Info. Given the device name, fill in the btreq structure including the address field for use with socket addressing as above.

`SIOCGBTINFOA` Get Bluetooth device Info from Address. Given the device address, fill in the btreq structure including the name field.

`SIOCNBTINFO` Next Bluetooth device Info . If name field is empty, the first device will be returned. Otherwise, the next device will be returned. Thus, you can cycle through all devices in the system.

`SIOCSBTFLAGS` Set Bluetooth device Flags. Not all flags are settable.

`SIOCSBTPOLICY` Set Bluetooth device Link Policy. Link Policy bits are defined in <`netbt/hci.h`>, though you can only set bits that the device supports.

`SIOCSBTPTYPE` Set Bluetooth device Packet Types. You can only set packet types that the device supports.

`SIOCGBTSTATS` Read device statistics.

`SIOCZBTSTATS` Read device statistics, and zero them.

Only the super-user may change device configurations.

SEE ALSO

`bind`(2), `getsockname`(2), `bluetooth`(3), `bcsp`(4), `bt3c`(4), `btbc`(4), `btuart`(4), `options`(4), `ubt`(4)

HISTORY

The Bluetooth Protocol Stack was written for NetBSD 4.0 by Iain Hibbert under the sponsorship of Itronix, Inc.

NAME

bm — Apple BMac Ethernet device driver

SYNOPSIS

bm* at obio?

Configuration of PHYs may also be necessary. See mii(4).

DESCRIPTION

The **bm** driver provides support for the BMac Ethernet hardware found mostly in Apple PowerBooks G3s and Power Macintosh G3s.

SEE ALSO

gem(4), mc(4), mii(4), nsphyter(4), tlp(4)

HISTORY

The **bm** device driver appeared in NetBSD 1.4.

AUTHORS

The **bm** driver was written by Tsubai Masanari ⟨tsubai@NetBSD.org⟩. The man page was written by Thomas Klausner ⟨wiz@NetBSD.org⟩.

NAME

bmd — Nereid bank memory disk driver

SYNOPSIS

```
bmd* at intio0 addr 0xece3f0
bmd* at intio0 addr 0xecebf0
```

DESCRIPTION

The **bmd** driver provides a memory disk for the Nereid bank memory space. There is no special command to configure it, because **bmd** is not a logical disk as md(4).

FILES

`/dev/bmd??` block mode bank memory disk
`/dev/rbmd??` raw mode bank memory disk

EXAMPLES

In order to use it, do as follows:

```
newfs /dev/bmd0c
mount /dev/bmd0c /mnt
```

SEE ALSO

intio(4)

HISTORY

bmd appeared in NetBSD 2.0.

NAME

bmtphy — Driver for Broadcom BCM5201 and BCM5202 “Mini-Theta” Ethernet PHYs and their derivatives

SYNOPSIS

bmtphy* **at mii? phy ?**

DESCRIPTION

The **bmtphy** driver supports the Broadcom BCM5201 and BCM5202 10/100 Ethernet PHYs and their derivatives such as BCM5214, BCM5221, BCM5222, and BCM4401. It also supports the internal PHY on the 3Com 3c905C 10/100 Ethernet interface, and the internal PHY on some 3Com 3c905B 10/100 Ethernet interfaces.

SEE ALSO

ex(4), ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

bnx — Broadcom NetXtreme II 10/100/1000 Ethernet device

SYNOPSIS

```
bnx* at pci?
brgphy* at mii?
```

DESCRIPTION

The **bnx** driver supports Broadcom's NetXtreme II product family, such as the BCM5706 PCI-X and BCM5708 PCI Express Ethernet controllers, which includes the following:

- Dell PowerEdge 1950 integrated BCM5708 NIC (10/100/1000baseT)
- Dell PowerEdge 2950 integrated BCM5708 NIC (10/100/1000baseT)
- HP NC370F PCI-X Multifunction Gigabit server adapter (1000baseSX)
- HP NC370T PCI-X Multifunction Gigabit server adapter (10/100/1000baseT)
- HP NC373F PCI Express Multifunction Gigabit server adapter (1000baseSX)
- HP NC373i PCI Express Multifunction Gigabit embedded server adapter (10/100/1000baseT)
- HP NC380T PCI Express Dual Port Multifunction Gigabit server adapter (10/100/1000baseT)
- IBM xSeries 3550 integrated BCM5708 NIC (10/100/1000baseT)
- IBM xSeries 3650 integrated BCM5708 NIC (10/100/1000baseT)

The NetXtreme II product family is composed of various Converged NIC (or CNIC) Ethernet controllers which support a TCP Offload Engine (TOE), Remote DMA (RDMA), and iSCSI acceleration, in addition to standard L2 Ethernet traffic, all on the same controller. The following features are supported in the **bnx** driver under NetBSD:

```
IPv4 receive IP/TCP/UDP checksum offload
Jumbo frames (up to 9022 bytes)
VLAN tag insertion
Interrupt coalescing
10/100/1000Mbps operation in full-duplex mode
10/100Mbps operation in half-duplex mode
```

The **bnx** driver supports the following media types:

autoselect Enable autoselection of the media type and options. The user can manually override the autoselected mode via `ifconfig`(8).

10baseT/UTP Set 10Mbps operation. The `ifconfig`(8) **mediaopt** option can also be used to select either **full-duplex** or **half-duplex** modes.

100baseTX Set 100Mbps (Fast Ethernet) operation. The `ifconfig`(8) **mediaopt** option can also be used to select either **full-duplex** or **half-duplex** modes.

1000baseTX Set 1000baseTX operation over twisted pair. Only **full-duplex** mode is supported.

The **bnx** driver supports the following media options:

full-duplex Force full duplex operation.

half-duplex Force half duplex operation.

For more information on configuring this device, see `ifconfig`(8).

SEE ALSO

`arp`(4), `brgphy`(4), `ifmedia`(4), `intro`(4), `mii`(4), `netintro`(4), `pci`(4), `ifconfig`(8)

HISTORY

The **bnx** device driver first appeared in NetBSD 4.0.

NAME

boca — multiplexing serial communications interface

SYNOPSIS

For 4-port BB1004 boards:

```
boca0 at isa? port 0x100 irq 5
com2 at boca? slave ?
com3 at boca? slave ?
com4 at boca? slave ?
com5 at boca? slave ?
```

For 8-port BB1008 boards:

```
boca0 at isa? port 0x100 irq 5
com2 at boca? slave ?
com3 at boca? slave ?
com4 at boca? slave ?
com5 at boca? slave ?
com6 at boca? slave ?
com7 at boca? slave ?
com8 at boca? slave ?
com9 at boca? slave ?
```

For 16-port BB2016 boards:

```
boca0 at isa? port 0x100 irq 5
com2 at boca? slave ?
com3 at boca? slave ?
com4 at boca? slave ?
com5 at boca? slave ?
com6 at boca? slave ?
com7 at boca? slave ?
com8 at boca? slave ?
com9 at boca? slave ?
boca1 at isa? port 0x140 irq 5
com10 at boca? slave ?
com11 at boca? slave ?
com12 at boca? slave ?
com13 at boca? slave ?
com14 at boca? slave ?
com15 at boca? slave ?
com16 at boca? slave ?
com17 at boca? slave ?
```

(The BB2016 is functionally equivalent to two BB1008 boards, and is configured as such.)

DESCRIPTION

The **boca** driver provides support for BOCA Research BB1004, BB1008 and BB2016 boards that multiplex together up to four, eight or sixteen EIA RS-232C (CCITT V.28) communications interfaces.

Each **boca** device is the master device for up to eight **com** devices. The kernel configuration specifies these **com** devices as slave devices of the **boca** device, as shown in the synopsis. The slave ID given for each **com** device determines which bit in the interrupt multiplexing register is tested to find interrupts for that device. The port specification for the **boca** device is used to compute the base addresses for the **com** subde-

vices and the port for the interrupt multiplexing register.

FILES

`/dev/tty??`

SEE ALSO

`com`(4)

HISTORY

The **boca** driver was written by Charles Hannum, based on the **ast** driver and source code from David Muir Sharnoff. David wishes to acknowledge the assistance of Jason Venner in determining how to use the BOCA boards.

NAME

bpf — Berkeley Packet Filter raw network interface

SYNOPSIS

```
pseudo-device bpfilter
```

DESCRIPTION

The Berkeley Packet Filter provides a raw interface to data link layers in a protocol independent fashion. All packets on the network, even those destined for other hosts, are accessible through this mechanism.

The packet filter appears as a character special device, `/dev/bpf`. After opening the device, the file descriptor must be bound to a specific network interface with the `BIOSETIF` ioctl. A given interface can be shared by multiple listeners, and the filter underlying each descriptor will see an identical packet stream.

Associated with each open instance of a **bpf** file is a user-settable packet filter. Whenever a packet is received by an interface, all file descriptors listening on that interface apply their filter. Each descriptor that accepts the packet receives its own copy.

Reads from these files return the next group of packets that have matched the filter. To improve performance, the buffer passed to read must be the same size as the buffers used internally by **bpf**. This size is returned by the `BIOCGBLEN` ioctl (see below), and under BSD, can be set with `BIOCSBLEN`. Note that an individual packet larger than this size is necessarily truncated.

The packet filter will support any link level protocol that has fixed length headers. Currently, only Ethernet, SLIP and PPP drivers have been modified to interact with **bpf**.

Since packet data is in network byte order, applications should use the `byteorder`(3) macros to extract multi-byte values.

A packet can be sent out on the network by writing to a **bpf** file descriptor. The writes are unbuffered, meaning only one packet can be processed per write. Currently, only writes to Ethernets and SLIP links are supported.

IOCTLS

The `ioctl`(2) command codes below are defined in ⟨`net/bpf.h`⟩. All commands require these includes:

```
#include <sys/types.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <net/bpf.h>
```

Additionally, `BIOCGETIF` and `BIOCSETIF` require `<net/if.h>`.

The (third) argument to the `ioctl`(2) should be a pointer to the type indicated.

`BIOCGBLEN (u_int)`
: Returns the required buffer length for reads on **bpf** files.

`BIOCSBLEN (u_int)`
: Sets the buffer length for reads on **bpf** files. The buffer must be set before the file is attached to an interface with `BIOCSETIF`. If the requested buffer size cannot be accommodated, the closest allowable size will be set and returned in the argument. A read call will result in `EINVAL` if it is passed a buffer that is not this size.

`BIOCGDLT (u_int)`
: Returns the type of the data link layer underlying the attached interface. `EINVAL` is returned if no interface has been specified. The device types, prefixed with “DLT_”, are defined in ⟨`net/bpf.h`⟩.

`BIOCGDLTLIST (struct bpf_dltlist)`
Returns an array of available type of the data link layer underlying the attached interface:

```
struct bpf_dltlist {
        u_int bfl_len;
        u_int *bfl_list;
};
```

The available type is returned to the array pointed to the *bfl_list* field while its length in u_int is supplied to the *bfl_len* field. `ENOMEM` is returned if there is not enough buffer. The *bfl_len* field is modified on return to indicate the actual length in u_int of the array returned. If *bfl_list* is `NULL`, the *bfl_len* field is returned to indicate the required length of an array in u_int.

`BIOCSDLT (u_int)`
Change the type of the data link layer underlying the attached interface. `EINVAL` is returned if no interface has been specified or the specified type is not available for the interface.

`BIOCPROMISC`
Forces the interface into promiscuous mode. All packets, not just those destined for the local host, are processed. Since more than one file can be listening on a given interface, a listener that opened its interface non-promiscuously may receive packets promiscuously. This problem can be remedied with an appropriate filter.

The interface remains in promiscuous mode until all files listening promiscuously are closed.

`BIOCFLUSH`
Flushes the buffer of incoming packets, and resets the statistics that are returned by `BIOCGSTATS`.

`BIOCGETIF (struct ifreq)`
Returns the name of the hardware interface that the file is listening on. The name is returned in the ifr_name field of *`ifr`*. All other fields are undefined.

`BIOCSETIF (struct ifreq)`
Sets the hardware interface associate with the file. This command must be performed before any packets can be read. The device is indicated by name using the `ifr_name` field of the *`ifreq`*. Additionally, performs the actions of `BIOCFLUSH`.

`BIOCSRTIMEOUT, BIOCGRTIMEOUT (struct timeval)`
Set or get the read timeout parameter. The *`timeval`* specifies the length of time to wait before timing out on a read request. This parameter is initialized to zero by `open`(2), indicating no timeout.

`BIOCGSTATS (struct bpf_stat)`
Returns the following structure of packet statistics:

```
struct bpf_stat {
        uint64_t bs_recv;
        uint64_t bs_drop;
        uint64_t bs_capt;
        uint64_t bs_padding[13];
};
```

The fields are:

bs_recv the number of packets received by the descriptor since opened or reset (including any buffered since the last read call);

bs_drop the number of packets which were accepted by the filter but dropped by the kernel because of buffer overflows (i.e., the application's reads aren't keeping up with the packet traffic); and

bs_capt the number of packets accepted by the filter.

`BIOCIMMEDIATE (u_int)`

Enable or disable "immediate mode", based on the truth value of the argument. When immediate mode is enabled, reads return immediately upon packet reception. Otherwise, a read will block until either the kernel buffer becomes full or a timeout occurs. This is useful for programs like `rarpd`(8), which must respond to messages in real time. The default for a new file is off.

`BIOCSETF (struct bpf_program)`

Sets the filter program used by the kernel to discard uninteresting packets. An array of instructions and its length is passed in using the following structure:

```
struct bpf_program {
        u_int bf_len;
        struct bpf_insn *bf_insns;
};
```

The filter program is pointed to by the *bf_insns* field while its length in units of 'struct bpf_insn' is given by the *bf_len* field. Also, the actions of `BIOCFLUSH` are performed.

See section **FILTER MACHINE** for an explanation of the filter language.

`BIOCVERSION (struct bpf_version)`

Returns the major and minor version numbers of the filter language currently recognized by the kernel. Before installing a filter, applications must check that the current version is compatible with the running kernel. Version numbers are compatible if the major numbers match and the application minor is less than or equal to the kernel minor. The kernel version number is returned in the following structure:

```
struct bpf_version {
        u_short bv_major;
        u_short bv_minor;
};
```

The current version numbers are given by `BPF_MAJOR_VERSION` and `BPF_MINOR_VERSION` from ⟨`net/bpf.h`⟩. An incompatible filter may result in undefined behavior (most likely, an error returned by `ioctl`(2) or haphazard packet matching).

`BIOCGHDRCMPLT BIOCSHDRCMPLT (u_int)`

Enable/disable or get the "header complete" flag status. If enabled, packets written to the bpf file descriptor will not have network layer headers rewritten in the interface output routine. By default, the flag is disabled (value is 0).

`BIOCGSEESENT BIOCSSEESENT (u_int)`

Enable/disable or get the "see sent" flag status. If enabled, packets sent will be passed to the filter. By default, the flag is enabled (value is 1).

STANDARD IOCTLS

bpf now supports several standard ioctl(2)'s which allow the user to do async and/or non-blocking I/O to an open file descriptor.

FIONREAD (int)
: Returns the number of bytes that are immediately available for reading.

SIOCGIFADDR (struct ifreq)
: Returns the address associated with the interface.

FIONBIO (int)
: Set or clear non-blocking I/O. If arg is non-zero, then doing a read(2) when no data is available will return -1 and *errno* will be set to EAGAIN. If arg is zero, non-blocking I/O is disabled. Note: setting this overrides the timeout set by BIOCSRTIMEOUT.

FIOASYNC (int)
: Enable or disable async I/O. When enabled (arg is non-zero), the process or process group specified by FIOSETOWN will start receiving SIGIO's when packets arrive. Note that you must do an FIOSETOWN in order for this to take affect, as the system will not default this for you. The signal may be changed via BIOCSRSIG.

FIOSETOWN FIOGETOWN (int)
: Set or get the process or process group (if negative) that should receive SIGIO when packets are available. The signal may be changed using BIOCSRSIG (see above).

BPF HEADER

The following structure is prepended to each packet returned by read(2):

```
struct bpf_hdr {
        struct timeval bh_tstamp;
        uint32_t bh_caplen;
        uint32_t bh_datalen;
        uint16_t bh_hdrlen;
};
```

The fields, whose values are stored in host order, and are:

bh_tstamp The time at which the packet was processed by the packet filter.

bh_caplen The length of the captured portion of the packet. This is the minimum of the truncation amount specified by the filter and the length of the packet.

bh_datalen The length of the packet off the wire. This value is independent of the truncation amount specified by the filter.

bh_hdrlen The length of the BPF header, which may not be equal to *sizeof(struct bpf_hdr)*.

The *bh_hdrlen* field exists to account for padding between the header and the link level protocol. The purpose here is to guarantee proper alignment of the packet data structures, which is required on alignment sensitive architectures and improves performance on many other architectures. The packet filter ensures that the *bpf_hdr* and the *network layer* header will be word aligned. Suitable precautions must be taken when accessing the link layer protocol fields on alignment restricted machines. (This isn't a problem on an Ethernet, since the type field is a short falling on an even offset, and the addresses are probably accessed in a bytewise fashion).

Additionally, individual packets are padded so that each starts on a word boundary. This requires that an application has some knowledge of how to get from packet to packet. The macro BPF_WORDALIGN is defined in ⟨net/bpf.h⟩ to facilitate this process. It rounds up its argument to the nearest word aligned

value (where a word is `BPF_ALIGNMENT` bytes wide).

For example, if '*p*' points to the start of a packet, this expression will advance it to the next packet:

```
p = (char *)p + BPF_WORDALIGN(p->bh_hdrlen + p->bh_caplen)
```

For the alignment mechanisms to work properly, the buffer passed to `read`(2) must itself be word aligned. `malloc`(3) will always return an aligned buffer.

FILTER MACHINE

A filter program is an array of instructions, with all branches forwardly directed, terminated by a **return** instruction. Each instruction performs some action on the pseudo-machine state, which consists of an accumulator, index register, scratch memory store, and implicit program counter.

The following structure defines the instruction format:

```
struct bpf_insn {
        uint16_t code;
        u_char  jt;
        u_char  jf;
        int32_t k;
};
```

The *k* field is used in different ways by different instructions, and the *jt* and *jf* fields are used as offsets by the branch instructions. The opcodes are encoded in a semi-hierarchical fashion. There are eight classes of instructions: BPF_LD, BPF_LDX, BPF_ST, BPF_STX, BPF_ALU, BPF_JMP, BPF_RET, and BPF_MISC. Various other mode and operator bits are or'd into the class to give the actual instructions. The classes and modes are defined in ⟨`net/bpf.h`⟩.

Below are the semantics for each defined BPF instruction. We use the convention that A is the accumulator, X is the index register, P[] packet data, and M[] scratch memory store. P[i:n] gives the data at byte offset "i" in the packet, interpreted as a word (n=4), unsigned halfword (n=2), or unsigned byte (n=1). M[i] gives the i'th word in the scratch memory store, which is only addressed in word units. The memory store is indexed from 0 to BPF_MEMWORDS-1. *k*, *jt*, and *jf* are the corresponding fields in the instruction definition. "len" refers to the length of the packet.

BPF_LD

These instructions copy a value into the accumulator. The type of the source operand is specified by an "addressing mode" and can be a constant (**BBPF_IMM**), packet data at a fixed offset (**BPF_ABS**), packet data at a variable offset (**BPF_IND**), the packet length (**BPF_LEN**), or a word in the scratch memory store (**BPF_MEM**). For **BPF_IND** and **BPF_ABS**, the data size must be specified as a word (**BPF_W**), halfword (**BPF_H**), or byte (**BPF_B**). The semantics of all the recognized BPF_LD instructions follow.

BPF_LD+BPF_W+BPF_ABS	A <- P[k:4]
BPF_LD+BPF_H+BPF_ABS	A <- P[k:2]
BPF_LD+BPF_B+BPF_ABS	A <- P[k:1]
BPF_LD+BPF_W+BPF_IND	A <- P[X+k:4]
BPF_LD+BPF_H+BPF_IND	A <- P[X+k:2]
BPF_LD+BPF_B+BPF_IND	A <- P[X+k:1]
BPF_LD+BPF_W+BPF_LEN	A <- len
BPF_LD+BPF_IMM	A <- k
BPF_LD+BPF_MEM	A <- M[k]

BPF_LDX

These instructions load a value into the index register. Note that the addressing modes are more restricted than those of the accumulator loads, but they include **BPF_MSH**, a hack

for efficiently loading the IP header length.

BPF_LDX+BPF_W+BPF_IMM X <- k
BPF_LDX+BPF_W+BPF_MEM X <- M[k]
BPF_LDX+BPF_W+BPF_LEN X <- len
BPF_LDX+BPF_B+BPF_MSH X <- 4*(P[k:1]&0xf)

BPF_ST

This instruction stores the accumulator into the scratch memory. We do not need an addressing mode since there is only one possibility for the destination.

BPF_ST M[k] <- A

BPF_STX

This instruction stores the index register in the scratch memory store.

BPF_STX M[k] <- X

BPF_ALU

The alu instructions perform operations between the accumulator and index register or constant, and store the result back in the accumulator. For binary operations, a source mode is required (**BPF_K** or **BPF_X**).

BPF_ALU+BPF_ADD+BPF_K A <- A + k
BPF_ALU+BPF_SUB+BPF_K A <- A - k
BPF_ALU+BPF_MUL+BPF_K A <- A * k
BPF_ALU+BPF_DIV+BPF_K A <- A / k
BPF_ALU+BPF_AND+BPF_K A <- A & k
BPF_ALU+BPF_OR+BPF_K A <- A | k
BPF_ALU+BPF_LSH+BPF_K A <- A << k
BPF_ALU+BPF_RSH+BPF_K A <- A >> k
BPF_ALU+BPF_ADD+BPF_X A <- A + X
BPF_ALU+BPF_SUB+BPF_X A <- A - X
BPF_ALU+BPF_MUL+BPF_X A <- A * X
BPF_ALU+BPF_DIV+BPF_X A <- A / X
BPF_ALU+BPF_AND+BPF_X A <- A & X
BPF_ALU+BPF_OR+BPF_X A <- A | X
BPF_ALU+BPF_LSH+BPF_X A <- A << X
BPF_ALU+BPF_RSH+BPF_X A <- A >> X
BPF_ALU+BPF_NEG A <- -A

BPF_JMP

The jump instructions alter flow of control. Conditional jumps compare the accumulator against a constant (**BPF_K**) or the index register (**BPF_X**). If the result is true (or non-zero), the true branch is taken, otherwise the false branch is taken. Jump offsets are encoded in 8 bits so the longest jump is 256 instructions. However, the jump always (**BPF_JA**) opcode uses the 32 bit *k* field as the offset, allowing arbitrarily distant destinations. All conditionals use unsigned comparison conventions.

BPF_JMP+BPF_JA pc += k
BPF_JMP+BPF_JGT+BPF_K pc += (A > k) ? jt : jf
BPF_JMP+BPF_JGE+BPF_K pc += (A ≥ k) ? jt : jf
BPF_JMP+BPF_JEQ+BPF_K pc += (A == k) ? jt : jf
BPF_JMP+BPF_JSET+BPF_K pc += (A & k) ? jt : jf

BPF_JMP+BPF_JGT+BPF_X pc += (A > X) ? jt : jf
BPF_JMP+BPF_JGE+BPF_X pc += (A ≥ X) ? jt : jf
BPF_JMP+BPF_JEQ+BPF_X pc += (A == X) ? jt : jf
BPF_JMP+BPF_JSET+BPF_X pc += (A & X) ? jt : jf

BPF_RET

The return instructions terminate the filter program and specify the amount of packet to accept (i.e., they return the truncation amount). A return value of zero indicates that the packet should be ignored. The return value is either a constant (**BPF_K**) or the accumulator (**BPF_A**).

BPF_RET+BPF_A accept A bytes
BPF_RET+BPF_K accept k bytes

BPF_MISC

The miscellaneous category was created for anything that doesn't fit into the above classes, and for any new instructions that might need to be added. Currently, these are the register transfer instructions that copy the index register to the accumulator or vice versa.

BPF_MISC+BPF_TAX X <- A
BPF_MISC+BPF_TXA A <- X

The BPF interface provides the following macros to facilitate array initializers:

BPF_STMT (opcode, operand)
BPF_JUMP (opcode, operand, true_offset, false_offset)

SYSCTLS

The following sysctls are available when **bpf** is enabled:

`net.bpf.maxbufsize`	Sets the maximum buffer size available for **bpf** peers.
`net.bpf.stats`	Shows **bpf** statistics. They can be retrieved with the `netstat`(1) utility.
`net.bpf.peers`	Shows the current **bpf** peers. This is only available to the super user and can also be retrieved with the `netstat`(1) utility.

FILES

`/dev/bpf`

EXAMPLES

The following filter is taken from the Reverse ARP Daemon. It accepts only Reverse ARP requests.

```
struct bpf_insn insns[] = {
        BPF_STMT(BPF_LD+BPF_H+BPF_ABS, 12),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, ETHERTYPE_REVARP, 0, 3),
        BPF_STMT(BPF_LD+BPF_H+BPF_ABS, 20),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, REVARP_REQUEST, 0, 1),
        BPF_STMT(BPF_RET+BPF_K, sizeof(struct ether_arp) +
            sizeof(struct ether_header)),
        BPF_STMT(BPF_RET+BPF_K, 0),
};
```

This filter accepts only IP packets between host 128.3.112.15 and 128.3.112.35.

```
struct bpf_insn insns[] = {
        BPF_STMT(BPF_LD+BPF_H+BPF_ABS, 12),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, ETHERTYPE_IP, 0, 8),
```

```
        BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 26),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 0x8003700f, 0, 2),
        BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 30),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 0x80037023, 3, 4),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 0x80037023, 0, 3),
        BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 30),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 0x8003700f, 0, 1),
        BPF_STMT(BPF_RET+BPF_K, (u_int)-1),
        BPF_STMT(BPF_RET+BPF_K, 0),
};
```

Finally, this filter returns only TCP finger packets. We must parse the IP header to reach the TCP header. The **BPF_JSET** instruction checks that the IP fragment offset is 0 so we are sure that we have a TCP header.

```
struct bpf_insn insns[] = {
        BPF_STMT(BPF_LD+BPF_H+BPF_ABS, 12),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, ETHERTYPE_IP, 0, 10),
        BPF_STMT(BPF_LD+BPF_B+BPF_ABS, 23),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, IPPROTO_TCP, 0, 8),
        BPF_STMT(BPF_LD+BPF_H+BPF_ABS, 20),
        BPF_JUMP(BPF_JMP+BPF_JSET+BPF_K, 0x1fff, 6, 0),
        BPF_STMT(BPF_LDX+BPF_B+BPF_MSH, 14),
        BPF_STMT(BPF_LD+BPF_H+BPF_IND, 14),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 79, 2, 0),
        BPF_STMT(BPF_LD+BPF_H+BPF_IND, 16),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 79, 0, 1),
        BPF_STMT(BPF_RET+BPF_K, (u_int)-1),
        BPF_STMT(BPF_RET+BPF_K, 0),
};
```

SEE ALSO

`ioctl`(2), `read`(2), `select`(2), `signal`(3), `tcpdump`(8)

S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture", *Proceedings of the 1993 Winter USENIX*.

HISTORY

The Enet packet filter was created in 1980 by Mike Accetta and Rick Rashid at Carnegie-Mellon University. Jeffrey Mogul, at Stanford, ported the code to BSD and continued its development from 1983 on. Since then, it has evolved into the ULTRIX Packet Filter at DEC, a STREAMS NIT module under SunOS 4.1, and BPF.

AUTHORS

Steven McCanne, of Lawrence Berkeley Laboratory, implemented BPF in Summer 1990. The design was in collaboration with Van Jacobson, also of Lawrence Berkeley Laboratory.

BUGS

The read buffer must be of a fixed size (returned by the `BIOCGBLEN` ioctl).

A file that does not request promiscuous mode may receive promiscuously received packets as a side effect of another file requesting this mode on the same hardware interface. This could be fixed in the kernel with additional processing overhead. However, we favor the model where all files must assume that the interface is promiscuous, and if so desired, must use a filter to reject foreign packets.

Data link protocols with variable length headers are not currently supported.

Under SunOS, if a BPF application reads more than 2^31 bytes of data, read will fail in `EINVAL`. You can either fix the bug in SunOS, or lseek to 0 when read fails for this reason.

“Immediate mode” and the “read timeout” are misguided features. This functionality can be emulated with non-blocking mode and `select`(2).

NAME

bpp — parallel port driver

SYNOPSIS

```
bpp0 at sbus?
```

DESCRIPTION

This driver provides access to parallel ports.

FILES

`/dev/bpp`

NAME

brgphy — Driver for Broadcom BCM5400-family Gigabit Ethernet PHYs

SYNOPSIS

```
brgphy* at mii? phy ?
```

DESCRIPTION

The **brgphy** driver supports the Broadcom BCM5400-family Gigabit Ethernet PHYs. These PHYs are found on a variety of Gigabit Ethernet interfaces.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

bridge — network bridge device

SYNOPSIS

```
pseudo-device bridge
```

DESCRIPTION

The **bridge** driver creates a logical link between two or more IEEE 802 networks that use the same (or "similar enough") framing format. For example, it is possible to bridge Ethernet and 802.11 networks together, but it is not possible to bridge Ethernet and Token Ring together.

To use **bridge**, the administrator must first create the interface and configure the bridge parameters. The bridge is created using the ifconfig(8) **create** subcommand. The learning and forwarding behavior and other parameters of a bridge are configured by the brconfig(8) utility.

A bridge can be used to provide several services, such as a simple 802.11-to-Ethernet bridge for wireless hosts, and traffic isolation.

A bridge works like a hub, forwarding traffic from one interface to another. Multicast and broadcast packets are always forwarded to all interfaces that are part of the bridge. For unicast traffic, the bridge learns which MAC addresses are associated with which interfaces and will forward the traffic selectively.

The **bridge** driver implements the IEEE 802.1D Spanning Tree protocol (STP). Spanning Tree is used to detect and remove loops in a network topology.

Transparent filtering for IP and IPv6 packets can be added with the kernel configuration option **options BRIDGE_IPF**.

When filtering is enabled, bridged packets will pass through the filter inbound on the originating interface and outbound on the appropriate interfaces. ARP and REVARP packets are forwarded without being filtered and others that are not IP nor IPv6 packets are not forwarded when filtering is enabled.

Note that packets to and from the bridging host will be seen by the filter on the interface with the appropriate address configured as well as on the interface on which the packet arrives or departs.

SEE ALSO

etherip(4), options(4), brconfig(8), ipf(8)

HISTORY

The **bridge** driver first appeared in NetBSD 1.6.

AUTHORS

The **bridge** driver was originally written by Jason L. Wright ⟨jason@thought.net⟩ as part of an undergraduate independent study at the University of North Carolina at Greensboro.

This version of the **bridge** driver has been heavily modified from the original version by Jason R. Thorpe ⟨thorpej@wasabisystems.com⟩.

BUGS

The **bridge** driver currently supports only Ethernet and Ethernet-like (e.g. 802.11) network devices, with exactly the same interface MTU size as the bridge device.

The **bridge** driver currently does not support snooping via bpf(4).

NAME

bt3c — 3Com Bluetooth PC Card driver

SYNOPSIS

bt3c* at pcmcia? function ?

DESCRIPTION

The **bt3c** driver provides support for the 3Com Bluetooth PC Card, model 3CRWB6096, to the Bluetooth protocol stack.

FIRMWARE

This card needs firmware loaded before it will work. Due to copyright restrictions we cannot distribute the firmware with NetBSD, but if you have the card then you should have received a CD with the drivers on, or you may download the latest version from the 3Com website. Create a directory named **bt3c** in the search path of the `firmload`(9) kernel subsystem. Now, extract the driver archive and find the firmware file called **BT3CPCC.bin**, and place this file in the newly created directory. The firmware will be loaded automatically as needed.

DIAGNOSTICS

bt3c%d: Cannot open firmware

This will be printed to the console if the device cannot open the firmware file as described above.

bt3c%d: Antenna In

bt3c%d: Antenna Out

If the kernel is compiled with the `DIAGNOSTIC` option, these messages will be produced on the console when the card antenna position is changed.

bt3c%d: sleeping

bt3c%d: waking up

These messages will be produced when the card is enabled or disabled due to power change events.

SEE ALSO

`bluetooth`(4), `pcmcia`(4), `firmload`(9)

HISTORY

This **bt3c** device driver was written by Iain Hibbert using FreeBSD and BlueZ drivers as a reference. It first appeared in NetBSD 4.0.

BUGS

NAME

btbc — AnyCom BlueCard driver

SYNOPSIS

btbc* at pcmcia? function ?

DESCRIPTION

The **btbc** driver provides support for the AnyCom BlueCard (LSE041, LSE039, LSE139), to the Bluetooth protocol stack.

SEE ALSO

bluetooth(4), pcmcia(4),

HISTORY

This **btbc** device driver was written by KIYOHARA Takashi using Linux bluecard_cs driver as a reference. It first appeared in NetBSD 4.0.

BUGS

NAME

bthidev — Bluetooth Human Interface Device support

SYNOPSIS

```
bthidev* at bthub?

btkbd*   at bthidev? reportid ?
btms*    at bthidev? reportid ?
```

DESCRIPTION

The **bthidev** driver handles all Bluetooth Human Interface Devices. Each HID device can have several components, e.g., a keyboard and a mouse. These components use different report identifiers to distinguish which component data is coming from. The **bthidev** driver may have several children attached that handle particular components and dispatches data to them based on the report id.

Normally, Bluetooth HIDs will be attached using the `btdevctl`(8) program. The following properties are used by the **bthidev** driver during autoconfiguration:

local-bdaddr	Local device address.
remote-bdaddr	Remote device address.
service-name	The **bthidev** driver matches the 'HID' service.
control-psm	This, if set, will indicate the PSM to use for the Control channel. If not set, `L2CAP_PSM_HID_CNTL` will be used.
interrupt-psm	This, if set, will indicate the PSM to use for the Interrupt channel. If not set, `L2CAP_PSM_HID_INTR` will be used.
descriptor	This required binary blob is the HID descriptor containing information about reports the device will produce, and obtained via SDP.
reconnect	If this boolean value is set, and is true, then the **bthidev** driver will initiate reconnections to the remote device when no connection is present.
link-mode	This optional string represents the link mode of the baseband link, and may be one of 'auth', 'encrypt', or 'secure'.

When the **bthidev** driver has configured its children, it will initiate a connection to the remote device. If this fails and the reconnect flag is not set, it will then wait for the device to initiate the connection.

SEE ALSO

`bluetooth`(4), `bthub`(4), `btkbd`(4), `btms`(4), `btdevctl`(8)

HISTORY

The **bthidev** driver was written by Iain Hibbert under the sponsorship of Itronix, Inc. and first appeared in NetBSD 4.0.

NAME

bthub — Bluetooth Remote Device Hub

SYNOPSIS

```
bthub* at bt3c?
bthub* at btbc?
bthub* at btuart?
bthub* at ubt?

bthidev* at bthub?
btsco* at bthub?
```

DESCRIPTION

The **bthub** device is used to attach remote Bluetooth devices to the system, and will attach to Bluetooth controllers as they are enabled.

CONFIGURATION

Normally, Bluetooth Remote Devices will be configured on the **bthub** using the `btdevctl`(8) program, which passes a `proplib`(3) dictionary to the control file `/dev/bthub` with the `BTDEV_ATTACH` and `BTDEV_DETACH` `ioctl`(2) commands.

The following properties are used by **bthub**:

local-bdaddr — Local `BD_ADDR`. This required property should be a six byte data blob.

remote-bdaddr — Remote `BD_ADDR`. This required property should be a six byte data blob.

service-name — Service name. This required property should be a string indicating the service to configure, and may be one of the following:

HF	Handsfree, see `btsco`(4).
HID	Human Interface Device, see `bthidev`(4).
HSET	Headset, see `btsco`(4).

Properties used by the configured device are listed in the appropriate device manual page.

FILES

`/dev/bthub`

SEE ALSO

`bluetooth`(4), `bthidev`(4), `btsco`(4), `btdevctl`(8)

HISTORY

The **bthub** driver was written by Iain Hibbert under the sponsorship of Itronix, Inc. and first appeared in NetBSD 4.0.

NAME

btkbd — Bluetooth keyboard support

SYNOPSIS

```
btkbd* at bthidev? reportid ?
wskbd* at btkbd? console ?

options     BTKBD_REPEAT
options     BTKBD_LAYOUT=XXX
```

DESCRIPTION

The **btkbd** driver provides support for Bluetooth wireless keyboards.

Bluetooth keyboards are configured using the `btdevctl`(8) program, and provide system access through the `wscons`(4) driver.

SEE ALSO

`bluetooth`(4), `bthidev`(4), `wskbd`(4), `btdevctl`(8)

HISTORY

The **btkbd** driver appeared in NetBSD 4.0 and was written by Iain Hibbert under the sponsorship of Itronix, Inc.

BUGS

Due to the configuration & connection requirements, Bluetooth keyboards cannot be used until the system is fully booted.

Bluetooth keyboards cannot be the system console

NAME

btms — Bluetooth mouse support

SYNOPSIS

```
btms* at bthidev? reportid ?
wsmouse*    at btms?
```

DESCRIPTION

The **btms** driver provides support for Bluetooth wireless mice.

Bluetooth mice must be configured with the `btdevctl`(8) program and provide system access through the `wscons`(4) driver.

SEE ALSO

`bluetooth`(4), `bthidev`(4), `wsmouse`(4), `btdevctl`(8)

HISTORY

The **btms** driver appeared in NetBSD 4.0 and was written by Iain Hibbert under the sponsorship of Itronix, Inc.

NAME

btsco — Bluetooth SCO Audio

SYNOPSIS

```
btsco* at bthub?
audio* at audiobus?
```

DESCRIPTION

The **btsco** driver provides support for Bluetooth SCO Audio devices through the audio(4) driver.

The **btsco** driver must be configured at run time with the btdevctl(8) program. The following properties are used by the **btsco** driver during autoconfiguration:

local-bdaddr
: Local device address.

remote-bdaddr
: Remote device address.

service-name
: The **btsco** driver matches the 'HF' and 'HSET' services. For the 'HF' service, the **btsco** device will, on open(2), listen for incoming connections from the remote device. Otherwise, **btsco** will attempt to initiate a connection to the remote device.

rfcomm-channel
: This integer value is not used directly, but will be stored and passed via the BTSCO_INFO ioctl as below:

SCO connections require a baseband connection between the two devices before they can be created. The **btsco** driver does not create this, but can provide information to facilitate an application setting up a control channel prior to use, via the BTSCO_INFO ioctl(2) call on the mixer device, which returns a *btsco_info* structure as follows:

```
#include <dev/bluetooth/btsco.h>

struct btsco_info {
        bdaddr_t        laddr;          /* controller bdaddr */
        bdaddr_t        raddr;          /* headset bdaddr */
        uint8_t         channel;        /* RFCOMM channel */
        int             vgs;            /* mixer index speaker */
        int             vgm;            /* mixer index mic */
};

#define BTSCO_INFO      _IOR('b', 16, struct btsco_info)
```

The **btsco** driver can be configured to act in Connect or Listen mode. In Connect mode, the **btsco** driver will initate a connection to the remote device on an open(2) call, whereas in Listen mode, open(2) will block until the remote device initiates the connection.

SEE ALSO

bthset(1), ioctl(2), audio(4), bluetooth(4), bthub(4), btdevctl(8)

HISTORY

The **btsco** driver was written for NetBSD 4.0 by Iain Hibbert under the sponsorship of Itronix, Inc.

BUGS

btsco takes no notice of the HCI Voice Setting in the Bluetooth controller, and this must be 0x0060 (the default) as alternate values are currently unsupported.

NAME

btuart — Bluetooth HCI UART driver

SYNOPSIS

```
pseudo-device btuart
```

DESCRIPTION

The **btuart** driver provides a tty(4) line discipline to send and receive Bluetooth packets over a serial line, as described in the "Bluetooth Host Controller Interface [Transport Layer] specification, Vol 4 part A."

SEE ALSO

bcsp(4) bluetooth(4), btattach(8)

HISTORY

The **btuart** driver was written with refererence to the BlueZ drivers for Linux, and first appeared in NetBSD 5.0.

AUTHORS

KIYOHARA Takashi ⟨kiyohara@kk.iij4u.or.jp⟩

NAME

bwtwo — Sun monochromatic frame buffer

SYNOPSIS

bwtwo* at sbus? slot ? offset ?

DESCRIPTION

The **bwtwo** is a memory based black and white frame buffer. It supports the minimal ioctl's needed to run X(1).

SEE ALSO

cgsix(4), cgthree(4)

NAME

bwtwo — Sun monochromatic frame buffer

SYNOPSIS

```
bwtwo0 at obmem0 addr 0x700000
bwtwo0 at obio addr 0x0
```

DESCRIPTION

The **bwtwo** is a memory based black and white frame buffer. It supports the minimal ioctl's needed to run X(1).

NAME

bwtwo — Sun monochromatic frame buffer

SYNOPSIS

```
bwtwo0 at obmem0 addr ?
```

DESCRIPTION

The **bwtwo** is a memory based black and white frame buffer. It supports the minimal ioctl's needed to run X(1).

SEE ALSO

cgtwo(4), cgfour(4)

NAME

cac — Compaq array controller driver

SYNOPSIS

```
cac* at eisa? slot ?
cac* at pci? dev ? function ?
```

DESCRIPTION

The **cac** driver provides basic message passing and DMA support for Compaq array controllers. Disk arrays are supported by the **ld** driver.

HARDWARE

The **cac** driver provides support for the following controllers:

Compaq Integrated Array
Compaq IAES
Compaq IDA
Compaq IDA-2
Compaq RAID LC2
Compaq Smart Array 221
Compaq Smart Array 3100ES
Compaq Smart Array 3200
Compaq Smart Array 4200
Compaq Smart Array 4250ES
Compaq Smart Array 431
Compaq SMART
Compaq SMART-2/E
Compaq SMART-2/P
Compaq SMART-2DH
Compaq SMART-2SL

SEE ALSO

intro(4), ld(4)

HISTORY

The **cac** driver first appeared in NetBSD 1.5.

NAME

cardbus, **cardslot**, **cbb** — CardBus driver

SYNOPSIS

```
cbb*      at pci? dev? function ?
cardslot* at cbb?
cardbus*  at cardslot?
pcmcia*   at cardslot?
XX*       at cardbus? function ?
```

DESCRIPTION

NetBSD provides machine-independent bus support and drivers for CardBus devices.

The **cbb** device represents the CardBus controller. Each controller has a number of slots, represented by the **cardslot** devices. A slot can have either a CardBus card or a PCMCIA card, which are attached with the **cardbus** or **pcmcia** devices, respectively.

SUPPORTED DEVICES

NetBSD includes the following machine-independent CardBus drivers, sorted by function and driver name:

Network interfaces

ath	Atheros 5210/5211/5212 802.11
atw	ADMtek ADM8211 (802.11)
ex	3Com 3c575TX and 3c575BTX
fxp	Intel i8255x
ral	Ralink Technology RT25x0 (802.11)
rtk	Realtek 8129/8139
rtw	Realtek 8180L (802.11)
tlp	DECchip 21143

Serial interfaces

com	Modems and serial cards

SCSI controllers

adv	AdvanSys 1200[A,B], 9xx[U,UA]
ahc	Adaptec ADP-1480
njs	Workbit NinjaSCSI-32

USB controllers

ehci	Enhanced Host Controller (2.0)
ohci	Open Host Controller
uhci	Universal Host Controller

IEEE1394 controllers

fwohci	OHCI controller

DIAGNOSTICS

cbb devices may not be properly handled by the system BIOS on i386-family systems. If, on an i386-family system, the **cbb** driver reports

```
cbb0: NOT USED because of unconfigured interrupt
```

then enabling

```
options PCI_ADDR_FIXUP
options PCI_BUS_FIXUP
options PCI_INTR_FIXUP
```

or (if ACPI is in use)

```
options PCI_INTR_FIXUP_DISABLED
```

in the kernel configuration might be of use.

SEE ALSO

adv(4), ahc(4), ath(4), atw(4), com(4), ehci(4), ex(4), fxp(4), njs(4), ohci(4), options(4), pci(4), pcmcia(4), ral(4), rtk(4), rtw(4), tlp(4), uhci(4)

HISTORY

The **cardbus** driver appeared in NetBSD 1.5.

BUGS

Memory space conflicts

NetBSD maps memory on Cardbus and PCMCIA cards in order to access the cards (including reading CIS tuples on PCMCIA cards) and access the devices using the RBUS abstraction. When the mapping does not work, PCMCIA cards are typically ignored on insert, and Cardbus cards are recognized but nonfunctional. On i386, the kernel has a heuristic to choose a memory address for mapping, defaulting to 1 GB, but choosing 0.5 GB on machines with less than 192 MB RAM and 2 GB on machines with more than 1 GB of RAM. The intent is to use an address that is larger than available RAM, but low enough to work; some systems seem to have trouble with addresses requiring more than 20 address lines. On i386, the following kernel configuration line disables the heuristics and forces Cardbus memory space to be mapped at 512M; this value makes Cardbus support (including PCMCIA attachment under a cbb) work on some notebook models, including the IBM Thinkpad 600E (2645-4AU) and the Compaq ARMADA M700:

```
options RBUS_MIN_START="0x20000000"
```

NAME

carp — Common Address Redundancy Protocol

SYNOPSIS

pseudo-device carp [*count*]

DESCRIPTION

The **carp** interface is a pseudo-device which implements and controls the CARP protocol. **carp** allows multiple hosts on the same local network to share a set of IP addresses. Its primary purpose is to ensure that these addresses are always available, but in some configurations **carp** can also provide load balancing functionality.

A **carp** interface can be created at runtime using the **ifconfig carp***N* **create** command.

To use **carp**, the administrator needs to configure at minimum a common virtual host ID and virtual host IP address on each machine which is to take part in the virtual group. Additional parameters can also be set on a per-interface basis: **advbase** and **advskew**, which are used to control how frequently the host sends advertisements when it is the master for a virtual host, and **pass** which is used to authenticate carp advertisements. Finally **carpdev** is used to specify which interface the **carp** device attaches to. If unspecified, the kernel attempts to set carpdev by looking for another interface with the same subnet. These configurations can be done using `ifconfig`(8), or through the `SIOCSVH` ioctl.

Additionally, there are a number of global parameters which can be set using `sysctl`(8):

net.inet.carp.allow	Accept incoming **carp** packets. Enabled by default.
net.inet.carp.preempt	Allow virtual hosts to preempt each other. It is also used to failover **carp** interfaces as a group. When the option is enabled and one of the **carp** enabled physical interfaces goes down, advskew is changed to 240 on all **carp** interfaces. See also the first example. Disabled by default.
net.inet.carp.log	Log bad **carp** packets. Disabled by default.
net.inet.carp.arpbalance	Balance local traffic using ARP. Disabled by default.

EXAMPLES

For firewalls and routers with multiple interfaces, it is desirable to failover all of the **carp** interfaces together, when one of the physical interfaces goes down. This is achieved by the preempt option. Enable it on both host A and B:

```
# sysctl -w net.inet.carp.preempt=1
```

Assume that host A is the preferred master and 192.168.1.x/24 is configured on one physical interface and 192.168.2.y/24 on another. This is the setup for host A:

```
# ifconfig carp0 create
# ifconfig carp0 vhid 1 pass mekmitasdigoat 192.168.1.1 \
        netmask 255.255.255.0
# ifconfig carp1 create
# ifconfig carp1 vhid 2 pass mekmitasdigoat 192.168.2.1/24 \
        netmask 255.255.255.0
```

The setup for host B is identical, but it has a higher advskew:

```
# ifconfig carp0 create
# ifconfig carp0 vhid 1 advskew 100 pass mekmitasdigoat \
        192.168.1.1 netmask 255.255.255.0
```

```
# ifconfig carp1 create
# ifconfig carp1 vhid 2 advskew 100 pass mekmitasdigoat \
        192.168.2.1 netmask 255.255.255.0
```

Because of the preempt option, when one of the physical interfaces of host A fails, advskew is adjusted to 240 on all its **carp** interfaces. This will cause host B to preempt on both interfaces instead of just the failed one.

In order to set up an ARP balanced virtual host, it is necessary to configure one virtual host for each physical host which would respond to ARP requests and thus handle the traffic. In the following example, two virtual hosts are configured on two hosts to provide balancing and failover for the IP address 192.168.1.10.

First the **carp** interfaces on Host A are configured. The **advskew** of 100 on the second virtual host means that its advertisements will be sent out slightly less frequently.

```
# ifconfig carp0 create
# ifconfig carp0 vhid 1 pass mekmitasdigoat 192.168.1.10 \
        netmask 255.255.255.0
# ifconfig carp1 create
# ifconfig carp1 vhid 2 advskew 100 pass mekmitasdigoat \
        192.168.1.10 netmask 255.255.255.0
```

The configuration for host B is identical, except the skew is on virtual host 1 rather than virtual host 2.

```
# ifconfig carp0 create
# ifconfig carp0 vhid 1 advskew 100 pass mekmitasdigoat \
        192.168.1.10 netmask 255.255.255.0
# ifconfig carp1 create
# ifconfig carp1 vhid 2 pass mekmitasdigoat 192.168.1.10 \
        netmask 255.255.255.0
```

Finally, the ARP balancing feature must be enabled on both hosts:

```
# sysctl -w net.inet.carp.arpbalance=1
```

When the hosts receive an ARP request for 192.168.1.10, the source IP address of the request is used to compute which virtual host should answer the request. The host which is master of the selected virtual host will reply to the request, the other(s) will ignore it.

This way, locally connected systems will receive different ARP replies and subsequent IP traffic will be balanced among the hosts. If one of the hosts fails, the other will take over the virtual MAC address, and begin answering ARP requests on its behalf.

Note: ARP balancing only works on the local network segment. It cannot balance traffic that crosses a router, because the router itself will always be balanced to the same virtual host.

SEE ALSO

`netstat`(1), `sysctl`(3), `arp`(4), `arp`(8), `ifconfig`(8), `sysctl`(8)

HISTORY

The **carp** device first appeared in OpenBSD 3.5.

NAME

ccd — Concatenated disk driver

SYNOPSIS

pseudo-device ccd [*count*]

DESCRIPTION

The **ccd** driver provides the capability of combining one or more disks/partitions into one virtual disk.

This document assumes that you're familiar with how to generate kernels, how to properly configure disks and pseudo-devices in a kernel configuration file, and how to partition disks.

Note that the 'raw' partitions of the disks must not be combined. Each component partition should be offset at least one cylinder from the beginning of the component disk. This avoids potential conflicts between the component disk's disklabel and the **ccd**'s disklabel. The kernel will only allow component partitions of type FS_CCD. But for now, it allows partition of all types since some port lacks support of an on-disk BSD disklabel. The partition of FS_UNUSED may be rejected because device driver of component disk will refuse it.

In order to compile in support for the **ccd**, you must add a line similar to the following to your kernel configuration file:

```
pseudo-device     ccd     4       # concatenated disk devices
```

The count argument is how many **ccd**s memory is allocated for at boot time. In this example, no more than 4 **ccd**s may be configured.

A **ccd** may be either serially concatenated or interleaved. To serially concatenate the partitions, specify the interleave factor of 0.

If a **ccd** is interleaved correctly, a "striping" effect is achieved, which can increase performance. Since the interleave factor is expressed in units of DEV_BSIZE, one must account for sector sizes other than DEV_BSIZE in order to calculate the correct interleave. The kernel will not allow an interleave factor less than the size of the largest component sector divided by DEV_BSIZE.

Note that best performance is achieved if all component disks have the same geometry and size. Optimum striping cannot occur with different disk types.

Also note that the total size of concatenated disk may vary depending on the interleave factor even if the exact same components are concatenated. And an old on-disk disklabel may be read after interleave factor change. As a result, the disklabel may contain wrong partition geometry and will cause an error when doing I/O near the end of concatenated disk.

There is a run-time utility that is used for configuring **ccd**s. See ccdconfig(8) for more information.

WARNINGS

If just one (or more) of the disks in a non-mirrored **ccd** fails, the entire file system will be lost.

FILES

/dev/{,r}ccd* **ccd** device special files.

SEE ALSO

config(1), MAKEDEV(8), ccdconfig(8), fsck(8), mount(8), newfs(8)

HISTORY

The concatenated disk driver was originally written at the University of Utah.

NAME

cd — SCSI and ATAPI CD-ROM driver

SYNOPSIS

```
cd* at scsibus? target ? lun ?
cd1 at scsibus0 target 4 lun 0
cd* at atapibus? drive ? flags 0x0000
```

DESCRIPTION

The **cd** driver provides support for a Small Computer Systems Interface (SCSI) bus and Advanced Technology Attachment Packet Interface (ATAPI) Compact Disc-Read Only Memory (CD-ROM) drive. In an attempt to look like a regular disk, the **cd** driver synthesizes a partition table, with one partition covering the entire CD-ROM. It is possible to modify this partition table using `disklabel`(8), but it will only last until the CD-ROM is unmounted. In general the interfaces are similar to those described by `wd`(4) and `sd`(4).

As the SCSI adapter is probed during boot, the SCSI bus is scanned for devices. Any devices found which answer as 'Read-only' type devices will be 'attached' to the **cd** driver.

For the use of flags with ATAPI devices, see `wd`(4).

The system utility `disklabel`(8) may be used to read the synthesized disk label structure, which will contain correct figures for the size of the CD-ROM should that information be required.

KERNEL CONFIGURATION

Any number of CD-ROM devices may be attached to the system regardless of system configuration as all resources are dynamically allocated.

IOCTLS

The following `ioctl`(2) calls which apply to SCSI CD-ROM drives are defined in the header files ⟨`sys/cdio.h`⟩ and ⟨`sys/disklabel.h`⟩.

`DIOCGDINFO`

`DIOCSDINFO` (`struct disklabel`) Read or write the in-core copy of the disklabel for the drive. The disklabel is initialized with information read from the SCSI inquiry commands, and should be the same as the information printed at boot. This structure is defined in `disklabel`(5).

`CDIOCPLAYTRACKS` (`struct ioc_play_track`) Start audio playback given a track address and length. The structure is defined as follows:

```
struct ioc_play_track
{
        u_char  start_track;
        u_char  start_index;
        u_char  end_track;
        u_char  end_index;
};
```

`CDIOCPLAYBLOCKS` (`struct ioc_play_blocks`) Start audio playback given a block address and length. The structure is defined as follows:

```
struct ioc_play_blocks
{
        int     blk;
        int     len;
```

```
};
```

`CDIOCPLAYMSF` `(struct ioc_play_msf)` Start audio playback given a 'minutes-seconds-frames' address and length. The structure is defined as follows:

```
struct ioc_play_msf
{
        u_char  start_m;
        u_char  start_s;
        u_char  start_f;
        u_char  end_m;
        u_char  end_s;
        u_char  end_f;
};
```

`CDIOCREADSUBCHANNEL` `(struct ioc_read_subchannel)` Read information from the subchannel at the location specified by this structure:

```
struct ioc_read_subchannel {
        u_char address_format;
#define CD_LBA_FORMAT  1
#define CD_MSF_FORMAT  2
        u_char data_format;
#define CD_SUBQ_DATA          0
#define CD_CURRENT_POSITION   1
#define CD_MEDIA_CATALOG      2
#define CD_TRACK_INFO         3
        u_char track;
        int     data_len;
        struct  cd_sub_channel_info *data;
};
```

`CDIOREADTOCHEADER` `(struct ioc_toc_header)` Return summary information about the table of contents for the mounted CD-ROM. The information is returned into the following structure:

```
struct ioc_toc_header {
        u_short len;
        u_char  starting_track;
        u_char  ending_track;
};
```

`CDIOREADTOCENTRYS` `(struct ioc_read_toc_entry)` Return information from the table of contents entries mentioned. (Yes, this command name is misspelled). The argument structure is defined as follows:

```
struct ioc_read_toc_entry {
        u_char  address_format;
        u_char  starting_track;
        u_short data_len;
        struct  cd_toc_entry *data;
};
```

The requested data is written into an area of size `data_len` and pointed to by `data`.

`CDIOCSETPATCH` `(struct ioc_patch)` Attach various audio channels to various output channels. The argument structure is defined thusly:

```
struct ioc_patch {
        u_char  patch[4];
        /* one for each channel */
};
```

`CDIOCGETVOL`

`CDIOCSETVOL` `(struct ioc_vol)` Get (set) information about the volume settings of the output channels. The argument structure is as follows:

```
struct  ioc_vol
{
        u_char  vol[4];
        /* one for each channel */
};
```

`CDIOCSETMONO` Patch all output channels to all source channels.

`CDIOCSETSTEREO` Patch left source channel to the left output channel and the right source channel to the right output channel.

`CDIOCSETMUTE` Mute output without changing the volume settings.

`CDIOCSETLEFT`

`CDIOCSETRIGHT` Attach both output channels to the left (right) source channel.

`CDIOCSETDEBUG`

`CDIOCCLRDEBUG` Turn on (off) debugging for the appropriate device.

`CDIOCPAUSE`

`CDIOCRESUME` Pause (resume) audio play, without resetting the location of the read-head.

`CDIOCRESET` Reset the drive.

`CDIOCSTART`

`CDIOCSTOP` Tell the drive to spin-up (-down) the CD-ROM.

`CDIOCALLOW`

`CDIOCPREVENT` Tell the drive to allow (prevent) manual ejection of the CD-ROM disc. Not all drives support this feature.

`CDIOCEJECT` Eject the CD-ROM.

`CDIOCLOADUNLOAD` Cause the ATAPI changer to load or unload discs.

`CDIOCCLOSE` Tell the drive to close its door and load the media. Not all drives support this feature.

In addition the general `scsi`(4) ioctls may be used with the **`cd`** driver, if used against the ‘whole disk’ partition (i.e. `/dev/rcd0d` for the bebox and i386 port, `/dev/rcd0c` for all other ports).

NOTES

When a CD-ROM is changed in a drive controlled by the **`cd`** driver, then the act of changing the media will invalidate the disklabel and information held within the kernel. To stop corruption, all accesses to the device will be discarded until there are no more open file descriptors referencing the device. During this period, all

new open attempts will be rejected. When no more open file descriptors reference the device, the first next open will load a new set of parameters (including disklabel) for the drive.

The audio code in the **cd** driver only support SCSI-2 standard audio commands. Because many CD-ROM manufacturers have not followed the standard, there are many CD-ROM drives for which audio will not work. Some work is planned to support some of the more common 'broken' CD-ROM drives; however, this is not yet under way.

FILES

`/dev/cd[0-9][a-h]` block mode CD-ROM devices
`/dev/rcd[0-9][a-h]` raw mode CD-ROM devices

DIAGNOSTICS

None.

SEE ALSO

`intro`(4), `scsi`(4), `sd`(4), `wd`(4), `disklabel`(5), `disklabel`(8)

HISTORY

The **cd** driver appeared in 386BSD 0.1.

BUGS

The names of the structures used for the third argument to **ioctl**() were poorly chosen, and a number of spelling errors have survived in the names of the **ioctl**() commands.

NAME

cdce — USB Communication Device Class Ethernet driver

SYNOPSIS

cdce* **at uhub? port ?**

DESCRIPTION

The **cdce** driver provides support for USB Host-to-Host (aka USB-to-USB) bridges based on the USB Communication Device Class (CDC) and Ethernet subclass, including the following:

- Prolific PL-2501
- Sharp Zaurus

The USB bridge appears as a regular network interface on both sides, transporting Ethernet frames.

For more information on configuring this device, see `ifconfig`(8).

USB 1.x bridges support speeds of up to 12Mbps, and USB 2.0 speeds of up to 480Mbps.

Packets are received and transmitted over separate USB bulk transfer endpoints.

The **cdce** driver does not support different media types or options.

DIAGNOSTICS

cdce%d: no union descriptor The driver couldn't fetch an interface descriptor from the USB device. For a manually added USB vendor/product, the CDCE_NO_UNION flag can be tried to work around the missing descriptor.

cdce%d: no data interface
cdce%d: could not read endpoint descriptor
cdce%d: unexpected endpoint
cdce%d: could not find data bulk in/out For a manually added USB vendor/product, these errors indicate that the bridge is not compatible with the driver.

cdce%d: watchdog timeout A packet was queued for transmission and a transmit command was issued, however the device failed to acknowledge the transmission before a timeout expired.

cdce%d: no memory for rx list -- packet dropped! Memory allocation through MGETHDR or MCLGET failed, the system is running low on mbufs.

cdce%d: abort/close rx/tx pipe failed
cdce%d: rx/tx list init failed
cdce%d: open rx/tx pipe failed
cdce%d: usb error on rx/tx

SEE ALSO

`arp`(4), `intro`(4), `netintro`(4), `usb`(4), `ifconfig`(8)

Universal Serial Bus Class Definitions for Communication Devices, http://www.usb.org/developers/devclass_docs/usbcdc11.pdf.

Data sheet Prolific PL-2501 Host-to-Host Bridge/Network Controller, http://tech.prolific.com.tw/visitor/fcabdl.asp?fid=20679530.

HISTORY

The **cdce** device driver first appeared in OpenBSD 3.6 and NetBSD 3.0.

AUTHORS

The **cdce** driver was written by Craig Boston ⟨craig@tobuj.gank.org⟩ based on the aue(4) driver written by Bill Paul ⟨wpaul@windriver.com⟩ and ported to OpenBSD by Daniel Hartmeier ⟨dhartmei@openbsd.org⟩.

CAVEATS

Many USB devices notoriously fail to report their class and interfaces correctly. Undetected products might work flawlessly when their vendor and product IDs are added to the driver manually.

NAME

cec — IEEE488 GPIB controller boards

SYNOPSIS

```
cec* at isa? port 0x2b8 irq 5 drq 1
gpib* at cec?
```

DESCRIPTION

The **cec** driver supports GPIB (IEEE488) controller boards based on the NEC uPD7210 GPIB controller chip. The following boards are supported:

- Capital Equipment Corp. IEEE488 board
- Keithley GPIB boards

The following GPIB boards are similar and support should be available reasonably easily:

- HAMEG HO-80 IEEE488 board
- National Instruments PCII board
- Measurement Computing (Computer Boards) ISA GPIB boards

SEE ALSO

gpib(4), isa(4)

HISTORY

The **cec** driver appeared in NetBSD 2.0.

NAME

cfb — PMAG-B CX colour unaccelerated 2-D framebuffer

SYNOPSIS

```
cfb* at tc? slot ? offset ?
wsdisplay* at cfb?
```

DESCRIPTION

The **cfb** driver provides support for the PMAG-B CX colour framebuffer for the TURBOchannel bus. The PMAG-B is an 8 bpp colour framebuffer capable of running at a resolution of 1024-by-864 at 60 Hz.

SEE ALSO

mfb(4), px(4), pxg(4), sfb(4), tc(4), tfb(4), wscons(4)

BUGS

NetBSD/pmax does not currently support the machine-independent wscons(4) interface and uses a machine-dependent version.

NAME

cgd — cryptographic disk driver

SYNOPSIS

pseudo-device cgd [*count*]

DESCRIPTION

The **cgd** driver provides the capability of encrypting blocks on their way to and from a disk or partition.

In order to compile support for the **cgd** into your kernel, you must add the driver to your kernel configuration file. To do this, add a line similar to:

```
pseudo-device   cgd    4      # cryptographic disk driver
```

The count argument defines how many **cgd**'s may be configured at a time.

Encryption Algorithms

Currently the following cryptographic algorithms are supported:

aes-cbc
: AES in CBC mode. AES uses a 128 bit blocksize and can accept keys of length 128, 192, or 256. The default key length is 128.

3des-cbc
: Triple DES in CBC mode. Triple DES uses a 64 bit blocksize and is performed in EDE3 mode with a 168 bit key. The key passed to the kernel is 192 bits but the parity bits are ignored.

blowfish-cbc
: Blowfish in CBC mode. Blowfish uses a 64 bit blocksize and can accept keys between 40 and 448 bits in multiples of 8. It is strongly encouraged that keys be at least 128 bits long. There are no performance advantages of using shorter keys. The default key length is 128 bits.

IV Methods

Currently, the only IV Method supported is *encblkno* (Encrypted Block Number). This method encrypts the block number of the physical disk block with the cipher and key provided and uses that as the IV for CBC mode. This method should ensure that each block has a different IV and that the IV is reasonably unpredictable.

IOCTLS

A **cgd** responds to all of the standard disk `ioctl`(2) calls defined in `sd`(4), and also defines the following:

CGDIOCSET
: configure the **cgd**. This `ioctl`(2) sets up the encryption parameters and points the **cgd** at the underlying disk.

CGDIOCCLR
: unconfigures the **cgd**.

These `ioctl`(2)'s and their associated data structures are defined in `/usr/include/dev/cgdvar.h`.

WARNINGS

It goes without saying that if you forget the passphrase that you used to configure a **cgd**, then you have irrevocably lost all of the data on the disk. Please ensure that you are using an appropriate backup strategy.

FILES

/dev/{,r}cgd* **cgd** device special files.

SEE ALSO

`config`(1), `ioctl`(2), `sd`(4), `MAKEDEV`(8), `cgdconfig`(8)

HISTORY

The **cgd** driver was written by Roland C. Dowdeswell for NetBSD. The **cgd** driver originally appeared in NetBSD 2.0.

NAME

cgeight — Sun 24-bit color frame buffer

SYNOPSIS

```
cgeight0 at obio0 addr 0xfb300000 level 4  ( sun4/300 )
cgeight0 at obio0 addr 0x0b300000 level 4  ( sun4/100 )
```

DESCRIPTION

The **cgeight** is a memory based color frame buffer. Its pixel memory can be mapped into a user process address space by using the mmap(2) system call. The **cgeight** driver supports the minimal ioctl's needed to run X(1).

SEE ALSO

bwtwo(4), cgtwo(4), cgthree(4), cgfour(4), cgsix(4), tcx(4)

NAME

cgfour — Sun 8-bit color frame buffer

SYNOPSIS

```
cgfour0 at obio0 addr 0xfb300000 level 4  ( sun4/300 )
cgfour0 at obio0 addr 0x0b300000 level 4  ( sun4/100 )
```

DESCRIPTION

The **cgfour** is a memory based color frame buffer with overlay plane. Its pixel memory and control planes can be mapped into a user process address space by using the mmap(2) system call. The **cgfour** driver supports the minimal ioctl's needed to run X(1).

SEE ALSO

bwtwo(4), cgtwo(4), cgthree(4), cgsix(4), cgeight(4), tcx(4)

NAME

cgfour — Sun 8-bit color frame buffer

SYNOPSIS

```
cgfour0 at obmem0 addr ?
```

DESCRIPTION

The **cgfour** is a memory based color frame buffer. It supports the minimal ioctl's needed to run X(1).

SEE ALSO

bwtwo(4), cgtwo(4)

NAME

cgfourteen — Sun accelerated 8/24-bit color frame buffer

SYNOPSIS

cgfourteen* **at obio?**

DESCRIPTION

The **cgfourteen** is a memory based color frame buffer, with graphics acceleration and overlay capabilities. Its pixel memory can be mapped into a user process address space by using the mmap(2) system call. The **cgfourteen** driver supports the minimal ioctl's needed to run X(1).

The driver operates by default in cgthree(4) emulation mode, i.e. in 8-bit unaccelerated mode. This emulation does include support for the hardware cursor present on the **cgfourteen**, however.

SEE ALSO

bwtwo(4), cgtwo(4), cgthree(4), cgfour(4), cgsix(4), cgeight(4), tcx(4)

NAME

cgsix — Sun accelerated 8-bit color frame buffer

SYNOPSIS

```
cgsix* at sbus? slot ? offset ?  (sun4c/sun4m)
cgsix0 at obio0 addr 0xfb000000 level 4  (sun4/300)
cgsix0 at obio0 addr 0x0b000000 level 4  (sun4/100)
```

DESCRIPTION

The **cgsix** is a memory based color frame buffer. It supports the minimal ioctl's needed to run X(7).

There are several versions of the **cgsix** board. The Sun part numbers and board types are:

501-1374, 501-1532	P4 GX
501-1505	P4 GX with 3/80 backpanel
501-1481, 501-1645	Sbus double-width GX
501-1672, 501-1996	Sbus GX
501-1717, 501-2018, 501-2039	Sbus GX+
501-2325, 501-2922	Sbus TGX
501-2253, 501-2955	Sbus TGX+

There are also on-board 'GX' cards in the 'SPARCstation IPX' and 'SPARCstation LX' machines.

The 'GX' and 'TGX' cards have 1Mb of on-board memory and support a maximum graphics resolution of 1152x900. The 'GX+' cards have 4Mb of on-board memory and support a maximum resolution of 1280x1024. The 'TGX+' cards have 4Mb of on-board memory and support a maximum resolution of 1600x1280. The 'TGX' (Turbo GX) cards are faster than the 'GX' cards.

The number of supported resolutions varies by card type. All cards support a resolution of 1152x900 at 66Hz. All but the P4 and double-width cards support a resolution of 1152x900 at 76Hz. The cards default to a resolution dependent on the attached monitor (usually 1152x900).

It is only possible to change the resolution of a **cgsix** card from the PROM before the operating system is loaded. For the primary card, this can be done using the 'output-device' PROM field. For example, for a 'TGX+' card, the following PROM command will set the resolution to 1280x1024 at 76Hz:

```
setenv output-device screen:r1280x1024x76
```

For secondary cards, a different method must be used to set the resolution. For a machine with OpenBoot 2.x or 3.x, and assuming a 'TGX' card at Sbus slot 1, the following PROM commands will set the resolution to 1024x768 at 60Hz:

```
nvedit
probe-all
" /iommu/sbus/cgsix@1" select-dev
r1024x768x60
" /iommu/sbus/cgsix@1" " set-resolution" execute-device-method
device-end
install-console
banner
^C
nvstore
setenv use-nvramrc? true
reset
```

For Sun4c machines, the device-path above would be:

```
" /sbus/cgsix@1"
```

For Sun-4 and Sun-3 systems, it is only possible to change PROM fields by altering byte values. For these systems, it is probably easier to use the `eeprom`(8) command to set the *scrsize* field to the desired resolution.

EXAMPLES

```
cgsix0 at obio0 addr 0xfb000000 level 4: cgsix/p4, 1152 x 900, rev 1
cgsix0 at sbus0 slot 0 offset 0x0 level 9: SUNW,501-2325, 1152 x 900, rev 11
cgsix0 at sbus0 slot 0 offset 0x0 level 9: SUNW,501-2253, 1280 x 1024, rev 11
```

SEE ALSO

`bwtwo`(4), `cgeight`(4), `cgfour`(4), `cgfourteen`(4), `cgthree`(4), `cgtwo`(4), `tcx`(4), `eeprom`(8)

BUGS

The double-width ‘GX’ and the ‘GX+’ cards are not compatible with UltraSPARC machines.

On Sun-4 systems using a P4 GX card as console, the *console* field in the PROM must be set to *p4opt*, otherwise the card will not be recognised as the console output device.

Several firmware revisions on **`cgsix`** boards have a terminal emulation bug that shows up when using the screen control sequences for inserting blank lines near the bottom end of the screen (i.e., the control sequences produced by the ‘al’ and ‘AL’ capabilities found in the `termcap`(5) database). The most likely occasion for triggering this bug is to use a full-screen editor such as `vi`(1) on the workstation’s console.

To work around this you can set your `TERM` environment variable to the ‘sun-cgsix’ terminal definition which is the same as the ‘sun’ entry, except that the ‘al’ and ‘AL’ capabilities have been removed (at the cost of making the scrolling of the screen slower).

NAME

cgthree — Sun 8-bit color frame buffer

SYNOPSIS

cgthree* at sbus? slot ? offset ?
cgthree* at obio0 slot ? offset ? (some sun4m models)

DESCRIPTION

The **cgthree** is a memory based color frame buffer. It supports the minimal ioctl's needed to run X(1).

SEE ALSO

bwtwo(4), cgtwo(4), cgfour(4), cgsix(4), cgeight(4), tcx(4)

NAME

cgtwo — Sun 8-bit color frame buffer

SYNOPSIS

```
cgtwo* at vmes0 addr 0xff400000 level 4 vect 0xa8
```

DESCRIPTION

The **cgtwo** is a memory based color frame buffer. Its control pixel memory can be mapped into a user process address space by using the mmap(2) system call. The **cgtwo** driver supports the minimal ioctl's needed to run X(1).

SEE ALSO

bwtwo(4), cgthree(4), cgfour(4), cgsix(4), cgeight(4), tcx(4)

NAME

cgtwo — Sun 8-bit VME bus color frame buffer

SYNOPSIS

```
cgtwo0 at vme2 addr 0x400000 ipl 4 vect 0xA8
```

DESCRIPTION

The **cgtwo** is a memory based, VME bus, color frame buffer. It supports the minimal ioctl's needed to run X(1).

SEE ALSO

bwtwo(4), cgfour(4)

NAME

ch — SCSI media changer driver

SYNOPSIS

```
ch* at scsibus? target ? lun ?
```

DESCRIPTION

The **ch** driver is essentially an `ioctl`(2) interface to a robot on a SCSI bus - a device that will change media (e.g. tapes, CD-ROMs, etc) in and out of drives for that media. The `chio`(1) utility program uses this interface to manipulate such robots.

FILES

`/dev/ch`*u* SCSI bus media changer unit *u*
`/usr/include/sys/chio.h`

DIAGNOSTICS

ch%d: waiting %d seconds for changer to settle... Some changers require a long time to settle out, to do tape inventory, for instance.

ch%d: offline The changer is not responding.

ch%d: warning, READ ELEMENT STATUS avail != count
ch%d: could not sense element address page
ch%d: could not sense capabilities page

SEE ALSO

`chio`(1), `ioctl`(2), `cd`(4), `intro`(4), `scsi`(4), `st`(4)

AUTHORS

Jason R. Thorpe

NAME

chipsfb — Chips Technologies 6555x based graphics chips

SYNOPSIS

```
chipsfb* at pci?
wsdisplay* at chipsfb?
```

DESCRIPTION

The **chipsfb** driver provides support for the CT 65550 and 65554 graphics controllers. Currently it depends on the firmware (usually Open Firmware) to set up the framebuffer, but all graphics operations used by wsdisplay use the blitter.

SEE ALSO

wscons(4), wsdisplay(4)

BUGS

The driver has been tested on macppc only.

NAME

cia — DECchip 2117x Core Logic chipset

SYNOPSIS

```
cia* at mainbus?
pci* at cia?
```

DESCRIPTION

The **cia** driver provides support for the DECchip 2117x Core Logic chipset (PCI controller) found on the AlphaStation 500/600, AlphaServer 1000 and AlphaServer 800/1000A systems.

SEE ALSO

intro(4), mainbus(4), pci(4)

NAME

ciphy — Driver for Cicada 10/100/1000 copper Ethernet PHYs

SYNOPSIS

ciphy* **at mii? phy ?**

DESCRIPTION

The **ciphy** driver supports PHYs commonly integrated on VIA Networkinng Technologies VT6122 Gigabit Ethernet adapters.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

HISTORY

Driver is ported from FreeBSD and first appeared in NetBSD 3.0.

AUTHORS

The driver was originally written by Bill Paul ⟨wpaul@windriver.com⟩.

NAME

cir — Consumer IR (remote control) driver

SYNOPSIS

```
cir* at XX?
```

DESCRIPTION

The **cir** provides access to consumer infrared devices such as remote control receivers and transmitters.

SEE ALSO

irframe(4)

HISTORY

The **cir** driver appeared in NetBSD 1.6.

BUGS

This device is not yet functional.

NAME

ciss — HP/Compaq Smart ARRAY 5/6 RAID controllers

SYNOPSIS

ciss* **at pci? function ?**

DESCRIPTION

The **ciss** driver provides support for the CISS interface implemented by fifth and later generations of the HP/Compaq Smart ARRAY family of controllers.

The CISS interface is defined in the document entitled *CISS Command Interface for SCSI-3 Support Open Specification, Version 1.04, Valence Number 1*, Compaq Computer Corporation, 2000/11/27.

This driver supports several Compaq and HP controllers implementing the CISS interface, including:

- Compaq Smart Array 5300 version 1
- Compaq Smart Array 5300 version 2
- Compaq Smart Array 5i version 1
- Compaq Smart Array 5i version 2
- HP Smart Array 5312
- HP Smart Array 6i
- HP Smart Array 641
- HP Smart Array 642
- HP Smart Array 6400
- HP Smart Array 6400 EM
- HP Smart Array E200
- HP Smart Array E200i
- HP Smart Array P400
- HP Smart Array P400i
- HP Smart Array P600
- HP Smart Array P800
- HP Smart Array V100
- HP Smart Array 1 through 13

These controllers support RAID 0, RAID 1, RAID 5, JBOD, and superpositions of those configurations.

Although the controllers are actual RAID controllers, the **ciss** driver makes them look just like SCSI controllers. All RAID configuration must be done through the controllers' BIOSes.

Hardware from previous generations of this product family may be supported by the cac(4) driver.

SEE ALSO

bio(4), cac(4), intro(4), pci(4), scsi(4), sd(4)

HISTORY

The **ciss** driver first appeared in NetBSD 3.1.

AUTHORS

The **ciss** driver was written by Michael Shalayeff ⟨mickey@openbsd.org⟩, and ported to NetBSD by Tonnerre Lombard ⟨tonnerre@netbsd.org⟩.

NAME

clcs — Cirrus Logic CS4280 audio device driver

SYNOPSIS

```
clcs*  at pci? dev ? function ?
audio* at audiobus?
midi*  at clcs?
```

DESCRIPTION

The **clcs** driver provides support for the Cirrus Logic CS4280 chip. Partial support exists for the CS461x chips, but is disabled. Instead, the wss(4) or sb(4) drivers should be used.

SEE ALSO

audio(4), midi(4), pci(4), sb(4), wss(4)

HISTORY

The **clcs** device driver appeared in NetBSD 1.5.

NAME

clct — Cirrus Logic CS4281 audio device driver

SYNOPSIS

```
clct*  at pci? dev ? function ?
audio* at audiobus?
```

DESCRIPTION

The **clct** driver provides support for the Cirrus Logic CS4281 chip.

SEE ALSO

ac97(4), audio(4), pci(4)

HISTORY

The **clct** device driver appeared in NetBSD 1.6.

NAME

clmpcc — Cirrus Logic CD2400/CD2401 serial communications controller

SYNOPSIS

```
clmpcc0   at pcctwo? ipl 4
```

DESCRIPTION

The **clmpcc** driver provides support for the Cirrus Logic CD2401 Multi-protocol Communications Controller found on Motorola MVME167 and MVME177 single-board computers.

The chip integrates four serial channels in one package, with each channel being completely independent and capable of running in Async (with optional DMA control), Bisync, HDLC/SDLC and X.21 modes. Each channel has 32 bytes of FIFO, split into 16 bytes for the Tx side and 16 bytes for the Rx side.

At the present time, the **clmpcc** driver supports the non-DMA Async mode of operation, using the channel FIFOs to maximize throughput with minimal interrupt overhead.

The Motorola MVME1x7 boards provide a 20MHz master clock to the device, which allows the Tx and Rx side to be independently set to any baud rate in the range 50 to 57600. The device should be capable of running at a baud rate of 115200, however it is not a rate documented in the device's datasheet for Async. mode so is not recommended.

FILES

`/dev/console`

`/dev/ttyC1`

`/dev/ttyC2`

`/dev/ttyC3`

DIAGNOSTICS

clmpcc%d: channel %d command timeout (idle) The chip failed to acknowledge a command sent to the specified channel.

clmpcc%d: Failed to reset chip The **clmpcc** driver was unable to determine if the chip completed its RESET processing.

SEE ALSO

`pcctwo`(4), `tty`(4)

HISTORY

The **clmpcc** driver first appeared in NetBSD 1.4 and is currently under development.

BUGS

The hardware flow control features of the chip are not yet fully supported.

NAME

clnp — Connectionless-Mode Network Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netiso/iso.h>
#include <netiso/clnp.h>

int
socket(AF_ISO, SOCK_RAW, 0);
```

DESCRIPTION

CLNP is the connectionless-mode network protocol used by the connectionless-mode network service. This protocol is specified in ISO 8473. It may be accessed through a "raw socket" for debugging purposes only. CLNP sockets are connectionless, and are normally used with the `sendto`(2) and `recvfrom`(2) system calls, though the `connect`(2) call may also be used to fix the destination for future packets (in which case the `read`(2) or `recv`(2) and `write`(2) or `send`(2) system calls may be used).

Outgoing packets automatically have a CLNP header prepended to them. Incoming packets received by the user contain the full CLNP header. The following `setsockopt`(2) options apply to CLNP:

`CLNPOPT_FLAGS` Sets the flags which are passed to clnp when sending a datagram. Valid flags are:

`CLNP_NO_SEG`	Do not allow segmentation
`CLNP_NO_ER`	Suppress ER pdus
`CLNP_NO_CKSUM`	Do not generate the CLNP checksum

`CLNPOPT_OPTS` Sets CLNP options. The options must be formatted exactly as specified by ISO 8473, section 7.5 "Options Part". Once an option has been set, it will be sent on all packets until a different option is set.

CONGESTION EXPERIENCE BIT

Whenever a packet is transmitted, the globally unique quality of service option is added to the packet. The sequencing preferred bit and the low transit delay bit are set in this option.

If a packet is forwarded containing the globally unique quality of service option, and the interface through which the packet will be transmitted has a queue length greater than *congest_threshold*, then the congestion experienced bit is set in the quality of service option.

The threshold value stored in *congest_threshold* may be tuned.

When a packet is received with the globally unique quality of service option present, and the congestion experienced bit is set, then the transport congestion control function is called.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

`[EISCONN]` When trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;

`[ENOTCONN]` When trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;

`[ENOBUFS]` When the system runs out of memory for an internal data structure;

[EADDRNOTAVAIL] When an attempt is made to create a socket with a network address for which no network interface exists;

[EHOSTUNREACH] When trying to send a datagram, but no route to the destination address exists.

[EINVAL] When specifying unsupported options.

SEE ALSO

recv(2), send(2), intro(4), iso(4)

BUGS

Packets are sent with the type code of 0x1d (technically an invalid packet type) for lack of a better way to identify raw CLNP packets.

No more than MLEN bytes of options can be specified.

NAME

clock — Clock driver for Motorola MVME68K Single Board Computers

SYNOPSIS

```
clock0 at pcc? ipl 5
clock0 at pcctwo? ipl 5
```

DESCRIPTION

The **clock** driver uses the counter/timer facilities of the MVME147's **pcc** driver or the MVME167/MVME177's **pcctwo** driver to provide two independent hardware clocks for use by the NetBSD kernel.

Both clock interrupts are set to 100Hz, with one used to provide the master kernel stats clock tick. The other is used for profiling purposes.

SEE ALSO

intro(4), pcc(4), pcctwo(4)

NAME

clock, **oclock** — Clock driver for Sun SPARC computers

SYNOPSIS

```
clock0 at mainbus0                # sun4c
clock0 at obio0                   # sun4m
clock0 at obio0 addr 0xf2000000   # sun4/300

ooclock0 at obio0 addr 0xf3000000 # sun4/200
ooclock0 at obio0 addr 0x03000000 # sun4/100
```

DESCRIPTION

The **clock** device contains common timer, clock and eeprom routines for the NetBSD kernel.

All system use the timer interrupt to drive the hard clock. The second interrupt is used to drive statistics clock, except sun4/100 and sun4/200 machines, which don't have a spare timer device.

HARDWARE

The **clock** driver provides support for the following chips:

Dallas DS1287A
Intersil 7170
Mostek Mk48T02
Mostek Mk48T08

SEE ALSO

sparc/timer(4)

HISTORY

The **clock** appeared in NetBSD 1.0.

NAME

clockctl — Clock subsystem user control

SYNOPSIS

```
pseudo-device clockctl
```

DESCRIPTION

The **clockctl** interface brings clock control to non-root users. Any user with write access to `/dev/clockctl` will be able to perform operations such as `settimeofday`(2), `clock_settime`(2), `adjtime`(2), or `ntp_adjtime`(2), which are normally restricted to the super-user. Using the **clockctl** pseudo-device, it is possible to run daemons such as `ntpd`(8) as non-privileged users, thus reducing the security exposure if a compromise is found in such a daemon.

The **clockctl** pseudo-device driver provides an `ioctl`(2) call for each privileged clock-related system call. The system call stubs in C library will use the `ioctl`(2) on `/dev/clockctl` if the special file is present and accessible, or will revert to the plain super-user-restricted system call if the special file is not accessible.

The following `ioctl`(2) calls are defined in ⟨`sys/clockctl.h`⟩:

`CLOCKCTL_SETTIMEOFDAY`
: This will run the `settimeofday`(2) system call. Argument should be a pointer to a *struct clockctl_settimeofday_args*:

```
struct clockctl_settimeofday_args {
    struct timeval tv;
    struct timezone tzp;
};
```

`CLOCKCTL_CLOCK_SETTIME`
: This will run the `clock_settime`(2) system call. Argument should be a pointer to a *struct clockctl_clock_settime_args*:

```
struct clockctl_clock_settime_args {
    clockid_t clock_id;
    struct timespec tp;
};
```

`CLOCKCTL_ADJTIME`
: This will run the `adjtime`(2) system call. Argument should be a pointer to a *struct clockctl_adjtime_args*:

```
struct clockctl_adjtime_args {
    struct timeval delta;
    struct timeval olddelta;
};
```

`CLOCKCTL_NTP_ADJTIME`
: This will run the `ntp_adjtime`(2) system call. Argument should be a pointer to a *struct clockctl_ntp_adjtime_args*:

```
struct clockctl_ntp_adjtime_args {
    struct timex tp;
};
```

SEE ALSO

`adjtime`(2), `clock_settime`(2), `ioctl`(2), `settimeofday`(2)

HISTORY

`clockctl` appeared in NetBSD 1.6.

NAME

cltp — ISO Connectionless Transport Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netiso/iso.h>

int
socket(AF_ISO, SOCK_DGRAM, 0);
```

DESCRIPTION

CLTP is a simple, unreliable datagram protocol which is accessed via the SOCK_DGRAM abstraction for the ISO protocol family. CLTP sockets are connectionless, and are normally used with the sendto(2) and recvfrom(2) calls, though the connect(2) call may also be used to fix the destination for future packets (in which case the recv(2) or read(2) and send(2) or write(2) system calls may be used).

CLTP address formats are identical to those used by TP. In particular CLTP provides a service selector in addition to the normal ISO NSAP. Note that the CLTP selector space is separate from the TP selector space (i.e. a CLTP selector may not be "connected" to a TP selector).

Options at the CLNP network level may be used with CLTP; see clnp(4).

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
[ENOTCONN]	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
[ENOBUFS]	when the system runs out of memory for an internal data structure;
[EADDRINUSE]	when an attempt is made to create a socket with a selector which has already been allocated;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

getsockopt(2), recv(2), send(2), socket(2), clnp(4), intro(4), iso(4)

NAME

cmdide — CMD Technology and Silicon Image IDE disk controllers driver

SYNOPSIS

```
cmdide* at pci? dev ? function ? flags 0x0000
options PCIIDE_CMD064x_DISABLE
options PCIIDE_CMD0646U_ENABLEUDMA
```

DESCRIPTION

The **cmdide** driver supports the CMD Technology PCI0640, PCI0643, PCI0646, PCI0648, PCI0649, and Silicon Image 0680 IDE controllers, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **cmdide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

There's no way to reliably know if a PCI064x controller is enabled or not. If the driver finds a PCI064x, it will assume it is enabled unless the PCIIDE_CMD064x_DISABLE option is specified in the kernel config file. This will be a problem only if the controller has been disabled in the BIOS and another controller has been installed and uses the ISA legacy I/O ports and interrupts.

The PCI0646U controller is known to be buggy with Ultra-DMA transfers, so Ultra-DMA is disabled by default for this controller. To enable Ultra-DMA, use the PCIIDE_CMD0646U_ENABLEUDMA option. Ultra-DMA can eventually be disabled on a per-drive basis with config flags, see wd(4).

The timings used for the PIO and DMA modes for controllers listed above are for a PCI bus running at 30 or 33 MHz. This driver may not work properly on overclocked systems.

NAME

cmos — Read/write access to IBM PC/AT CMOS RAM

SYNOPSIS

```
pseudo-device cmos
```

DESCRIPTION

Use **cmos** to read the real-time clock and ISA configuration data from an ISA-compatible CMOS RAM, and to write the ISA configuration data.

A program reads between 0 and 48 bytes from the CMOS RAM, starting at byte 0 of the RAM, using a single call to `read`(2). Likewise, a program writes between 0 and 48 bytes to the CMOS RAM, starting at byte 0 of the RAM, using a single call to `write`(2).

cmos does not allow programs to overwrite the real-time clock data (bytes 0 through 9), the status registers (10 through 13), the diagnostic status or CMOS shutdown status (bytes 14 and 15), or the CMOS checksum (bytes 46 and 47). Writes to those bytes are ignored.

On writes, **cmos** recomputes the CMOS checksum and writes it to the CMOS RAM.

EXAMPLES

Display entire contents of CMOS RAM:

```
# dd if=/dev/cmos bs=48 count=1 | od -t x1
0000000   37  00  09  00  22  00  06  13  04  80  26  02  50  80  00  00
0000020   00  51  f0  00  01  80  02  00  fc  0f  2f  00  00  00  00  00
0000040   00  80  81  f0  ff  00  00  00  00  00  00  00  00  00  05  ee
0000060
```

Change boot order on Soekris net4521 to PXE ROM, Primary HDD, Secondary HDD:

```
# dd if=/dev/cmos of=/tmp/cmos0 bs=48 count=1
1+0 records in
1+0 records out
48 bytes transferred in 0.001 secs (48000 bytes/sec)
# cp /tmp/cmos0 /tmp/cmos
# printf '\xf0\x80\x81\xff' | dd bs=1 seek=33 conv=notrunc of=/tmp/cmos
4+0 records in
4+0 records out
4 bytes transferred in 0.001 secs (4000 bytes/sec)
# dd if=/tmp/cmos of=/dev/cmos
0+1 records in
0+1 records out
48 bytes transferred in 0.001 secs (48000 bytes/sec)
```

ERRORS

A program can read or write no more than 48 bytes to **cmos**. `read`(2) / `write`(2) will return `EINVAL` if more than 48 bytes are read / written at once.

AUTHORS

The original **cmos** driver was written by Takahiro Kambe ⟨taca@back-street.net⟩.
David Young ⟨dyoung@NetBSD.org⟩ modified the original and added it to NetBSD.

NAME

cmpci — C-Media CMI8x38 audio device driver

SYNOPSIS

```
cmpci* at pci? dev ? function ?
audio* at audiobus?
mpu*   at cmpci?
opl*   at cmpci? flags 1
```

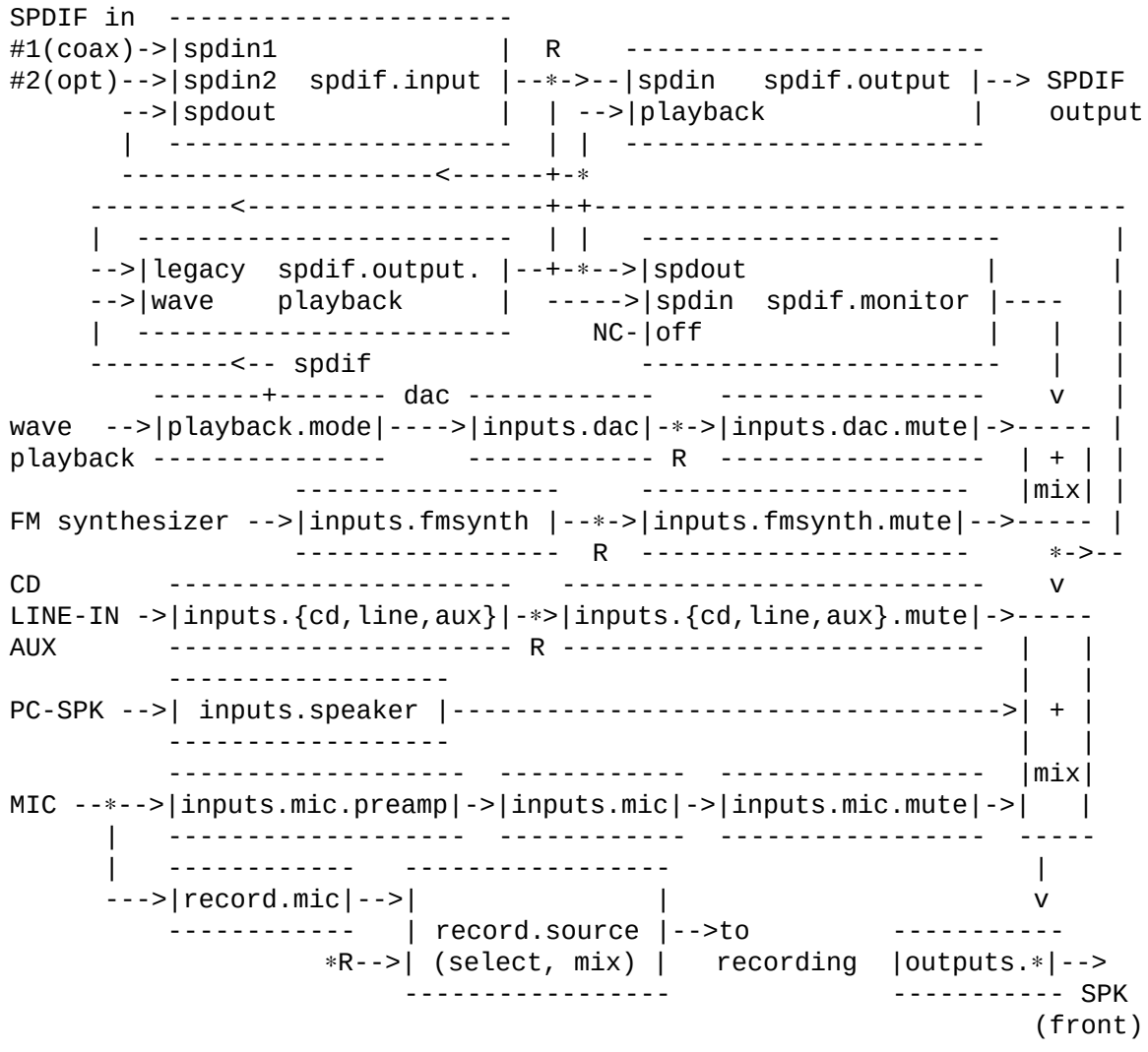
DESCRIPTION

The **cmpci** device driver supports C-Media CMI8x38 based sound cards.

The device has SPDIF input/output interfaces, 16bit CODEC with analog mixer, OPL3 FM Synthesizer, and MPU401 compatible MIDI I/O port interface.

MIXER DEVICE

The mixer device of **cmpci** driver can be accessed via `mixerctl`(1) command. The complex structure is analyzed as follows.



Note the 2nd SPDIF input exists only on CMI8738/PCI-6ch versions.

MIXER EXAMPLES

Here are examples about wave playback and SPDIF input/output ports.

Playback to speaker, SPDIF input to SPDIF output

```
mixerctl -w playback.mode=dac spdif.output=spdin spdif.monitor=off
```

Playback to SPDIF output, SPDIF input to speaker

```
mixerctl -w playback.mode=spdif spdif.output=playback
spdif.output.playback=wave spdif.monitor=spdin
```

SPDIF input to both SPDIF output and speaker

```
mixerctl -w spdif.output=spdin spdif.monitor=spdin
```

Playback to both SPDIF output and speaker

```
mixerctl -w playback.mode=spdif spdif.output=playback
spdif.output.playback=wave spdif.monitor=spdout
```

Mix playback and SPDIF input to speaker

```
mixerctl -w playback.mode=dac spdif.monitor=spdin
```

SEE ALSO

`mixerctl`(1), `audio`(4), `midi`(4), `mpu`(4), `opl`(4), `pci`(4)

HISTORY

The **`cmpci`** device driver appeared in NetBSD 1.5.

BUGS

4ch/6ch playback is not yet available. Joystick port is not supported.

`spdif.output.playback=legacy` does not seem to work properly.

NAME

cms — Creative Music System device driver

SYNOPSIS

```
cms0  at isa? port 0x220
midi* at cms?
```

DESCRIPTION

The **cms** driver provides support for the Creative Music System (C/MS). These cards were developed by Creative Labs, the same people who designed the SoundBlaster cards. Chips were available for the SoundBlaster to make them compatible with CMS.

The CMS cards are only capable of playing basic notes and noises, making them suitable for playing midi, but not much else. The output is stereo, but the **cms** driver doesn't support stereo control. The cards have external volume control, line-output and speaker.

The base I/O port address is usually jumper-selected to 0x220. Valid jumper settings are for 0x210, 0x220, 0x230, 0x240, 0x250 and 0x260. There are no interrupt settings.

SEE ALSO

isa(4), midi(4)

HISTORY

The **cms** device driver appeared in NetBSD 1.5.

NAME

cnw — Netwave AirSurfer wireless network driver

SYNOPSIS

cnw* **at pcmcia? function ?**

DESCRIPTION

The **cnw** interface provides access to a theoretical 1 Mb/s wireless Ethernet network based on the Netwave AirSurfer Wireless LAN (formerly known as the Xircom Netwave Wireless LAN).

Note that the driver does not support newer devices such as the Netwave AirSurfer “Plus”, or the BayStack 650/660. These devices are supported by the awi(4) driver.

Netwave devices are not compatible with IEEE 802.11 wireless networks. Also note that there are Netwave devices with different wireless frequency, depending on the radio band plan in each country.

The card uses 36K of I/O memory mapped to the card. You may need to increase memory space available to the PCMCIA controller. See pcmcia(4) for details.

In use, the cards appear to achieve up to a 420Kb/s transfer rate, though a transfer rate between 250Kb/s and 350Kb/s is typical.

The card operates in the 2.4GHz frequency range and is subject to interference from microwaves, IEEE 802.11 wireless network devices, as well as earth. For example, it seems that IEEE 802.11 channel 14 conflicts with Netwave (US frequency). They interfere with each other if they are both operated in the same geographic region, causing weird packet loss. You may be able to avoid the interference with IEEE 802.11 devices, by changing the IEEE 802.11 channel.

HARDWARE

Cards supported by the **cnw** driver include:

Xircom CreditCard Netwave

NetWave AirSurfer

DIAGNOSTICS

cnw0: can’t map memory Indicates that the driver was not able to allocate enough PCMCIA bus address space into which to map the device. See pcmcia(4) and increase memory available to the PCMCIA controller.

SEE ALSO

arp(4), awi(4), inet(4), intro(4), pcmcia(4), cnwctl(8)

NAME

com — serial communications interface

SYNOPSIS

```
com0 at isa? port "IO_COM1" irq 4
com1 at isa? port "IO_COM2" irq 3
com* at acpi?
com* at cardbus?
com* at isapnp?
com* at mca? slot ?
com* at mhzc?
com* at ofisa?
com* at pcmcia?
com* at pcmcom?
com* at pnpbios? index ?
com* at puc? port ?
com* at xirc?
options COM_HAYESP
options RND_COM
```

Arm32

```
com0 at mainbus? base 0x00210fe0
com1 at mainbus? base 0x00210be0
```

HP 9000/300 and 400 Series

```
com* at dio? scode ?
com* at frodo? offset ?
```

IBM PowerPC 4xx

```
com* at opb?
```

SPARC

```
com* at ebus?
com* at obio0
```

DESCRIPTION

The **com** driver provides support for NS8250-, NS16450-, and NS16550-based EIA RS-232C (CCITT V.28) communications interfaces. The NS8250 and NS16450 have single character buffers, and the NS16550 has a 16 character buffer.

Input and output for each line may set to one of following baud rates; 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, or 115200, or any other baud rate which is a factor of 115200.

The ttyXX devices are traditional dial-in devices; the dtyXX devices are used for dial-out. (See `tty`(4).)

The *COM_HAYESP* kernel option adds support for the Hayes ESP serial board.

With options RND_COM enabled, the **com** driver can be used to collect entropy for the `rnd`(4) entropy pool. The entropy is generated from interrupt randomness.

Arm32 specific

If “flags 1” is specified, the **com** driver will not set the MCR_IENABLE bit on the UART. This is mainly for use on AST multiport boards, where the MCR_IENABLE bit is used to control whether or not the devices use a shared interrupt.

FILES

```
/dev/dty00
/dev/dty01
/dev/dty02
/dev/tty00
/dev/tty01
/dev/tty02
```

DIAGNOSTICS

com%d: %d silo overflows The input “silo” has overflowed and incoming data has been lost.

com%d: weird interrupt: iir=%x The device has generated an unexpected interrupt with the code listed.

SEE ALSO

acpi(4), ast(4), cardbus(4), isa(4), isapnp(4), mca(4), mhzc(4), ofisa(4), pcmcia(4), pcmcom(4), pnpbios(4), puc(4), rtfps(4), tty(4), xirc(4)

HISTORY

The **com** driver was originally derived from the HP9000/300 **dca** driver.

BUGS

Data loss is possible on busy systems with unbuffered UARTs at high speed.

The name of this driver and the constants which define the locations of the various serial ports are holdovers from DOS.

NAME

cons − keyboard and console display interface

DESCRIPTION

The keyboard and various possible displays combine to provide a terminal-like interface to the system. Internally, these are separate devices which software combines to emulate a normal terminal. See the appropriate manual pages for information about each display and the keyboard.

The keyboard adapter also supports the speaker, which is activated when the ASCII character **bel** (**^G**) is sent to the display with software. For additional information on speaker control, see *speaker*(4).

Console Device Control

The display devices, */dev/ttyaed*, */dev/ttyap16*, */dev/ttyap8c*, */dev/ttyapa8*, */dev/ttyega*, */dev/ttymono*, */dev/ttympel*, */dev/ttyvga*, and */dev/tty8514* are all minor devices under */dev/console*, and are all capable of displaying console output. Unique to this system is the fact that you may have one or more of these displays on your workstation at a time and any one can act as a console. With only one keyboard and system mouse, the console driver multiplexes these input devices to the many displays. All of the displays may have simultaneous logins and the user can ''hot key'' between each display. At first, this ''input focus'' is on the first device in the above sequence to be found at initialization time. The input focus can be manually switched to the next available display by pressing the default ''hot key'' <Alt><Scroll Lock>. When the input focus is on a display, all keyboard and mouse data are sent to the process(es) that read from that display.

If no other console tty device is open, and only the default input emulator is used (see *kbdemul*(4)), the input focus is set to */dev/console*. In this case, <Alt><Scroll Lock> only switches which display gets console output. In the case where one or more tty devices are open, or the default input emulator changes, */dev/console* gets no input. It tries to send output to the currently focused device. A user can redirect these console messages to any tty devices with the TIOCCONS ioctl.

To support the many displays and the multiplexing between them, an emulator package was developed to work with the console driver. This package allows different types of emulation on input and output to be written independently of device.

The display devices */dev/aed*, */dev/apa16*, */dev/apa8c*, */dev/apa8*, */dev/ega*, */dev/mono*, */dev/mpel*, */dev/vga*, and */dev/ibm8514* are also minor devices to */dev/console*. They are typically used by window managers and other graphic applications. When the focus is pointed to one of these display devices, the console messages are put in a circular buffer (see *bufemul*(4)) unless redirected with the TIOCCONS ioctl. The buffer is flushed to the screen upon closing the display device.

The following are generic console *ioctls* defined in *screen_cousf.u*:

CON_SELECT_SCREEN
: Output focus is set to display number (arg > 0) or to next display in list (arg < 0). Previous display number is returned.

CON_GET_SCREEN
: Just returns the current output focus display number.

EIGETD
: Gets the number of the current input emulator for this display.

EOGETD
: Gets the number of the current output emulator for this display.

EISETD
: Sets the input emulator and returns the previous for this display.

EOSETD
: Sets the output emulator and returns the previous for this display.

CON_INIT_SCREEN
: Initializes the specified display (arg 0) or this display (arg < 0).

CON_GET_FOCUS_CODE
: Gets the current keyboard code for setting the console focus (affects xemul only).

CON_SET_FOCUS_CODE
: Sets the current keyboard code for setting the console focus (affects xemul only), and return the previous code.

All of the above commands take integer arguments.

The following are generic console *ioctls* defined in *consio.h*:

SCRIOCGETF Gets the screen control flags for the given display number.

SCRIOCSETC Sets the screen control flags for the given display number.

SCRIOCGETF and SCRIOCSETC use the following structure:

```
struct screen_control {
        int   device; /* which screen/display to control */
        int   switches; /* Flags for this screen */
};
```

CONSDEV_PRESENT
: Display is present on this system. This bit cannot be changed by SCRIOSETC.

CONSDEV_KERNEL
: Display is available to the kernel.

CONSDEV_USER
: Display is available to the user.

CONSDEV_INIT
: Display has been initialized for output.

CONSDEV_TTY
: Diplay has been initialuzed for output. This bit cannot be changed by SCRIOSETC.

CONSDEV_GRA
: Graphics display has been opened directly by minor device number. This bit cannot be changed by SCRIOSETC.

CONSDEV_NOINPUT
: Prevents the "round-robin" console focus-switching from finding this display. This flag is cleared when *buf_emul* is closed.

SCRSETNIP
: Sets the no-input bit in the screen control flags for the display's current file description.

SCRCLRNIP
: Clears the no-input bit in the screen control flags for the display's current file description.

The following *ioctl* is defined in *bufemul.h*:

BUFDISPINFO *Arg* returns the following information about the display:

BUF_IS_ATR(arg)
: True when the CPU is an IBM 6152 Academic System.

BUF_IS_RTPC(arg)
: True when the CPU is an IBM RT PC.

BUF_GET_PCCODE(x)
: Get PC-Code version/type byte (IBM 6152 only).

BUF_GET_VGA(arg) Get the type of display connected to the VGA. 0=none, 1=color, 2=gray. Valid only for the IBM 6152 Academic System.

BUF_GET_8514(arg) Get the type of display connected to the IBM 8514/A. 0=none, 1=color, 2=gray. Valid only for the IBM 6152 Academic System.

BUF_GET_EGA(arg) Returns the value of the switches on the EGA display. Valid only for the IBM RT PC with an EGA card installed.

All of the above *ioctl* system calls are device-independent controls for dealing with the emulators.

Each emulator has its own set of *ioctls* for its own emulation purposes. These other *iotls* are used in window-manager emulators for operations such as passing/positioning the mouse locator for/on the display. See the man page for any particular emulator for more information.

NOTE

On the IBM RT PC, the kernel flashes ‘‘98’’ on the LEDs if it cannot find any configured display during initialization, and then proceeds.

DIAGNOSTICS

None.

FILES

For the IBM RT PC:
/dev/console
/dev/aed
/dev/apa16
/dev/apa8c
/dev/apa8
/dev/ega
/dev/mono
/dev/mpel

For the IBM 6152 Academic System:
/dev/vga
/dev/ibm8514

SEE ALSO

bufemul(4), bus(4), ibm5081(4), ibm5151(4), ibm6153(4), ibm6154(4), ibm6155(4), ibm8514(4), ibmaed(4), ibmemul(4), kbdemul(4), speaker(4), stdemul(4), tty(4), vga(4), xemul(4), setscreen(8)
‘‘IBM/4.3 Console Emulators’’, in Volume II, Supplementary Documents

NAME

cons — HP300 console interface

DESCRIPTION

This release supports a “virtual” console device used for *kernel printf* messages and accessed in user mode via `/dev/console`. It is virtual in the sense that it is attached to a hardware interface at boot time. Currently the choices are limited to: a bit-mapped display acting as an *internal terminal emulator* “ITE”, the builtin serial interface `com`(4), or a `null`(4) console in that order.

FILES

`/dev/console`

SEE ALSO

`com`(4), `tty`(4), `reboot`(8)

BUGS

You should be able to specify potential console devices at `config`(1) time.

NAME

cons — VAX-11 console interface

DESCRIPTION

The console is available to the processor through the console registers. It acts like a normal terminal, except that when the local functions are not disabled, **^P** (control-P) puts the console in local console mode (where the prompt is '>>>'). The operation of the console in this mode varies slightly per-processor.

VAX-11/780 or 785

On either the VAX-11/780 or 785 the following commands may be used after placing the console in local mode with **^P**.

c
continue Re-enter conversational mode if the processor was halted.

h
halt Halt the CPU. On an 11/780 or 785 the processor is not stopped by entering local console mode.

set t p (set terminal program) Re-enter conversational mode if the processor is still running.

P (proceed) Get out of ODT mode.

<break> If you hit the break key on the console, then the console LSI-11 will go into ODT (console debugger mode).

VAX-11/750 or 11/730

On an 11/750 or an 11/730 the processor is halted whenever the console is not in conversational mode.

C Return to conversational mode.

ret Return from remote diagnosis mode to local console mode.

^D (11/750 only) When in console mode on an 11/750 which has a remote diagnosis module, a **^D** will put you in remote diagnosis mode, where the prompt will be RDM>.

VAX-8600 or 8650

The VAX-8600 (8650) console normally works in the same way as the 11/750, except that there are many additional modes and commands.

c
continue Return to conversational mode.

halt Halt the processor if HEX debug enabled.

p Halt the processor if in normal mode.

With the above proviso's the console works like any other UNIX terminal.

FILES

/dev/console

SEE ALSO

tty(4), reboot(8)

VAX Hardware Handbook.

HISTORY

The **cons** interface appeared in 4.0BSD.

NAME

console — amiga console interface

DESCRIPTION

This release supports a “virtual” console device used for *kernel printf* messages and accessed in user mode via `/dev/console`. It is virtual in the sense that it is attached to a hardware interface at boot time. Currently the choices are limited to: a bit-mapped display acting as an *internal terminal emulator* “ITE”, the builtin serial interface `ser`(4), or a `null`(4) console in that order.

FILES

`/dev/console`

SEE ALSO

`tty`(4), `reboot`(8)

HISTORY

The **console** device is currently under development.

BUGS

You should be able to specify potential console devices at `config`(1) time.

NAME

console — i386 console interface

SYNOPSIS

```
options CONSDEVNAME=string
options CONADDR=integer
options CONSPEED=integer
options CONS_OVERRIDE
options CONMODE=integer
```

DESCRIPTION

The "console" device is used for *kernel printf* messages and accesses to the `/dev/console` character special device in user mode. It is attached to a hardware interface at boot time controlled by options in the kernel configuration file, or information passed by the boot loader.

Bootblocks from NetBSD 1.4 or newer select their console device from a compiled-in list, and then pass their choice of console device and console parameters to the kernel.

As of NetBSD 1.5, the **consdev** bootblock command allows changing the console device on-the-fly.

The kernel will use the same console device as the bootblock; no special kernel configuration is required.

To override the bootblock's choice of console, or to use a serial kernel console with older bootblocks, you must specify kernel config-file options to override the information passed by the bootblock. The current option choices are:

- the standard PC keyboard and display
  (with either the "pc" or the `wscons`(4) driver)

- standard PC serial ports
  (with `com`(4) driver)

The available *kernel configuration* options are:

options CONSDEVNAME=string
specifies the name of the console device. Valid values are "pc" for the pc keyboard / display (default) and "com" for a serial port.

options CONADDR=integer
sets the base address for the serial console port (default: 0x3f8).

options CONSPEED=integer
sets the baudrate for the serial console (default: 9600).

options CONS_OVERRIDE
causes console information passed by the bootloader to be ignored and the settings specified by the three options above (or the defaults) to be used. Default behaviour is to use the settings from the bootloader if present, and to use option / default values only if no information was passed.

options CONMODE=integer
allows to specify terminal control flags. The argument is a "cflag" value, see `termios`(4) for details. Default is `(CREAD | CS8 | HUPCL)` (8N1). This option takes always effect, because mode settings are not passed by the bootloader.

FILES

`/dev/console`

`/sys/arch/i386/conf/CONFIG`

EXAMPLES

```
options CONSDEVNAME="\"com\"",CONADDR=0x2f8,CONSPEED=57600
```

SEE ALSO

`config`(1), `tty`(4), `boot`(8), `boot_console`(8)

BUGS

The console device is chosen early in system startup regardless if the specified driver / device is present in the system configuration file. If the driver asked for by the bootloader or "**`options CONSDEVNAME`**" is not configured into the system, a panic is caused. Because there is no console device, no explaining message will be printed. If the driver is present, but the specific device instance not, kernel printf will work, but `/dev/console` becomes a dummy.

NAME

coretemp — Intel Core on-die digital thermal sensor

SYNOPSIS

```
options INTEL_CORETEMP
```

DESCRIPTION

The **coretemp** driver provides support for the on-die digital thermal sensor present on Intel Core and newer CPUs.

The **coretemp** driver reports each core's temperature through the envsys(4) API. The driver has only 1 temperature sensor:

Sensor	Units	Typical Use
sensor0	uK	cpuN temperature

EVENTS

The **coretemp** driver is able to send a *critical* event to the */etc/powerd/scripts/sensor_temperature* powerd(8) script (if running) when the temperature has reached a critical state.

SEE ALSO

envstat(4), powerd(8)

HISTORY

The **coretemp** driver first appeared in FreeBSD 7.0. And then was ported to NetBSD 5.0.

AUTHORS

The **coretemp** driver was written by Rui Paulo ⟨rpaulo@FreeBSD.org⟩ as part of a Google Summer of Code project. It was adapted to NetBSD by Juan Romero Pardines.

NAME

cosc — MCS Connect32 SCSI II Card device interface

SYNOPSIS

```
cosc0 at podulebus?
```

DESCRIPTION

The **cosc** interface provides access to MCS Connect32 interfaces.

SEE ALSO

asc(4), csc(4), oak(4), ptsc(4)

NAME

cpc — IBM CPC700 bridge support

SYNOPSIS

```
cpc0 at mainbus0
com0 at cpc0 addr 0xff600300
com1 at cpc0 addr 0xff600400
pci0 at cpc0
```

DESCRIPTION

The **cpc** provides support for the IBM CPC700 host bridge aimed at Power PC.

SEE ALSO

com(4), mainbus(4), pci(4)

HISTORY

The **cpc** driver appeared in NetBSD 2.0.

NAME

cpi — parallel printer driver for Creative Systems Inc. Hurdler CPI Nubus card

SYNOPSIS

cpi* **at nubus?**

DESCRIPTION

The **cpi** interface provides access to parallel printer ports.

HARDWARE

The **cpi** interface supports the Creative Systems Inc. Hurdler CPI Nubus card, which is based on a Zilog Z8536 CIO.

The parallel port on the Hurdler CPI card is wired as follows:

Signal		SubD pin	Z8536 pin	Z8536 signal	
/STROBE		Strobe	1	22	PC3
D0		2	33	PA0	
D1		3	32	PA1	
D2		4	31	PA2	
D3		5	30	PA3	
D4		6	29	PA4	
D5		7	28	PA5	
D6		8	27	PA6	
D7		9	26	PA7	
/ACK		10	21 + 11	PC2 + PB3	
BUSY		11	19 + 14	PC0 + PB6	
PE	Paper Error	12	20	PC1	
SEL	Select	13	13	PB5	
/AUTOFD		Auto Feed	14	12	PB4
/FAULT		15	9	PB1	
/RESET		16	8	PB0	
/SELIN	Select In	17	10	PB2	

The Z8536 INT line (pin 24) is wired to PB7 (pin 15).

SEE ALSO

autoconf(4), lpt(4), nubus(4), printcap(5)

IEEE Standard 1284-1994

HISTORY

cpi first appeared in NetBSD 5.0.

AUTHORS

The **cpi** driver was written by Hauke Fath ⟨hauke@NetBSD.org⟩.

CAVEATS

The Hurdler CPI Nubus card does not use a TTL buffer to drive the parallel interface. Instead, the card's Z8536 CIO drives the printer port directly. Printers terminating the parallel interface with less than 2 kOhms may cause permanent damage to the Z8536 CIO.

NAME

cpu — Device driver for CPU-specific features

SYNOPSIS

```
cpu0 at root flags 0
options CPU_ARM2
options CPU_ARM250
options CPU_ARM3
```

DESCRIPTION

The **cpu** driver provides a convenient hook for identifying the system's CPU and setting various parameters relating to it.

Setting the bottom bit of the flags will cause the cache on the ARM3 to be disabled, otherwise it's enabled.

The following options are available relating to the **cpu** driver:

options CPU_ARM2
Enables support for ARM2 CPUs.

options CPU_ARM250
Enables support for ARM2as CPUs (as found in the ARM250).

options CPU_ARM3
Enables support for ARM3 CPUs.

NAME

cpu — HP PA-RISC CPU

SYNOPSIS

```
cpu*        at mainbus0 irq 31
```

DESCRIPTION

The following table lists the PA-RISC CPU types and their characteristics, such as TLB, maximum cache sizes and HP 9000/700 machines they were used in (see also `intro`(4) for the reverse list).

CPU	PA	Clock (max) MHz	Caches (max) KB	TLB	BTLB	Models
7000	1.1a	66	256 L1I 256 L1D	96I 96D	4 I 4 D	705,710,720 730,750
7100	1.1b	100	1024 L1I 2048 L1D	120	16	715/33/50/75 725/50/75 {735,755}/100 742i, 745i, 747i
7150	1.1b	125	1024 L1I 2048 L1D	120	16	{735,755}/125
7100LC	1.1c	100	1 L1I 1024 L2I 1024 L2D	64	8	712/60/80/100 715/64/80/100 715/100XC 725/64/100 743i, 748i SAIC
7200	1.1d	140	2 L1 1024 L2I 1024 L2D	120	16	C100,C110 J200,J210
7300LC	1.1e	180	64 L1I 64 L1D 8192 L2	96	8	A180,A180C B132,B160,B180 C132L,C160L 744, 745, 748 RDI PrecisioBook

FLOATING-POINT COPROCESSOR

The following table summarizes available floating-point coprocessor models for the 32-bit PA-RISC processors.

FPU	Model
Indigo	
Sterling I MIU (TYCO)	
Sterling I MIU (ROC w/Weitek)	
FPC (w/Weitek)	
FPC (w/Bit)	
Timex-II	
Rolex	725/50, 745i
HARP-I	
Tornado	J2x0,C1x0
PA-50 (Hitachi)	
PCXL	712/60/80/100

SUPERSCALAR EXECUTION

The following table summarizes the superscalar execution capabilities of 32-bit PA-RISC processors.

CPU	Units	Bundles
7100	1 integer ALU	load-store/fp
	1 FP	int/fp
		branch/*
7100LC	2 integer ALU	load-store/int
	1 FP	load-store/fp
		int/fp
		branch/*
7200	2 integer ALU	load-store/int
	1 FP	load-store/fp
		int/int
		int/fp
		branch/*
7300LC	2 integer ALU	load-store/int
	1 FP	load-store/fp
		int/fp
		branch/*

In conclusion, all of the above CPUs are dual-issue, or 2-way superscalar, with the exception that on CPUs with two integer ALUs only one of these units is capable of doing shift, load/store, and test operations. Additionally, there are several kinds of restrictions placed upon the superscalar execution:

For the purpose of showing which instructions are allowed to proceed together through the pipeline, they are divided into classes:

Class	Description
flop	floating point operation
ldst	loads and stores
flex	integer ALU
mm	shifts, extracts and deposits
nul	might nullify successor
bv	BV, BE
br	other branches
fsys	FTEST and FP status/exception
sys	system control instructions

For CPUs with two integer ALUs (7100LC, 7200, 7300LC), the following table lists the instructions which are allowed to be executed concurrently:

First	Second instruction
flop	+ ldst/flex/mm/nul/bv/br
ldst	+ flop/flex/mm/nul/br
flex	+ flop/ldst/flex/mm/nul/br/fsys
mm	+ flop/ldst/flex/fsys
nul	+ flop
sys	never bundled

ldst + ldst is also possible under certain circumstances, which is then called "double word load/store".

The following restrictions are placed upon the superscalar execution:

- An instruction that modifies a register will not be bundled with another instruction that takes this register as operand. Exception: a flop can be bundled with an FP store of the flop's result register.
- An FP load to one word of a doubleword register will not be bundled with a flop that uses the other doubleword of this register.
- A flop will not be bundled with an FP load if both instructions have the same target register.
- An instruction that could set the carry/borrow bits will not be bundled with an instruction that uses carry/borrow bits.
- An instruction which is in the delay slot of a branch is never bundled with other instructions.
- An instruction which is at an odd word address and executed as a target of a taken branch is never bundled.
- An instruction which might nullify its successor is never bundled with this successor. Only if the successor is a flop instruction is this bundle allowed.

PERFORMANCE MONITOR COPROCESSOR

The performance monitor coprocessor is an optional, implementation-dependent coprocessor which provides a minimal common software interface to implementation-dependent performance monitor hardware.

DEBUG SPECIAL UNIT

The debug special function unit is an optional, architected SFU which provides hardware assistance for software debugging using breakpoints. The debug SFU is currently defined only for Level 0 processors.

SEE ALSO

`asp`(4), `intro`(4), `lasi`(4), `mem`(4), `wax`(4), `http://www.openpa.net/`

Hewlett-Packard, *PA-RISC 1.1 Architecture and Instruction Set Reference Manual*, May 15, 1996.

Hewlett-Packard, *PA7100LC ERS*, Public version 1.0, March 30 1999.

Hewlett-Packard Journal, *Design of the PA7200 CPU*, February 1996.

Hewlett-Packard, *PA7300LC ERS*, Version 1.0, March 18 1996.

HISTORY

The **cpu** driver was written by Michael Shalayeff ⟨mickey@openbsd.org⟩ for the HPPA port for OpenBSD 2.5. It was ported to NetBSD 1.6 by Matthew Fredette.

NAME

crime — CPU, Rendering, Interconnect and Memory Engine

SYNOPSIS

```
crime0 at mainbus0 addr 0x14000000
```

DESCRIPTION

The **crime** ASIC acts as a controller between the CPU and VICE, MACE, the graphics back-end, and main memory. **crime** can be typically found in O2 machines.

SEE ALSO

mace(4)

HISTORY

The **crime** driver first appeared in NetBSD 1.5.

BUGS

crime does not pay.

NAME

crl — VAX-8600 console RL02 interface

DESCRIPTION

This is a simple interface to the DEC RL02 disk unit which is part of the console subsystem on the VAX-8600 and 8650. Access is given to the entire RL02 disk; the pack format is the same as that of RL02 disks on other controllers. As on other VAX console media, transfers are done a word at a time using privileged registers (i.e., slowly).

All I/O is raw; the seek addresses in raw transfers should be a multiple of 512 bytes and a multiple of 512 bytes should be transferred, as in other "raw" disk interfaces. (Although the sector size is actually 256 bytes, the driver allows operations only on 512-byte boundaries.)

FILES

`/dev/crl`

SEE ALSO

`arff`(8)

HISTORY

The **crl** driver appeared in 4.3BSD.

NAME

crypto, **swcrypto** — user-mode access to hardware-accelerated cryptography

SYNOPSIS

```
hifn*   at pci? dev ? function ?
ubsec*  at pci? dev ? function ?

pseudo-device crypto
pseudo-device swcrypto

#include <sys/ioctl.h>
#include <sys/time.h>
#include <crypto/cryptodev.h>
```

DESCRIPTION

The **crypto** driver gives user-mode applications access to hardware-accelerated cryptographic transforms, as implemented by the opencrypto(9) in-kernel interface. The **swcrypto** driver is a software-only implementation of the opencrypto(9) interface, and must be included to use the interface without hardware acceleration. The /dev/crypto special device provides an ioctl(2) based interface. User-mode applications should open the special device, then issue ioctl(2) calls on the descriptor. The **crypto** device provides two distinct modes of operation: one mode for symmetric-keyed cryptographic requests, and a second mode for both asymmetric-key (public-key/private-key) requests, and for modular arithmetic (for Diffie-Hellman key exchange and other cryptographic protocols). The two modes are described separately below.

THEORY OF OPERATION

Regardless of whether symmetric-key or asymmetric-key operations are to be performed, use of the device requires a basic series of steps:

1. Open a file descriptor for the device. See open(2).
2. If any symmetric operation will be performed, create one session, with CIOCGSESSION, or multiple sessions, with CIOCNGSESSION. Most applications will require at least one symmetric session. Since cipher and MAC keys are tied to sessions, many applications will require more. Asymmetric operations do not use sessions.
3. Submit requests, synchronously with CIOCCRYPT (symmetric) or CIOCFKEY (asymmetric) or asynchronously with CIOCNCRYPTM (symmetric) or CIOCNFKEYM (asymmetric). The asynchronous interface allows multiple requests to be submitted in one call if the user so desires.
4. If the asynchronous interface is used, wait for results with select(2) or poll(2), then collect them with CIOCNCRYPTRET (a particular request) or CIOCNCRYPTRETM (multiple requests).
5. Destroy one session with CIOCFSESSION or many at once with CIOCNFSESSION.
6. Close the device with close(2).

SYMMETRIC-KEY OPERATION

The symmetric-key operation mode provides a context-based API to traditional symmetric-key encryption (or privacy) algorithms, or to keyed and unkeyed one-way hash (HMAC and MAC) algorithms. The symmetric-key mode also permits fused operation, where the hardware performs both a privacy algorithm and an integrity-check algorithm in a single pass over the data: either a fused encrypt/HMAC-generate operation, or a fused HMAC-verify/decrypt operation.

To use symmetric mode, you must first create a session specifying the algorithm(s) and key(s) to use; then issue encrypt or decrypt requests against the session.

Symmetric-key privacy algorithms

Contingent upon device drivers for installed cryptographic hardware registering with `opencrypto`(9), as providers of a given algorithm, some or all of the following symmetric-key privacy algorithms may be available:

CRYPTO_DES_CBC
CRYPTO_3DES_CBC
CRYPTO_BLF_CBC
CRYPTO_CAST_CBC
CRYPTO_SKIPJACK_CBC
CRYPTO_AES_CBC
CRYPTO_ARC4

Integrity-check operations

Contingent upon hardware support, some or all of the following keyed one-way hash algorithms may be available:

CRYPTO_RIPEMD160_HMAC
CRYPTO_MD5_KPDK
CRYPTO_SHA1_KPDK
CRYPTO_MD5_HMAC
CRYPTO_SHA1_HMAC
CRYPTO_SHA2_HMAC
CRYPTO_MD5
CRYPTO_SHA1

The *CRYPTO_MD5* and *CRYPTO_SHA1* algorithms are actually unkeyed, but should be requested as symmetric-key hash algorithms with a zero-length key.

IOCTL Request Descriptions

`CRIOGET` *`int *fd`*

This operation is deprecated and will be removed after NetBSD 5.0. It clones the fd argument to `ioctl`(4), yielding a new file descriptor for the creation of sessions. Because the device now clones on open, this operation is unnecessary.

`CIOCGSESSION` *`struct session_op *sessp`*

```
struct session_op {
    u_int32_t cipher;  /* e.g. CRYPTO_DES_CBC */
    u_int32_t mac;     /* e.g. CRYPTO_MD5_HMAC */

    u_int32_t keylen;  /* cipher key */
    void * key;
    int mackeylen;     /* mac key */
    void * mackey;

    u_int32_t ses;     /* returns: ses # */
};
```

Create a new cryptographic session on a file descriptor for the device; that is, a persistent object specific to the chosen privacy algorithm, integrity algorithm, and keys specified in *`sessp`*. The special value 0 for either privacy or integrity is reserved to indicate that the indicated operation (privacy or integrity) is not desired for this session.

Multiple sessions may be bound to a single file descriptor. The session ID returned in *`sessp->ses`* is supplied as a required field in the symmetric-operation structure *`crypt_op`*

for future encryption or hashing requests.

This implementation will never return a session ID of 0 for a successful creation of a session, which is a NetBSD extension.

For non-zero symmetric-key privacy algorithms, the privacy algorithm must be specified in *sessp->cipher*, the key length in *sessp->keylen*, and the key value in the octets addressed by *sessp->key*.

For keyed one-way hash algorithms, the one-way hash must be specified in *sessp->mac*, the key length in *sessp->mackey*, and the key value in the octets addressed by *sessp->mackeylen*.

Support for a specific combination of fused privacy and integrity-check algorithms depends on whether the underlying hardware supports that combination. Not all combinations are supported by all hardware, even if the hardware supports each operation as a stand-alone non-fused operation.

CIOCNGSESSION *struct crypt_sgop *sgop*

```
struct crypt_sgop {
    size_t     count;                /* how many */
    struct session_n_op * sessions; /* where to get them */
};

struct session_n_op {
    u_int32_t cipher;         /* e.g. CRYPTO_DES_CBC */
    u_int32_t mac;            /* e.g. CRYPTO_MD5_HMAC */

    u_int32_t keylen;         /* cipher key */
    void * key;
    u_int32_t mackeylen;      /* mac key */
    void * mackey;

    u_int32_t ses;            /* returns: session # */
    int status;
};
```

Create one or more sessions. Takes a counted array of *session_n_op* structures in *sgop*. For each requested session (array element n), the session number is returned in *sgop->sessions[n].ses* and the status for that session creation in *sgop->sessions[n].status*.

CIOCCRYPT *struct crypt_op *cr_op*

```
struct crypt_op {
    u_int32_t ses;
    u_int16_t op;      /* e.g. COP_ENCRYPT */
    u_int16_t flags;
    u_int len;
    void * src, *dst;
    void * mac;        /* must be large enough for result */
    void * iv;
};
```

Request a symmetric-key (or hash) operation. The file descriptor argument to `ioctl`(4) must

have been bound to a valid session. To encrypt, set *cr_op->op* to COP_ENCRYPT. To decrypt, set *cr_op->op* to COP_DECRYPT. The field *cr_op->len* supplies the length of the input buffer; the fields *cr_op->src*, *cr_op->dst*, *cr_op->mac*, *cr_op->iv* supply the addresses of the input buffer, output buffer, one-way hash, and initialization vector, respectively.

CIOCNCRYPTM *struct crypt_mop *cr_mop*

```
struct crypt_mop {
    size_t count;             /* how many */
    struct crypt_n_op * reqs; /* where to get them */
};

struct crypt_n_op {
    u_int32_t ses;
    u_int16_t op;             /* e.g. COP_ENCRYPT */
    u_int16_t flags;
    u_int len;

    u_int32_t reqid;          /* request id */
    int status;                      /* accepted or not */

    void *opaque;             /* opaque pointer ret to user */
    u_int32_t keylen;         /* cipher key - optional */
    void * key;
    u_int32_t mackeylen;      /* mac key - optional */
    void * mackey;

    void * src, * dst;
    void * mac;
    void * iv;
};
```

This is the asynchronous version of CIOCCRYPT, which allows multiple symmetric-key (or hash) operations to be started (see CIOCRYPT above for the details for each operation).

The *cr_mop->count* field specifies the number of operations provided in the cr_mop->reqs array.

Each operation is assigned a unique request id returned in the *cr_mop->reqs[n].reqid* field.

Each operation can accept an opaque value from the user to be passed back to the user when the operation completes ((e.g. to track context for the request). The opaque field is *cr_mop->reqs[n].opaque.*

If a problem occurs with starting any of the operations then that operation's *cr_mop->reqs[n].status* field is filled with the error code. The failure of an operation does not prevent the other operations from being started.

The select(2) or poll(2) functions must be used on the device file descriptor to detect that some operation has completed; results are then retrieved with CIOCNCRYPTRETM.

The *key* and *mackey* fields of the operation structure are currently unused. They are intended for use to immediately rekey an existing session before processing a new request.

`CIOCFSESSION` *`void`*

Destroys the /dev/crypto session associated with the file-descriptor argument.

`CIOCNFSESSION` *`struct crypt_sfop *sfop;`*

```
struct crypt_sfop {
    size_t count;
    u_int32_t *sesid;
};
```

Destroys the *`sfop->count`* sessions specified by the *`sfop`* array of session identifiers.

ASYMMETRIC-KEY OPERATION

Asymmetric-key algorithms

Contingent upon hardware support, the following asymmetric (public-key/private-key; or key-exchange subroutine) operations may also be available:

Algorithm	Input parameter Count	Output parameter Count
`CRK_MOD_EXP`	3	1
`CRK_MOD_EXP_CRT`	6	1
`CRK_MOD_ADD`	3	1
`CRK_MOD_ADDINV`	2	1
`CRK_MOD_SUB`	3	1
`CRK_MOD_MULT`	3	1
`CRK_MOD_MULTINV`	2	1
`CRK_MOD`	2	1
`CRK_DSA_SIGN`	5	2
`CRK_DSA_VERIFY`	7	0
`CRK_DH_COMPUTE_KEY`	3	1

See below for discussion of the input and output parameter counts.

Asymmetric-key commands

`CIOCASSYMFEAT` *`int *feature_mask`*

Returns a bitmask of supported asymmetric-key operations. Each of the above-listed asymmetric operations is present if and only if the bit position numbered by the code for that operation is set. For example, `CRK_MOD_EXP` is available if and only if the bit `(1 << CRK_MOD_EX)` is set.

`CIOCFKEY` *`struct crypt_kop *kop`*

```
struct crypt_kop {
    u_int crk_op;               /* e.g. CRK_MOD_EXP */
    u_int crk_status;           /* return status */
    u_short crk_iparams;        /* # of input params */
    u_short crk_oparams;        /* # of output params */
    u_int crk_pad1;
    struct crparam crk_param[CRK_MAXPARAM];
};

/* Bignum parameter, in packed bytes. */
struct crparam {
    void * crp_p;
    u_int crp_nbits;
```

```
};
```

Performs an asymmetric-key operation from the list above. The specific operation is supplied in *kop->crk_op*; final status for the operation is returned in *kop->crk_status*. The number of input arguments and the number of output arguments is specified in *kop->crk_iparams* and *kop->crk_iparams*, respectively. The field *crk_param[]* must be filled in with exactly *kop->crk_iparams + kop->crk_oparams* arguments, each encoded as a *struct crparam* (address, bitlength) pair.

The semantics of these arguments are currently undocumented.

CIOCNFKEYM *struct crypt_mkop *mkop*

```
struct crypt_mkop {
    size_t count;             /* how many */
    struct crypt_n_op * reqs; /* where to get them */
};

struct crypt_n_kop {
    u_int crk_op;             /* e.g. CRK_MOD_EXP */
    u_int crk_status;         /* accepted or not */
    u_short crk_iparams;      /* # of input params */
    u_short crk_oparams;      /* # of output params */
    u_int32_t crk_reqid;      /* request id */
    struct crparam crk_param[CRK_MAXPARAM];
    void *crk_opaque;         /* opaque pointer ret to user */
};
```

This is the asynchronous version of CIOCFKEY, which starts one or more key operations. See CIOCNCRYPTM above and CIOCNCRYPTRETM below for descriptions of the *mkop>count*, *mkop>reqs*, *mkop>reqs[n].crk_reqid*, *mkop>reqs[n].crk_status*, and *mkop>reqs[n].crk_opaque* fields of the argument structure, and result retrieval.

Asynchronous status commands

When requests are submitted with the CIOCNCRYPTM or CIOCNFKEYM commands, result retrieval is asynchronous (the submit ioctls return immediately). Use the select(2) or poll(2) functions to determine when the file descriptor has completed operations ready to be retrieved.

CIOCNCRYPTRET *struct crypt_result *cres*

```
struct crypt_result {
    u_int32_t reqid;  /* request ID */
    u_int32_t status; /* 0 if successful */
    void * opaque;    /* pointer from user */
};
```

Check for the status of the request specified by *cres->reqid*. This requires a linear search through all completed requests and should be used with extreme care if the number of requests pending on this file descriptor may be large.

The *cres->status field is set as follows:*

0 The request has completed, and its results have been copied out to the original *crypt_n_op or crypt_n_kop* structure used to start the request. The copyout occurs during this ioctl, so the calling process must be the process

that started the request.

EINPROGRESS
The request has not yet completed.

EINVAL The request was not found.

Other values indicate a problem during the processing of the request.

`CIOCNCRYPTRETM` *`struct cryptret_t *cret`*

```
struct cryptret {
    size_t count;                    /* space for how many */
    struct crypt_result * results;   /* where to put them */
};
```

Retrieve a number of completed requests. This ioctl accepts a count and an array (each array element is a *`crypt_result_t`* structure as used by `CIOCNCRYPTRET` above) and fills the array with up to *`cret->count`* results of completed requests.

This ioctl fills in the *`cret->results[n].reqid field`*, so that the request which has completed may be identified by the application. Note that the results may include requests submitted both as symmetric and asymmetric operations.

SEE ALSO

`hifn`(4), `ubsec`(4), `opencrypto`(9)

HISTORY

The **`crypto`** driver is derived from a version which appeared in FreeBSD 4.8, which in turn is based on code which appeared in OpenBSD 3.2.

The "new API" for asynchronous operation with multiple basic operations per system call (the "N" ioctl variants) was contributed by Coyote Point Systems, Inc. and first appeared in NetBSD 5.0.

BUGS

Error checking and reporting is weak.

The values specified for symmetric-key key sizes to `CIOCGSESSION` must exactly match the values expected by `opencrypto`(9). The output buffer and MAC buffers supplied to `CIOCCRYPT` must follow whether privacy or integrity algorithms were specified for session: if you request a non-`NULL` algorithm, you must supply a suitably-sized buffer.

The scheme for passing arguments for asymmetric requests is Baroque.

The naming inconsistency between `CRIOGET` and the various `CIOC*` names is an unfortunate historical artifact.

NAME

cs — Cirrus Logic Crystal CS89x0 Ethernet driver

SYNOPSIS

```
cs0 at isa? port 0x300 iomem ? irq ? drq ?
cs* at ofisa?
cs* at isapnp?
cs* at pcmcia? function ?
cs0 at mainbus0
```

(PM/PPC port)

DESCRIPTION

The **cs** driver supports Ethernet interfaces based on the Cirrus Logic Crystal CS8900, 8920 and 8920M ISA bus Ethernet controllers.

SEE ALSO

ifmedia(4), intro(4), isa(4), ofisa(4), ifconfig(8)

http://www.cirrus.com/

HISTORY

The **cs** driver appeared in NetBSD 1.4.

NAME

cs80bus — support for CS80/SS80 on the IEEE488 GPIB

SYNOPSIS

cs80bus* at gpib?

DESCRIPTION

The **cs80bus** driver supports devices on the IEEE488 GPIB which communicate using the CS80/SS80 protocol. These device are primarily block devices such as tapes and disks drives commonly found on HP/Agilent equipment.

SEE ALSO

ct(4), gpib(4), mt(4), rd(4)

HISTORY

The **cs80bus** driver appeared in NetBSD 2.0.

NAME

csc — Cumana SCSI II Card device interface

SYNOPSIS

```
csc0 at podulebus?
```

DESCRIPTION

The **csc** interface provides access to Cumana SCSI II interfaces.

SEE ALSO

asc(4), cosc(4), oak(4), ptsc(4)

NAME

css — DEC IMP-11A LH/DH IMP network interface

SYNOPSIS

```
pseudo-device imp device css0 at uba0 csr 167600 flags 10 vector cssrint
    cssxint
```

DESCRIPTION

NOTE: At the moment, NetBSD does not support IMP, so this manual page is not relevant.

The **css** device provides a Local Host/Distant Host interface to an IMP. It is normally used when participating in the DARPA Internet. The controller itself is not accessible to users, but instead provides the hardware support to the IMP interface described in imp(4). The configuration entry for the imp(4) must also include the *pseudo-device* as shown above.

DIAGNOSTICS

css%d: not alive. The initialization routine was entered even though the device did not autoconfigure. This is indicates a system problem.

css%d: can't initialize. Insufficient UNIBUS resources existed to initialize the device. This is likely to occur when the device is run on a buffered data path on an 11/750 and other network interfaces are also configured to use buffered data paths, or when it is configured to use buffered data paths on an 11/730 (which has none).

css%d: imp doesn't respond, icsr=%b. The driver attempted to initialize the device, but the IMP failed to respond after 500 tries. Check the cabling.

css%d: stray output interrupt csr=%b. An interrupt occurred when no output had previously been started.

css%d: output error, ocsr=%b icsr=%b. The device indicated a problem sending data on output.

css%d: recv error, csr=%b. The device indicated a problem receiving data on input.

css%d: bad length=%d. An input operation resulted in a data transfer of less than 0 or more than 1008 bytes of data into memory (according to the word count register). This should never happen as the maximum size of a host-IMP message is 1008 bytes.

HISTORY

The **css** interface appeared in 4.2BSD.

NAME

ct — CS/80 cartridge tape interface

SYNOPSIS

```
ct0 at hpibbus? slave ?
```

DESCRIPTION

The cartridge tape interface as found in the 7946 and 9144 products provides a standard tape drive interface as described in mtio(4) with the following exceptions:

1. There is only one density.
2. Only the “raw” interface is supported.
3. The MTIOCTOP ioctl(2) is limited. In particular, the command, MTFSR is not supported.
4. The MTIOCGET ioctl(2) is not supported.
5. The record size for read and write operations must be between 1K and 64K inclusive.

Special files rct0 through rct3 refer to rewind on close interfaces to drives 0 to 3. Files rct4 through rct7 refer to no-rewind interfaces. Files rct8 through rct11 refer to streaming rewind on close interfaces. (Only 9144 type devices can stream.) Lastly, rct12 through rct15 refer to streaming no-rewind interfaces.

SEE ALSO

mt(1), tar(1), mtio(4)

BUGS

Read and writes of less than 1024 bytes will not behave as expected.

NAME

ct — C/A/T phototypesetter interface

SYNOPSIS

```
ct0 at uba0 csr 0167760 vector ctintr
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

This is an interface to either a Graphic Systems C/A/T phototypesetter or an Autologic APS-Micro5 using a DR-11 C interface.

The **ct** is a write only device.

FILES

`/dev/cat`

DIAGNOSTICS

None.

SEE ALSO

`troff`(1)

Phototypesetter interface specification.

HISTORY

The **ct** driver appeared in 4.1BSD.

NAME

cuda — support for CUDA microcontrollers found in many Power Macintosh and compatible computers

SYNOPSIS

```
cuda* at obio?
nadb* at cuda?
iic* at cuda?
```

DESCRIPTION

The **cuda** driver provides support for the CUDA microcontroller found in many Power Macintosh and compatible computers, mostly Old World desktop machines. CUDA controls the real time clock, ADB, power and on some machines an iic(9) bus.

SEE ALSO

nadb(4), obio(4), pmu(4), sgsmix(4), iic(9)

NAME

cue — CATC USB-EL1201A USB Ethernet driver

SYNOPSIS

cue* **at uhub?**

HARDWARE

The **cue** driver supports the following adapters:

Belkin F5U111
CATC Netmate
CATC Netmate II

DESCRIPTION

The **cue** driver provides support for USB Ethernet adapters based on the Computer Access Technology Corporation's USB-EL1202A chipset.

The USB-EL1202A supports a 512-bit multicast hash filter, single perfect filter entry for the station address and promiscuous mode. Packets are received and transmitted over separate USB bulk transfer endpoints.

The CATC adapter supports only 10Mbps half-duplex mode, hence there are no `ifmedia`(4) modes to select.

For more information on configuring this device, see `ifconfig`(8).

DIAGNOSTICS

cue%d: watchdog timeout A packet was queued for transmission and a transmit command was issued, however the device failed to acknowledge the transmission before a timeout expired.

cue%d: no memory for rx list The driver failed to allocate an mbuf for the receiver ring.

SEE ALSO

`arp`(4), `netintro`(4), `usb`(4), `ifconfig`(8)

HISTORY

The **cue** device driver first appeared in FreeBSD 4.0, and in NetBSD 1.5.

AUTHORS

The **cue** driver was written by Bill Paul ⟨wpaul@ee.columbia.edu⟩.

NAME

cy — Cyclades Cyclom-{4, 8, 16, 32}Y asynchronous comms board serial device driver

SYNOPSIS

```
cy0 at isa? iomem 0xd4000 irq 12
cy* at pci? dev ? function ?
```

DESCRIPTION

This driver provides an interface to Cyclades Cyclom-4Y, Cyclom-8Y, Cyclom-16Y, and Cyclom-32Y asynchronous multiport serial boards. These boards are based around Cirrus Logic CD1400 communication controllers.

The device minor numbers for this driver are encoded as follows:

```
d c c p p p p p   - bits in the minor device number

bits    meaning
----    -------
ppppp   physical serial line (i.e. port) to use:
           0-3 on Cyclom-4Y
           0-7 on Cyclom-8Y
           0-15 on Cyclom-16Y
           0-31 on Cyclom-32Y

cc      card unit number; note this limits the driver to
       four cards per system

d       set to use as a dial-out line
```

FLOW CONTROL

The **cy** driver makes use of the CD1400's automatic CTS flow control. In addition, the CD1400's automatic input flow control can be used. This requires the kernel configuration option *CY_HW_RTS* and a special cable that exchanges the RTS and DTR lines.

DIAGNOSTICS

cy%d: port %d: can't allocate tty There is not enough memory to allocate tty data structures.

cy%d: can't allocate input buffer There is not enough memory to allocate the data input buffer.

Additional debugging output can be enable with the kernel configuration option *CY_DEBUG*. Diagnostic counters may be enabled with the kernel configuration option *CY_DEBUG1*.

SEE ALSO

`termios`(4), `tty`(4)

AUTHORS

The **cy** driver was written by Timmo Rossi.

BUGS

Support for the Cyclom-32Y has not been tested.

NAME

cypide — Cypress IDE disk controllers driver

SYNOPSIS

cypide* **at pci? dev ? function ? flags 0x0000**

DESCRIPTION

The **cypide** driver supports the Cypress 82C693 IDE controllers, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **cypide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

The timings used for the PIO and DMA modes for controllers listed above are for a PCI bus running at 30 or 33 MHz. This driver may not work properly on overclocked systems.

NAME

cz — Cyclades-Z series multi-port serial adapter device driver

SYNOPSIS

```
cz* at pci? dev ? function ?
```

DESCRIPTION

The **cz** device driver supports the Cyclades-Z series of multi-port serial adapters. The Cyclades-Z is an intelligent serial controller comprising:

- PLX9060ES PCI bus interface
- Xilinx XC5204 FPGA
- IDT R3052 MIPS CPU

The MIPS CPU runs firmware provided by the device driver. Communication with the MIPS is performed by modifying data structures located in board local RAM or host RAM.

The Cyclades-Z comes in three basic flavors:

- Cyclades-8Zo rev. 1 -- This is an older 8-port board with no FPGA. The serial ports are provided by an octopus cable.
- Cyclades-8Zo rev. 2 -- This is the newer 8-port board. The serial ports are provided by an octopus cable.
- Cyclades-Ze -- This is the expandable version of the Cyclades-Z. It uses an HD-50 SCSI cable to connect the board to a 1U rack mountable serial expansion box. Each box has 16 RJ45 serial ports, and up to 4 boxes may be chained together, for a total of 64 ports. Boxes 3 and 4 require their own external power supply, otherwise the firmware will refuse to start (as it cannot communicate with the UARTs in those boxes).

The Cyclades-Z has several features to improve performance under high serial I/O load:

- The board may operate in interrupt-driven mode or polled mode to reduce interrupt load.
- Each channel has a large input and output buffer.
- Each channel may be programmed to generate an interrupt based on reception of a specific character, e.g. a PPP End-Of-Frame character.
- The MIPS CPU on the board performs all flow-control handling.

FILES

```
/dev/ttyCZnnnn -- dial-in (normal) TTY device
/dev/dtyCZnnnn -- dial-out TTY device
```

SEE ALSO

pci(4), termios(4), tty(4)

HISTORY

The **cz** driver first appeared in NetBSD 1.5.

AUTHORS

The **cz** driver was written by Jason R. Thorpe ⟨thorpej@zembu.com⟩ and
Bill Studenmund ⟨wrstuden@zembu.com⟩ of Zembu Labs, Inc.

BUGS

The **cz** driver does not currently implement communication via host RAM. While this may improve performance by reducing the number of PCI memory space read/write cycles, it is not straightforward to implement with the current bus_dma(9) API.

Interrupt mode has not been tested.

There is no support for reading or writing the EEPROM connected to the PLX PCI bus controller.

NAME

daic — isdn4bsd driver for EICON.Diehl active ISDN cards

SYNOPSIS

```
daic0 at isa? iomem 0xd8000 irq 10
```

DESCRIPTION

The **daic** driver supports the old Diehl active cards: *S*, *SX*, *SXn*, *SCOM* and *QUADRO*.

For a *QUADRO* card, the driver will detect the board type and use all four ports, each attached as a controller of its own to the isdn4bsd system, which can be listed using the daicctl utility.

The **daic** driver interfaces the ISDN card to the ISDN4BSD kernel subsystem. All lower layer ISDN control is handled by the card. This should allow you to run any national ISDN protocol delivered by EICON.Diehl for your card, but the driver has only been tested with the DSS1 protocol and some parts of the cards interface are ISDN protocol dependent.

The **daic** driver is written to conform to the software interface documented by Diehl in their **ISDN-Karten Benutzerhandbuch** from 1992.

MICROCODE DOWNLOAD

Every active card needs its own operating software before it can work. You have to download this to the card before using it with isdn4bsd. Use the daicctl utility to do this, i.e. call **daicctl -d te_etsi.sx 1** to download the file **te_etsi.sx** to controller number 1. Use **daicctl -l** to list all available controllers (and ports). You have to select the correct ISDN protocol file for your ISDN interface, see the Diehl documentation for details.

The cards bootstrap process involves another file, which is independent of the card type you use and the protocol you run. It is called **download.bin** in current versions of the Diehl software distribution and has to be copied to the kernel compile directory under **dev/microcode/daic** and converted into a header file used when compiling the kernel by running **make** in that directory. Your kernel compile will fail and remind you of this if you forget to do this. Due to copyright restrictions we cannot distribute the driver with this file integrated. But if you own a card, you do have the file (or can get it from the Diehl web server).

SEE ALSO

daicctl(1)

BUGS

The driver is not yet finished.

NAME

dbri — SUNW,DBRI audio device driver

SYNOPSIS

```
dbri* at sbus? slot ? offset ?
audio* at audiobus?
```

DESCRIPTION

The **dbri** driver provides support for the audio part of DBRI ISDN controllers found in various sun4m class machines. It works with both onboard codecs and external speakerboxes.

SEE ALSO

audio(4), sbus(4)

NAME

dcl — HP 98628A serial communications link

SYNOPSIS

```
dcl0 at dio? scode ? flags 0x1
```

DESCRIPTION

The 98628A is a buffered EIA RS-232C (CCITT V.28) communications interface. It has one port with full modem control.

Input and output for each line may set to one of following baud rates; 0, 50, 75, 110, 134.5, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200.

An optional argument *flags* may be set to 1 if the line should be treated as hard-wired with carrier always present, or to 0 if modem control is desired.

Use HP cable "98626 & 98628 opts.002, RS232-C DCE CABLE, 5061-4216" to attach non-modem devices. Use HP cable "98626 & 98628 opts.001, RS232-C DTE CABLE, 5061-4215" to attach modems.

The 98628A has a 256 byte input silo and a 256 output silo. Input interrupts happen on a per character basis.

The high water and low water marks in the kernel tty routines are completely inappropriate for a device like this with a large input buffer. Don't use tandem mode if possible. A fast system can handle input at 19.2K baud without receive overflow.

For output to devices that make heavy use of XON/XOFF a write size of less than 256 will improve performance marginally.

FILES

`/dev/ttyl[0-9]`

DIAGNOSTICS

dcl%d: error 0x%x RESET CARD. Where the errors are encoded:

0x06 card failure
0x0d uart receive overflow
0x0e receive overflow
0x0f missing external clock
0x10 cts false too long
0x11 lost carrier
0x12 activity timeout
0x13 connection not established
0x19 illegal databits/parity
0x1a register address out of range
0x1b register value out of range
0x-- unknown error

SEE ALSO

`tty`(4)

BUGS

The **dcl** device is not actually supported in NetBSD. This man page is only for information purposes in case someone wishes to port the driver from 4.3BSD.

Breaks received at a faster rate than 1 break every second will be recognized as a single break.

Console use is not supported.

The RS-422/423/499, MTS-DSN/DL modes of the card are not supported.

NAME

dcm — HP98642A serial communications multiplexer

SYNOPSIS

dcm* at dio? scode ? flags 0xe

DESCRIPTION

The 98642A is a four port EIA RS-232C (CCITT V.28) communications multiplexer. The 98642A has three direct-connect ports and one port with full modem control.

Input and output for each line may set to one of following baud rates; 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400.

Flags is usually specified as 0xe since 3 of the 4 ports (1-3) do not support modem control and should be treated as hard-wired with carrier always present. If port 0 does not have the need for modem control then flags can be specified as `0xf`.

Each port on the 98642A has a 128 byte input silo and a 16 byte output silo. Interrupts happen on a per character basis unless the interrupt rate for the card reaches 70 interrupts per second at which time the driver changes to a 16.7ms (60 interrupts per second) polling scheme until the interrupt rate drops.

FILES

`/dev/tty0[0-9a-f]`

DIAGNOSTICS

dcm%d port%d: silo overflow Input Silo has overflowed and incoming data has been lost.

dcm%d port%d: uart overflow The 3 character buffer in the UART has overflowed.

SEE ALSO

`dio`(4), `tty`(4)

BUGS

Total throughput per card, all ports together, is limited to 76800 bits per second continuous input rate.

NAME

ddb — in-kernel debugger

SYNOPSIS

options DDB

To enable history editing:
options DDB_HISTORY_SIZE=integer

To disable entering **ddb** upon kernel panic:
options DDB_ONPANIC=0

To enable teeing all **ddb** output to the kernel msgbuf:
options DDB_TEE_MSGBUF=1

To specify commands which will be executed on each entry to **ddb**:
options DDB_COMMANDONENTER="trace;show registers"
In this case, "trace" and then "show registers" will be executed automatically.

To enable extended online help:
options DDB_VERBOSE_HELP.

DESCRIPTION

ddb is the in-kernel debugger. It may be entered at any time via a special key sequence, and optionally may be invoked when the kernel panics.

ENTERING THE DEBUGGER

Unless DDB_ONPANIC is set to 0, **ddb** will be activated whenever the kernel would otherwise panic.

ddb may also be activated from the console. In general, sending a break on a serial console will activate **ddb**. There are also key sequences for each port that will activate **ddb** from the keyboard:

alpha	<Ctrl>-<Alt>-<Esc> on PC style keyboards.
amd64	<Ctrl>-<Alt>-<Esc> <Break> on serial console.
amiga	<LAlt>-<LAmiga>-<F10>
atari	<Alt>-<LeftShift>-<F9>
hp300	<Shift>-<Reset>
hp700	+++++ (five plus signs) <Break> on serial console.
hpcarm	<Ctrl>-<Alt>-<Esc>
hpcmips	<Ctrl>-<Alt>-<Esc>
hpcsh	<Ctrl>-<Alt>-<Esc>
i386	<Ctrl>-<Alt>-<Esc> <Break> on serial console.
mac68k	<Command>-<Power>, or the Interrupt switch.
macppc	Some models: <Command>-<Option>-<Power>
mvme68k	Abort switch on CPU card.
pmax	<Do> on LK-201 rcons console. <Break> on serial console.
sparc	<L1>-A, or <Stop>-A on a Sun keyboard. <Break> on serial console.
sparc64	<L1>-A, or <Stop>-A on a Sun keyboard. <Break> on serial console.

sun3	<L1>-A, or <Stop>-A on a Sun keyboard. <Break> on serial console.
vax	<Esc>-<Shift>-D on serial console.
x68k	Interrupt switch on the body.
xen	+++++ (five plus signs)

The key sequence to activate **ddb** can be changed by modifying "hw.cnmagic" with `sysctl`(8). If the console is not dedicated to **ddb** the sequence should not be easily typed by accident. In addition, **ddb** may be explicitly activated by the debugging code in the kernel if **DDB** is configured.

COMMAND SYNTAX

The general command syntax is:

command[*/modifier*] *address* [,*count*]

The current memory location being edited is referred to as *dot*, and the next location is *next*. They are displayed as hexadecimal numbers.

Commands that examine and/or modify memory update *dot* to the address of the last line examined or the last location modified, and set *next* to the next location to be examined or modified. Other commands don't change *dot*, and set *next* to be the same as *dot*.

A blank line repeats the previous command from the address *next* with the previous **count** and no modifiers. Specifying **address** sets *dot* to the address. If **address** is omitted, *dot* is used. A missing **count** is taken to be 1 for printing commands, and infinity for stack traces.

The syntax:

,*count*

repeats the previous command, just as a blank line does, but with the specified **count**.

ddb has a `more`(1)-like functionality; if a number of lines in a command's output exceeds the number defined in the *lines* variable, then **ddb** displays "--db more--" and waits for a response, which may be one of:

⟨return⟩	one more line.
⟨space⟩	one more page.
q	abort the current command, and return to the command input mode.

You can set *lines* variable to zero to disable this feature.

If **ddb** history editing is enabled (by defining the
options DDB_HISTORY_SIZE=num
kernel option), then a history of the last **num** commands is kept. The history can be manipulated with the following key sequences:

<Ctrl>-P	retrieve previous command in history (if any).
<Ctrl>-N	retrieve next command in history (if any).

COMMANDS

ddb supports the following commands:

!*address*[(*expression*[*,...*])]
A synonym for **call**.

break[**/u**] *address*[,*count*]
Set a breakpoint at *address*. If *count* is supplied, continues (*count*-1) times before stopping at the breakpoint. If the breakpoint is set, a breakpoint number is printed with '#'. This number can be

used to **delete** the breakpoint, or to add conditions to it.

If **/u** is specified, set a breakpoint at a user-space address. Without **/u**, *address* is considered to be in the kernel-space, and an address in the wrong space will be rejected, and an error message will be emitted. This modifier may only be used if it is supported by machine dependent routines.

Warning: if a user text is shadowed by a normal user-space debugger, user-space breakpoints may not work correctly. Setting a breakpoint at the low-level code paths may also cause strange behavior.

bt[**/ul**] [*frame-address*][,*count*]
A synonym for **trace**.

bt/t[**/ul**] [*pid*][,*count*]
A synonym for **trace/t**.

bt/a[**/ul**] [*lwpaddr*][,*count*]
A synonym for **trace/a**.

call *address*[(*expression*[*,...*])]
Call the function specified by *address* with the argument(s) listed in parentheses. Parentheses may be omitted if the function takes no arguments. The number of arguments is currently limited to 10.

continue[**/c**]
Continue execution until a breakpoint or watchpoint. If **/c** is specified, count instructions while executing. Some machines (e.g., pmax) also count loads and stores.

Warning: when counting, the debugger is really silently single-stepping. This means that single-stepping on low-level may cause strange behavior.

delete *address* |*#number*
Delete a breakpoint. The target breakpoint may be specified by *address*, as per **break**, or by the breakpoint number returned by **break** if it's prefixed with '**#**'.

dmesg [*count*]
Prints the contents of the kernel message buffer. The optional *count* argument will limit printing to at most the last *count* bytes of the message buffer.

dwatch *address*
Delete the watchpoint at *address* that was previously set with **watch** command.

examine[*/modifier*] *address*[,*count*]
Display the address locations according to the format in *modifier*. Multiple modifier formats display multiple locations. If *modifier* isn't specified, the modifier from the last use of **examine** is used.

The valid format characters for *modifier* are:

- **b** examine bytes (8 bits).
- **h** examine half-words (16 bits).
- **l** examine words (legacy "long", 32 bits).
- **L** examine long words (implementation dependent)
- **a** print the location being examined.
- **A** print the location with a line number if possible.
- **x** display in unsigned hex.
- **z** display in signed hex.
- **o** display in unsigned octal.
- **d** display in signed decimal.

u display in unsigned decimal.
r display in current radix, signed.
c display low 8 bits as a character. Non-printing characters as displayed as an octal escape code (e.g., '\000').
s display the NUL terminated string at the location. Non-printing characters are displayed as octal escapes.
m display in unsigned hex with a character dump at the end of each line. The location is displayed as hex at the beginning of each line.
i display as a machine instruction.
I display as a machine instruction, with possible alternative formats depending upon the machine:

alpha	print register operands
m68k	use Motorola syntax
vax	don't assume that each external label is a procedure entry mask

kill *pid*[,*signal_number*]
Send a signal to the process specified by the *pid*. Note that *pid* is interpreted using the current radix (see **trace/t** command for details). If *signal_number* isn't specified, the SIGTERM signal is sent.

match[**/p**]
A synonym for **next**.

next[**/p**]
Stop at the matching return instruction. If **/p** is specified, print the call nesting depth and the cumulative instruction count at each call or return. Otherwise, only print when the matching return is hit.

print[**/axzodurc**] *address* [*address ...*]
Print addresses *address* according to the modifier character, as per **examine**. Valid modifiers are: **/a**, **/x**, **/z**, **/o**, **/d**, **/u**, **/r**, and **/c** (as per **examine**). If no modifier is specified, the most recent one specified is used. *address* may be a string, and is printed "as-is". For example:

```
print/x "eax = " $eax "\necx = " $ecx "\n"
```

will produce:

```
eax = xxxxxx
ecx = yyyyyy
```

ps[**/a**][**/n**][**/w**][**/l**]
A synonym for **show all procs**.

reboot [*flags*]
Reboot, using the optionally supplied boot *flags*, which is a bitmask supporting the same values as for reboot(2). Some of the more useful flags:

Value	Name	Description
0x1	RB_ASKNAME	Ask for file name to reboot from
0x2	RB_SINGLE	Reboot to single user mode
0x4	RB_NOSYNC	Don't sync before reboot
0x8	RB_HALT	Halt instead of reboot
0x40	RB_KDB	Boot into kernel debugger
0x100	RB_DUMP	Dump unconditionally before reboot
0x808	RB_POWERDOWN	Power off (or at least halt)

Note: Limitations of the command line interface preclude specification of a boot string.

search[**/bhl**] *address value* [*mask*] [,*count*]
Search memory from *address* for *value*. The unit size is specified with a modifier character, as per **examine**. Valid modifiers are: **/b**, **/h**, and **/l**. If no modifier is specified, **/l** is used.

This command might fail in interesting ways if it doesn't find *value*. This is because **ddb** doesn't always recover from touching bad memory. The optional *count* limits the search.

set $*variable* [**=**] *expression*
Set the named variable or register to the value of *expression*. Valid variable names are described in **VARIABLES**.

show all callout
Display information about callouts in the system. See callout(9) for more information on callouts.

show all pages
Display basic information about all physical pages managed by the VM system. For more detailed information about a single page, use **show page**.

show all pools[**/clp**]
Display all pool information. Modifiers are the same as **show pool**.

show all procs[**/a**][**/n**][**/w**][**/l**]
Display all process information. Valid modifiers:

/n show process information in a ps(1) style format (this is the default). Information printed includes: process ID, parent process ID, process group, UID, process status, process flags, process command name, and process wait channel message.

/a show the kernel virtual addresses of each process' proc structure, u-area, and vmspace structure. The vmspace address is also the address of the process' vm_map structure, and can be used in the **show map** command.

/w show each process' PID, command, system call emulation, wait channel address, and wait channel message.

/l show each process' associated LWP information, including each LWP's LID, flags, kernel LWP structure address, u-area, and wait channel.

show arptab
Dump the entire AF_INET routing table. This command is available only on systems which support inet and ARP.

show breaks
Display all breakpoints.

show buf[**/f**] *address*
Print the struct buf at *address*. The **/f** does nothing at this time.

show event[**/f**]
Print all the non-zero evcnt(9) event counters. If **/f** is specified, all event counters with a count of zero are printed as well.

show lock *address*
Display information about a lock at *address*. This command is useful only if a kernel is compiled with **options LOCKDEBUG**.

show malloc *address*
If *address* is supplied, display the kernel memory allocator's idea on the allocation status for it. Also, print out global statistics for the memory allocator. This command is useful only if a kernel is

compiled with **options MALLOC_DEBUG**.

show map[**/f**] *address*
Print the vm_map at *address*. If **/f** is specified, the complete map is printed.

show mount[**/f**] *address*
Print the mount structure at *address*. If **/f** is specified, the complete vnode list is printed.

show mbuf[**/c**] *address*
Print the mbuf structure at *address*. If **/c** is specified, the mbufs in the chain are followed.

show ncache *address*
Dump the namecache list associated with vnode at *address*.

show object[**/f**] *address*
Print the vm_object at *address*. If **/f** is specified, the complete object is printed.

show page[**/f**] *address*
Print the vm_page at *address*. If **/f** is specified, the complete page is printed.

show pool[**/clp**] *address*
Print the pool at *address*. Valid modifiers:
/c Print the cachelist and its statistics for this pool.
/l Print the log entries for this pool.
/p Print the pagelist for this pool.

show registers[**/u**]
Display the register set. If **/u** is specified, display user registers instead of kernel registers or the currently save one.

Warning: support for **/u** is machine dependent. If not supported, incorrect information will be displayed.

show sched_qs
Print the state of the scheduler's run queues. For each run queue that has an LWP, the run queue index and the list of LWPs will be shown. If the run queue has LWPs, but the sched_whichqs bit is not set for that queue, the queue index will be prefixed with a '!'.

show uvmexp
Print a selection of UVM counters and statistics.

show vnode[**/f**] *address*
Print the vnode at *address*. If **/f** is specified, the complete vnode is printed.

show watches
Display all watchpoints.

sifting[**/F**] *string*
Search the symbol tables for all symbols of which *string* is a substring, and display them. If **/F** is specified, a character is displayed immediately after each symbol name indicating the type of symbol.

For `a.out`(5)-format symbol tables, absolute symbols display **@**, text segment symbols display ∗, data segment symbols display **+**, BSS segment symbols display **-**, and filename symbols display **/**. For ELF-format symbol tables, object symbols display **+**, function symbols display ∗, section symbols display **&**, and file symbols display **/**.

To sift for a string beginning with a number, escape the first character with a backslash as:

```
sifting \386
```

step[**/p**] [,*count*]
Single-step *count* times. If **/p** is specified, print each instruction at each step. Otherwise, only print the last instruction.

Warning: depending on the machine type, it may not be possible to single-step through some low-level code paths or user-space code. On machines with software-emulated single-stepping (e.g., pmax), stepping through code executed by interrupt handlers will probably do the wrong thing.

sync Force a crash dump, and then reboot.

trace[**/u**[**l**]] [*frame-address*][,*count*]
Stack trace from *frame-address*. If **/u** is specified, trace user-space, otherwise trace kernel-space. *count* is the number of frames to be traced. If *count* is omitted, all frames are printed. If **/l** is specified, the trace is printed and also stored in the kernel message buffer.

Warning: user-space stack trace is valid only if the machine dependent code supports it.

trace/t[**l**] [*pid*][,*count*]
Stack trace by "thread" (process, on NetBSD) rather than by stack frame address. Note that *pid* is interpreted using the current radix, whilst **ps** displays pids in decimal; prefix *pid* with '0t' to force it to be interpreted as decimal (see **VARIABLES** section for radix). If **/l** is specified, the trace is printed and also stored in the kernel message buffer.

Warning: trace by pid is valid only if the machine dependent code supports it.

trace/a[**l**] [*lwpaddr*][,*count*]
Stack trace by light weight process (LWP) address rather than by stack frame address. If **/l** is specified, the trace is printed and also stored in the kernel message buffer.

Warning: trace by LWP address is valid only if the machine dependent code supports it.

until[**/p**]
Stop at the next call or return instruction. If **/p** is specified, print the call nesting depth and the cumulative instruction count at each call or return. Otherwise, only print when the matching return is hit.

watch *address*[,*size*]
Set a watchpoint for a region. Execution stops when an attempt to modify the region occurs. *size* defaults to 4.

If you specify a wrong space address, the request is rejected with an error message.

Warning: attempts to watch wired kernel memory may cause an unrecoverable error in some systems such as i386. Watchpoints on user addresses work the best.

whatis *address*
Describe what an address is.

write[**/bhl**] *address expression* [*expression ...*]
Write the *expression*s at succeeding locations. The unit size is specified with a modifier character, as per **examine**. Valid modifiers are: **/b**, **/h**, and **/l**. If no modifier is specified, **/l** is used.

Warning: since there is no delimiter between *expression*s, strange things may occur. It's best to enclose each *expression* in parentheses.

x[*/modifier*] *address*[,*count*]
A synonym for **examine**.

MACHINE-SPECIFIC COMMANDS

The "glue" code that hooks **ddb** into the NetBSD kernel for any given port can also add machine specific commands to the **ddb** command parser. All of these commands are preceded by the command word *machine* to indicate that they are part of the machine-specific command set (e.g. **machine reboot**). Some of these commands are:

ALPHA

halt Call the PROM monitor to halt the CPU.

reboot Call the PROM monitor to reboot the CPU.

ARM32

panic Print the current "panic" string.

frame Given a trap frame address, print out the trap frame.

MIPS

cp0 Dump CP0 (coprocessor 0) register values.

kvtop Print the physical address for a given kernel virtual address.

tlb Print out the Translation Lookaside Buffer (TLB). Only works in NetBSD kernels compiled with DEBUG option.

SH3

tlb Print TLB entries

cache Print cache entries

frame Print switch frame and trap frames.

stack Print kernel stack usage. Only works in NetBSD kernels compiled with the KSTACK_DEBUG option.

SPARC

prom Exit to the Sun PROM monitor.

SPARC64

ctx Print process context information.

cpu Switch to another cpu.

dtlb Print data translation look-aside buffer context information.

dtsb Display data translation storage buffer information.

kmap Display information about the listed mapping in the kernel pmap. Use the “f” modifier to get a full listing.

extract Extract the physical address for a given virtual address from the kernel pmap.

fpstate Dump the FPU state.

itlb Print instruction translation look-aside buffer context information.

itsb Display instruction translation storage buffer information.

lwp Display a struct lwp

pcb Display information about the “struct pcb” listed.

pctx Attempt to change process context.

page Display the pointer to the “struct vm_page” for this physical address.

phys Display physical memory.

pmap Display the pmap. Use the “f” modifier to get a fuller listing.

proc Display some information about the process pointed to, or curproc.

prom Enter the OFW PROM.

pv Display the "struct pv_entry" pointed to.
sir Reset the machine and enter prom (do a Software Initiated Reset).
stack Dump the window stack. Use the "u" modifier to get userland information.
tf Display full trap frame state. This is most useful for inclusion with bug reports.
ts Display trap state.
traptrace Display or set trap trace information. Use the "r" and "f" modifiers to get reversed and full information, respectively.
uvmdump Dumps the UVM histories.
watch Set or clear a physical or virtual hardware watchpoint. Pass the address to be watched, or "0" (or omit the address) to clear the watchpoint. Optional modifiers are "p" for physical address, "r" for trap on read access (default: trap on write access only), "b" for 8 bit width, "h" for 16 bit, "l" for 32 bit or "L" for 64 bit.
window Print register window information. Argument is a stack frame number (0 is top of stack, which is used when no index is given).

SUN3 and SUN3X

abort Drop into monitor via abort (allows continue).
halt Exit to Sun PROM monitor as in `halt`(8).
reboot Reboot the machine as in `reboot`(8).
pgmap Given an address, print the address, segment map, page map, and Page Table Entry (PTE).

VARIABLES

ddb accesses registers and variables as **$***name*. Register names are as per the **show registers** command. Some variables are suffixed with numbers, and may have a modifier following a colon immediately after the variable name. For example, register variables may have a 'u' modifier to indicate user register (e.g., `$eax:u`).

Built-in variables currently supported are:

lines The number of lines. This is used by the **more** feature. When this variable is set to zero the **more** feature is disabled.
maxoff Addresses are printed as `'symbol'+offset` unless `offset` is greater than *maxoff*.
maxwidth The width of the displayed line. **ddb** wraps the current line by printing new line when *maxwidth* column is reached. When this variable is set to zero **ddb** doesn't perform any wrapping.
onpanic If non-zero (the default), **ddb** will be invoked when the kernel panics. If the kernel configuration option

```
options DDB_ONPANIC=0
```

is used, *onpanic* will be initialized to off.
fromconsole
If non-zero (the default), the kernel allows to enter **ddb** from the console (by break signal or special key sequence). If the kernel configuration option

```
options DDB_FROMCONSOLE=0
```

is used, *fromconsole* will be initialized to off.
radix Input and output radix.
tabstops Tab stop width.
tee_msgbuf If explicitly set to non zero (zero is the default) all **ddb** output will not only be displayed on screen but also be fed to the msgbuf. The default of the variable can be set using the kernel configuration option

```
options DDB_TEE_MSGBUF=1
```

which will initialize *tee_msgbuf* to be 1. This option is especially handy for poor souls who don't have a serial console but want to recall **ddb** output from a crash investigation. This option is more generic than the /l command modifier possible for selected com-

mands as discussed above to log the output. Mixing both /l and this setting can give double loggings.

All built-in variables are accessible via `sysctl`(3).

EXPRESSIONS

Almost all expression operators in C are supported, except '~', '^', and unary '&'. Special rules in **ddb** are:

identifier name of a symbol. It is translated to the address (or value) of it. '.' and ':' can be used in the identifier. If supported by an object format dependent routine, [*filename*:]*function*[:*line number*], [*filename*:]*variable*, and *filename*[:*line number*], can be accepted as a symbol. The symbol may be prefixed with *symbol_table_name*:: (e.g., `emulator::mach_msg_trap`) to specify other than kernel symbols.

number number. Radix is determined by the first two characters: '0x' - hex, '0o' - octal, '0t' - decimal, otherwise follow current radix.

. *dot*

+ *next*

.. address of the start of the last line examined. Unlike *dot* or *next*, this is only changed by the **examine** or **write** commands.

" last address explicitly specified.

$*name* register name or variable. It is translated to the value of it. It may be followed by a ':' and modifiers as described above.

a binary operator which rounds up the left hand side to the next multiple of right hand side.

∗*expr* expression indirection. It may be followed by a ':' and modifiers as described above.

SEE ALSO

`reboot`(2), `options`(4), `reboot`(8), `sysctl`(8), `cnmagic`(9)

HISTORY

The **ddb** kernel debugger was written as part of the MACH project at Carnegie-Mellon University.

NAME

ddc — VESA Display Data Channel V2 devices

SYNOPSIS

```
ddc at iic? addr 0x50
```

DESCRIPTION

The **ddc** driver provides support for accessing the VESA Display Data Channel Version 2 supported by many video displays.

BUGS

This driver does not provide any mechanism for access from user applications. Its only use at this point is to provide a means for framebuffer device drivers to query monitor description data (EDID) using a specialized in-kernel API. No support for sending control commands to display devices is provided.

SEE ALSO

`iic`(4), `ddc`(9), `edid`(9)

HISTORY

The **ddc** device appeared in NetBSD 4.0.

AUTHORS

Garrett D'Amore ⟨gdamore@NetBSD.org⟩

NAME

ddn — DDN Standard Mode X.25 IMP network interface

SYNOPSIS

```
ddn0 at uba0 csr 166740 vector ddnintr
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **ddn** device provides a DDN Standard Mode X.25 interface to an IMP using the ACC ACP625 X.25 board. It is normally used for connecting to the Defense Data Network (DDN). The controller itself is not accessible to users, but instead provides a network interface for the Internet Protocol described in ip(4).

DIAGNOSTICS

ddn%d: not alive. The initialization routine was entered even though the device did not autoconfigure. This indicates a system problem.

ddn%d: failed getting UBA resources for lcn %d." Insufficient UNIBUS resources existed to initialize the device. This is likely to be a shortage of UNIBUS mapping registers.

ddn%d: couldn't get X25 init buffer. This indicates that an *mbuf* could not be allocated for sending the initialization message to the ACP625.

DDN: illegal X25 address length!
DDN: illegal X25 address format! These errors indicate a problem with the called X.25 address received from the IMP on an incoming call.

X25 RESET on lcn = %d. This indicates that an unexpected X.25 RESET was received on the indicated LCN.

X25 INTERRUPT on lcn = %d, code = %d. This indicates that an unexpected X.25 INTERRUPT Packet was received on the indicated LCN.

ddn%d: failed to get supr msg bfr! This indicates that an *mbuf* could not be allocated for sending a supervisor message to the ACP625.

Any other error message from ddn%d: indicates a serious error detected by either the driver or the ACP625 firmware.

SEE ALSO

ip(4), netintro(4)

HISTORY

The **ddn** interface appeared in 4.3BSD.

NAME

de — DECchip 21040, 21140, 21141, 21142, and 21143 PCI Ethernet interface driver

SYNOPSIS

```
de* at pci? dev ? function ?
```

Configuration of PHYs may also be necessary. See `mii`(4).

DESCRIPTION

The **de** driver supports Ethernet and Fast Ethernet interfaces based on the Digital Equipment Corporation DECchip 21040, 21140, 21141, 21142, and 21143 model controllers for PCI bus.

21040 10BASE-T and AUI/BNC
21140 10BASE-T and 100BASE-TX
21141 10BASE-T and 100BASE-TX
21142 10BASE-T and 100BASE-TX
21143 10BASE-T and 100BASE-TX

Supported Network Interface Cards (NIC) include, but are not limited to:

Accton EN1207i
Cogent EM100
Asante AsanteFAST
DEC DE435
DEC DE450
DEC DE500
SMC 9332
Znyx NetBlaster ZX340 series (345, 348, 346)

Generally, if the NIC is for PCI bus, and the major controller chip has the DIGITAL logo on it, with one of the model numbers above, the **de** driver should recognize and control it. Beware, however, that many NIC manufacturers change the Ethernet controller chip without changing the product model number, and a physical inspection of the NIC, or a probe of the NIC with a PCI diagnostic tool is the only way to tell if a real DEC TULIP controller is present.

Multi-port interfaces are typically configured as a PCI−PCI bridge with multiple **de** instances on the PCI bus on the other side of the bridge.

DEC sold its semiconductor division to Intel. In addition, there are many "knock-offs" of the DEC TULIP Ethernet controller chips; NICs based on these chips are handled by the `tlp`(4) driver.

This **de** driver should not be confused with the NetBSD VAX **de** driver for the DEC DEUNA/DELUA Ethernet controller for UniBus.

SEE ALSO

`ifmedia`(4), `intro`(4), `mii`(4), `pci`(4), `tlp`(4), `ifconfig`(8)

`http://www.adaptec.com/`, `http://www.asante.com/`, `http://www.smc.com/`, `http://www.znyx.com/`

HISTORY

The **de** driver first appeared in NetBSD 1.1

AUTHORS

Matt Thomas ⟨matt@3am-software.com⟩.

NAME

de — DEC DEUNA/DELUA 10 Mb/s Ethernet interface

SYNOPSIS

```
de0 at uba? csr 174510
```

DESCRIPTION

The **de** interface provides access to a 10 Mb/s Ethernet network through a Digital Equipment UNIBUS Network Adapter (DEUNA) .

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **de** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

DIAGNOSTICS

de%d: hardware address %s. This is a normal autoconfiguration message noting the 6 byte physical Ethernet address of the adapter.

de%d: oerror, flags=%b tdrerr=%b (len=%d). The hardware indicated an error in transmitting a packet to the cable. The status and error flags are reported.

de%d: ierror, flags=%b lenerr=%b (len=%d). The hardware indicated an error in reading a packet from the cable. The status and error flags are reported.

The following messages indicate a probable hardware error performing the indicated operation during autoconfiguration or initialization. The two control and status registers should indicate the nature of the failure. See the hardware manual for details.

de%d: reset failed, csr0=%b csr1=%b.
de%d: ppcb failed, csr0=%b csr1=%b.
de%d: read addr failed, csr0=%b csr1=%b.
de%d: wtring failed, csr0=%b csr1=%b.
de%d: wtmode failed, csr0=%b csr1=%b.

SEE ALSO

`arp`(4), `inet`(4), `netintro`(4)

HISTORY

The **de** driver appeared in 4.3BSD.

NAME

dge — Intel i82597EX Ten Gigabit Ethernet driver

SYNOPSIS

dge* **at pci? dev ? function ?**

DESCRIPTION

The **dge** device driver supports the Intel i82597EX PRO/10GbE LR Ethernet adapter, which uses a single mode fiber (1310nm) interface.

The i82597EX supports IPv4/TCP/UDP checksumming in hardware, as well as TCP Segmentation Offloading (TSO). The driver does currently only support the hardware checksumming features. See `ifconfig`(8) for information on how to enable the hardware checksum calculations.

The driver also makes use of the `ifconfig`(8) link flags *`link0`* and *`link1`* to set the PCIX burst size. The burst size is set according to this table:

link0	*link1*	*burst size*
`off`	off	512
`on`	off	1024
`off`	on	2048
`on`	on	4096

A larger burst size will increase the transmit capacity of the card dramatically but may have negative effect on other devices in the system.

DIAGNOSTICS

dge%d: Tx packet consumes too many DMA segments, dropping... The packet consisted of too many small mbufs and could therefore not be loaded into a DMA map. This is most unlikely, the driver can currently handle up to 100 segments, but over 80 segments has been seen using large (16k) jumbo frames.

dge%s: device timeout (txfree %d txsfree %d txnext %d) The i82597EX had been given packets to send, but didn't interrupt within 5 seconds. This diagnostic is most likely the result of a hardware failure, and the chip will be reset to resume normal operation.

dge%d: Receive overrun If the computer is under heavy load, the software may not be able to keep up removing received datagrams from the receive queue, and will therefore loose datagrams. To avoid this, check that the other end is using the XON/XOFF protocol, if possible, or increase the receive descriptor ring size in the driver.

dge%d: symbol error
dge%d: parity error An error in the XGMII communication was detected. This is a hardware error in the MAC<->PHY communication bus.

dge%d: CRC error A CRC error in the received datagram was detected. The error is probably caused in the fiber communication.

dge%d: WARNING: reset failed to complete This is a fatal error and means that the hardware is broken and will most likely not function correctly.

dge%d: unable to allocate or map rx buffer %d error = %d The driver was not able to map a mbuf cluster page to a receive descriptor entry in the receive ring. Most likely the system has run out of mbuf clusters or have a too small cluster map. See the errno for more information.

SEE ALSO

`arp`(4), `ifmedia`(4), `netintro`(4), `pci`(4), `ifconfig`(8)

HISTORY

The **dge** driver first appeared in NetBSD 2.0.

AUTHORS

The **dge** driver was written by Anders Magnusson ⟨ragge@ludd.luth.se⟩.

BUGS

There should be an XGMII framework for the driver to use.

NAME

dh — DH-11/DM-11 serial multiplexer device interface

SYNOPSIS

```
dh0 at uba0 csr 0160020 vector dhrint dhxint [flags]
dm0 at uba0 csr 0170500 vector dmintr [flags]
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

A DH-11 provides 16 serial communication lines; DM-11s may optionally be paired with DH-11s to provide modem control for the lines.

An optional argument *flags* may be supplied with the device specification in the `config`(1) file indicating that the line corresponding to bit number *i* is not properly connected, and should be treated as hard-wired with carrier always present. Thus specifying `flags 0x0004` for `dh0` would cause line `ttyh2` to be treated in this way.

Normal I/O control parameters for individual lines are managed by `ioctl`(2) calls. Line speeds may be initiated via `getty`(8) and `stty`(1) or may be communicated by other programs which use `ioctl`(2) such as `ifconfig`(8), see `tty`(4).

The **dh** driver monitors the rate of input on each board, and switches between the use of character-at-a-time interrupts and input silos. While the silo is enabled during periods of high-speed input, the driver polls for input 30 times per second.

FILES

```
/dev/tty[h-o][0-9a-f]
/dev/ttyd[0-9a-f]
```

DIAGNOSTICS

dh%d: NXM. No response from UNIBUS on a DMA transfer within a timeout period. This is often followed by a UNIBUS adapter error. This occurs most frequently when the UNIBUS is heavily loaded and when devices which hog the bus (such as RK07s) are present. It is not serious.

dh%d: silo overflow. The character input silo overflowed before it could be serviced. This can happen if a hard error occurs when the CPU is running with elevated priority, as the system will then print a message on the console with interrupts disabled. It is not serious.

SEE ALSO

`tty`(4)

HISTORY

A **dh** driver appeared in Version 6 AT&T UNIX.

NAME

dhu — DHU-11/DHV-11 serial communications multiplexer

SYNOPSIS

```
dhu0 at uba0 csr 0160440
```

DESCRIPTION

A DHU-11 provides 16 communication lines.

Normal I/O control parameters for individual lines are managed by `ioctl`(2) calls. Individual DHU-11 lines may be configured to run at any of 13 speeds (50, 200 and 38400 baud are not available); the speed may be set via `getty`(8) or `stty`(1) or may be communicated by other programs which use `ioctl`(2) such as `ifconfig`(8), see `tty`(4).

The DHU-11 driver normally uses input silos and delays receiver interrupts by 20 milliseconds rather than taking an interrupt on each input character.

FILES

`/dev/tty[S-Z][0-9a-f]`

NOTES

The driver currently does not make full use of the hardware capabilities of the DHU-11, for dealing with XON/XOFF flow-control or hard-wired lines for example.

Although the devices are not the same, a DHU-11 can convince the DH-11 autoconfiguration code that it is a DH-11.

The 4 40-way cables are a pain.

SEE ALSO

`tty`(4)

HISTORY

The **dhu** driver appeared in 4.3BSD.
A new **dhu** driver showed up in 1.2BSD.

BUGS

Even if the **dhu** hardware supports DMA, the driver cannot make use of this capability.

NAME

dino — Dino and Cujo Host/PCI bridges

SYNOPSIS

```
dino* at phantomas?
com*  at dino?
pci*  at dino?
```

DESCRIPTION

This driver supports Dino and Cujo Host/PCI bridges found in [ABC][0-9][0-9][0-9]-Class workstations and GSC / HSC cards. Cujo is a 64 bit datapath version of Dino.

On some machines it may also provide an additional serial port through the com(4) driver, or PS/2 keyboard and mouse ports, though the latter are not yet supported.

MACHINES

An incomplete list of machines that use the Dino / Cujo Host/PCI bridges:

- 744
- 748[i]
- A180[C]
- B132L[+], B160L, B180L+
- C132L, C160[L], C180, C200, C240, C360
- J2240
- D390
- R380, R390

SEE ALSO

com(4), intro(4), pci(4), phantomas(4)

HISTORY

The **dino** driver appeared in OpenBSD 3.5. It was ported to NetBSD 2.0 by Jochen Kunz.

BUGS

dino bridges of revision earlier than three may exhibit data corruption on DMA. This hardware bug does not affect **cujo** or card mode **dino** bridges. See HP Service Note Numbers A4190A-01 and A4191A-01 for more details. Systems affected are those shipped before Aug 20, 1997 and of models: B132L, B160L, C160, C180, C200, C240.

NAME

dio — DIO/DIO-II bus

SYNOPSIS

```
dio0 at mainbus0
```

DESCRIPTION

This driver is for the DIO/DIO-II bus on HP 9000/3xx series workstations.

SUPPORTED DEVICES

NetBSD includes DIO drivers, sorted by device type and driver name:

Framebuffers

dvbox	HP98730 ‘‘DaVinci’’ graphics device interface
gbox	HP98700 ‘‘Gatorbox’’ graphics device interface
hyper	A1096A ‘‘Hyperion’’ graphics device interface
rbox	HP98720 ‘‘Renaissance’’ graphics device interface
topcat	HP98544 98550 ‘‘Topcat’’ and ‘‘Catseye’’ device interface

Serial interfaces

com	HP98644A serial communications interface (formerly dca)
dcm	HP98642A serial communications multiplexer

Network interfaces

le	HP98643 built-in and add-on Ethernet boards

HP-IB interfaces

fhpib	‘‘fast’’ HP-IB interface
nhpib	‘‘normal’’ HP-IB interface

SCSI interfaces

spc	HP98265A SCSI interface

SEE ALSO

com(4), dcm(4), dvbox(4), gbox(4), hpib(4), hyper(4), le(4), rbox(4), spc(4), topcat(4)

NAME

dk — Disk partition (wedge) driver

SYNOPSIS

```
options DKWEDGE_AUTODISCOVER
options DKWEDGE_METHOD_BSDLABEL
options DKWEDGE_METHOD_GPT
options DKWEDGE_METHOD_MBR
```

DESCRIPTION

The **dk** driver provides a disk-like interface to an area of a physical disk. Wedges may be configured manually with `dkctl`(8) or automatically by the kernel upon the attachment of the physical disk.

KERNEL OPTIONS

`DKWEDGE_AUTODISCOVER`	Automatically detect and configure wedges using any available methods.
`DKWEDGE_METHOD_BSDLABEL`	BSD disklabel detection method.
`DKWEDGE_METHOD_GPT`	Extensible Firmware Interface Globally Unique Identifier Partition Table (GPT) detection method.
`DKWEDGE_METHOD_MBR`	IBM PC-compatible Master Boot Record (MBR) partitioning detection method, with support for Extended MBRs.

FILES

`/dev/{,r}dk*` **dk** device special files.

SEE ALSO

`config`(1), `MAKEDEV`(8), `disklabel`(8), `dkctl`(8), `fdisk`(8)

HISTORY

The **dk** driver first appeared in NetBSD 3.0.

AUTHORS

The **dk** driver was written by Jason R. Thorpe.

NAME

dl — DL-11/DLV-11 serial device driver

SYNOPSIS

```
dl0 at uba? csr 0176500
dl1 at uba? csr 0176510
dl2 at uba? csr 0176520
dl3 at uba? csr 0176530
```

DESCRIPTION

The **dl** driver controls a DL-11-compatible asynchronous serial card, and probably things compatible with it. A four-port DLV-11J will appear four times in the device list, as the ports look like separate cards to the driver.

The **dl** driver provides the normal interface described in tty(4), but many of the configuration calls are unsupported, since their functions are handled by jumpers or switches on the serial card itself. Calls related to modem-control lines are also ignored, since these cards lack them.

There's a chance this driver might also work with an LP11, an LPV11 or even a PC11, but it hasn't been tested.

FILES

/dev/ttyJ? Special files for communicating with dl devices.

DIAGNOSTICS

dl%d: rx overrun. The character in the receive buffer wasn't read before the next character arrived, and has been lost.

dl%d: stray rx interrupt. The driver was called to service a receive interrupt, but there was nothing for it to read.

SEE ALSO

tty(4)

HISTORY

The **dl** driver was written for NetBSD 1.3.

AUTHORS

Ben Harris <bjh21@NetBSD.org>

BUGS

The DL-11 and friends only have single-character receive and transmit buffers, so an interrupt is generated for every character received or transmitted. Attempting to receive data at even moderately high rates will cause rx overruns. Fast transmission seems to be fine though.

There is no support in the driver for the paper-tape reader on an LT33 attached via a DLV-11KA or similar.

The overrun message is logged in the interrupt routine itself, which will probably just make the problem worse.

The CSR printed on startup is that of the receiver, while the interrupt vector is that of the transmitter.

In order to determine the card's interrupt vector, the driver sends a NUL to each port. This may confuse things attached to them.

The driver has so far only been tested on a DLV-11J. It may or may not work on the other cards it claims to support.

NAME

dmc — DEC DMC-11/DMR-11 point-to-point serial communications device

SYNOPSIS

```
dmc0 at uba0 csr 167600
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **dmc** interface provides access to a point-to-point communications device which runs at either 1 Mb/s or 56 Kb/s. DMC-11s communicate using the DEC DDCMP link layer protocol.

The **dmc** interface driver also supports a DEC DMR-11 providing point-to-point communication running at data rates from 2.4 Kb/s to 1 Mb/s. DMR-11s are a more recent design and thus are preferred over DMC-11s. The `NXMT` and `NRCV` constants in the driver may be increased in this case, as the DMR can accept up to 64 transmit and receive buffers, as opposed to 7 for the DMC.

The configuration flags specify how to set up the device,

0	full duplex DDCMP (normal mode)
1	DDCMP Maintenance mode (generally useless)
2	DDCMP Half Duplex, primary station
3	DDCMP Half Duplex, secondary station

The host's address must be specified with an `SIOCSIFADDR` `ioctl`(2), and the destination address specified with a `SIOCSIFDSTADDR` `ioctl`(2), before the interface will transmit or receive any packets.

ROUTING

The driver places a HOST entry in the kernel routing tables for the address given in the `SIOCSIFDSTADDR` `ioctl`(2). To use the DMC as a link between local nets, the route to the remote net must be added manually with the `route`(8) command, or by the use of the routing process `routed`(8) on each end of the link.

DIAGNOSTICS

dmc%d: bad control %o. A bad parameter was passed to the *dmcload* routine.

dmc%d: unknown address type %d. An input packet was received which contained a type of address unknown to the driver.

DMC fatal error 0%o. A fatal error in DDMCP occurred, causing the device to be restarted.

DMC soft error 0%o. A non-fatal error in DDMCP has occurred.

dmc%d: af%d not supported. The interface was handed a message which has addresses formatted in an unsuitable address family.

SEE ALSO

`inet`(4), `intro`(4)

HISTORY

The **dmc** driver appeared in 4.2BSD.

BUGS

The current version of the driver uses a link-level encapsulation so that multiple protocol types may be used. It is thus incompatible with earlier drivers, including the 4.2BSD version.

NAME

dmf — DMF-32 serial terminal multiplexor

SYNOPSIS

```
dmf0 at uba? csr 0160340 vector dmfsrint dmfsxint dmfdaint dmfdbint
     dmfrint dmfxint dmflint
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **dmf** device provides 8 lines of asynchronous serial line support. The first two of these have full modem control. The device also provides a line printer port similar to the LP-11. Other features of the DMF-32 are not supported. During autoconfiguration, the driver examines the configuration of each DMF-32 and adjusts the interrupt vectors so that fewer vector locations are used if possible.

An optional argument *flags* may be supplied with the device specification in the config file indicating that the line corresponding to bit number *i* is not properly connected, and should be treated as hard-wired with carrier always present. Thus specifying `flags 0x04` for `dmf0` would cause line `ttyA2` to be treated in this way. Flags should be set for all lines without hardware support for modem control.

Normal I/O control parameters for individual lines are managed by `ioctl`(2) calls. Line speeds may be initiated via `getty`(8) and `stty`(1) or may be communicated by other programs which use `ioctl`(2) such as `ifconfig`(8), see `tty`(4).

The serial line part of the **dmf** driver normally enables the input silos with a short timeout (30 milliseconds); this allows multiple characters to be received per interrupt during periods of high-speed input.

A line printer port on a **dmf** is designated by a minor device number of the form 128+*n*. See `MAKEDEV`(8). Column and lines per page may be changed from the default 132 columns and 66 lines by encoding the number of columns in bits 8-15 of flags and the number of lines in bits 16-23. This device does not provide the fancy output canonicalization features of the `lp`(4) driver.

FILES

```
/dev/tty[A-CE-I][0-7]
/dev/ttyd[0-7]
/dev/lp
```

DIAGNOSTICS

dmf%d: NXM line %d. No response from UNIBUS on a DMA transfer within a timeout period. This is often followed by a UNIBUS adapter error. This occurs most frequently when the UNIBUS is heavily loaded and when devices which hog the bus (such as RK07s) are present. It is not serious.

dmf%d: silo overflow. The character input silo overflowed before it could be serviced. This can happen if a hard error occurs when the CPU is running with elevated priority, as the system will then print a message on the console with interrupts disabled. It is not serious.

dmfsrint, dmfsxint, dmfdaint, dmfdbint. One of the unsupported parts of the dmf interrupted; something is amiss, check your interrupt vectors for a conflict with another device.

SEE ALSO

`tty`(4)

HISTORY

The **dmf** driver appeared in 4.2BSD.

BUGS

It should be possible to set the silo timeout with a configuration file option, as the value is a trade-off between efficiency and response time for flow control and character echo.

NAME

dmoverio — hardware-assisted data mover interface

SYNOPSIS

```
pseudo-device dmoverio

#include <dev/dmover/dmover_io.h>
```

DESCRIPTION

The **dmoverio** pseudo-device driver provides an interface to hardware-assisted data movers, which the kernel supports using the dmover(9) facility. This can be used to copy data from one location in memory to another, clear a region of memory, fill a region of memory with a pattern, and perform simple operations on multiple regions of memory, such as an XOR, without intervention by the CPU.

A **dmoverio** function always has one output region. A function may have zero or more input regions, or may use an immediate value as an input. For functions which use input regions, the lengths of each input region and the output region must be the same. All **dmoverio** functions with the same name will have the same number of and type inputs.

To use **dmoverio**, the client must first create a session. This is achieved by performing the following steps:

- Create a session handle by opening the /dev/dmoverio device.
- Select the **dmoverio** function using the DMIO_SETFUNC ioctl, which takes the following argument:

```
#define DMIO_MAX_FUNCNAME     64
struct dmio_setfunc {
        char dsf_name[DMIO_MAX_FUNCNAME];
};
```

  If the specified function is not available, the DMIO_SETFUNC ioctl will fail with an error code of ESRCH.

To submit a request for processing the following steps must be performed:

- Fill in a request structure:

```
typedef struct {
        struct iovec *dmbuf_iov;
        u_int dmbuf_iovcnt;
} dmio_buffer;

struct dmio_usrreq {
        /* Output buffer. */
        dmio_buffer req_outbuf;

        /* Input buffer. */
        union {
                uint8_t _immediate[8];
                dmio_buffer *_inbuf;
        } _req_inbuf_un;

#define req_immediate           _req_inbuf_un._immediate
#define req_inbuf               _req_inbuf_un._inbuf

        uint32_t req_id;        /* request ID; passed in response */
};
```

For functions which use an immediate value as an input, the *req_immediate* member is used to specify the value. Values smaller than 8 bytes should use the least-significant bytes first. For example, a 32-bit integer would occupy bytes 0, 1, 2, and 3.

For functions which use input regions, *req_inbuf* should point to an array of *dmio_buffer*'s.

The *req_id* should be a unique value for each request submitted by the client. It will be passed back unchanged in the response when processing of the request has completed.

- Write the request structure to the session handle using the `write`(2) system call. Multiple requests may be written to the session in a single call.
- Read the response structure back from the session handle using the `read`(2) system call. The response structure is defined as follows:

```
struct dmio_usrresp {
        uint32_t resp_id;
        int resp_error;
};
```

The *resp_id* corresponds to the *req_id* in the request. *resp_error* contains 0 if the request succeeded or an `errno`(2) value indicating why the request failed. Multiple responses may be read back in a single call. Note that responses may not be received in the same order as requests were written.

When a client is finished using a **dmoverio** session, the session is destroyed by closing the session handle using `close`(2).

EXAMPLES

The following is an example of a client using **dmoverio** to zero-fill a region of memory. In this example, the application would be able to perform other work while the hardware-assisted data mover clears the specified block of memory.

```
int
hw_bzero(void *buf, size_t len)
{
        static uint32_t reqid;

        struct dmio_setfunc dsf;
        struct iovec iov;
        struct dmio_usrreq req;
        struct dmio_usrresp resp;
        int fd;

        fd = open("/dev/dmoverio", O_RDWR, 0666);
        if (fd == -1)
                return (-1);

        strcpy(dsf.dsf_name, "zero");

        if (ioctl(fd, DMIO_SETFUNC, &dsf) == -1) {
                close(fd);
                return (-1);
        }

        iov.iov_base = buf;
        iov.iov_len = len;
```

```
        req.req_outbuf.dmbuf_iov = &iov;
        req.req_outbuf.dmbuf_iovcnt = 1;
        req.req_id = reqid++;

        if (write(fd, &req, sizeof(req)) != sizeof(req)) {
                close(fd);
                return (-1);
        }

        /* Application can do other work here. */

        if (read(fd, &resp, sizeof(resp)) != sizeof(resp)) {
                close(fd);
                return (-1);
        }

        if (resp.resp_id != req.req_id) {
                close(fd);
                return (-1);
        }

        if (resp.resp_error != 0) {
                close(fd);
                return (-1);
        }

        close(fd);
        return (0);
}
```

SEE ALSO

dmover(9)

HISTORY

The **dmoverio** device first appeared in NetBSD 2.0.

AUTHORS

The **dmoverio** device was designed and implemented by Jason R. Thorpe ⟨thorpej@wasabisystems.com⟩ and contributed by Wasabi Systems, Inc.

NAME

dmphy — Driver for Davicom DM9101 and AMD 79c873 10/100 Ethernet PHYs

SYNOPSIS

dmphy* **at mii? phy ?**

DESCRIPTION

The **dmphy** driver supports the Davicom DM9101 and AMD 79c873 10/100 Ethernet PHYs.

The AMD 79c873 is a work-alike (most likely an OEM of the core) of the Davicom part.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

dmv — DEC DMV-11 point-to-point serial communications device

SYNOPSIS

```
dmv0 at uba0 csr 167000 vector dmvrint dmvxint
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **dmv** interface provides access to a point-to-point communications device which runs at up to 56 Kb/s. DMV-11s communicate using the DEC DDCMP link layer protocol.

The host's address must be specified with an `SIOCSIFADDR` `ioctl`(2), and the destination address specified with a `SIOCSIFDSTADDR` `ioctl`(2), before the interface will transmit or receive any packets.

ROUTING

The driver places a HOST entry in the kernel routing tables for the address given in the `SIOCSIFDSTADDR` `ioctl`(2). To use the DMV as a link between local nets, the route to the remote net must be added manually with the `route`(8) command, or by the use of the routing process `routed`(8) on each end of the link.

DIAGNOSTICS

dmvprobe: can't start device.
dmvprobe: device init failed, bsel4=%o, bsel6=%o. The probe routine was unable to start the device.

dmvinit: dmv%d not running.
dmvrestart: can't start device.
dmv%d: device init failed, bsel4=%o, bsel6=%o. The initialization/restart routine was unable to start the device.

dmv%d: far end on-line. The other end of the connection has come online.

dmv%d: far end restart. The other end of the line has restarted.

dmv%d: bad control %o. A bad parameter was passed to the *dmvload* routine.

dmvxint: dmv%d bad rcv pkt addr 0x%x len 0x%x. A bad packet was received.

dmv%d: bad packet address 0x%x. An input packet was received which contained a type of address unknown to the driver.

dmvxint: dmv%d unallocated packet 0x%x. A protocol error has occurred with the board.

dmvoutput, dmv%d can't handle af%d. A packet for an unsupported address family has been sent.

dmv%d: output timeout, bsel0=%b bsel2=%b. A device timeout occurred.

Numerous other device errors may be displayed.

SEE ALSO

`dmc`(4), `inet`(4), `intro`(4)

HISTORY

The **dmv** driver appeared in 4.3BSD−Tahoe.

NAME

dmz — DMZ-32 serial terminal multiplexor

SYNOPSIS

```
dmz0 at uba? csr 0160540 vector dmzrinta dmzxinta dmzrintb dmzxintb
     dmzrintc dmzxintc
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **dmz** device provides 24 lines of asynchronous serial line support. Modem control on all ports is available as an option for the H3014 distribution panel.

An optional argument *flags* may be supplied with the device specification for **dmz** in the config file indicating that the line corresponding to bit number *i* is not properly connected, and should be treated as hardwired with carrier always present. Thus specifying `flags 0x000004` for `dmz0` would cause line `ttya2` to be treated in this way.

Normal I/O control parameters for individual lines are managed by `ioctl`(2) calls. Line speeds (there are 16 choices for the DMZ) may be initiated via `getty`(8) and `stty`(1) or may be communicated by other programs which use `ioctl`(2) such as `ifconfig`(8), see `tty`(4).

The **dmz** driver normally enables the input silos with a short timeout (30 milliseconds); this allows multiple characters to be received per interrupt during periods of high-speed input.

FILES

`/dev/tty[abcefg][0-9a-n]`

DIAGNOSTICS

dmz%d: NXM line %d. No response from the UNIBUS on a DMA transfer within a timeout period. This is often followed by a UNIBUS adapter error. This occurs most frequently when the UNIBUS is heavily loaded and when devices which hog the bus (such as RK07s) are present. It is not serious.

dmz%d: silo overflow. The character input silo overflowed before it could be serviced. This can happen if a hard error occurs when the CPU is running with elevated priority, as the system will then print a message on the console with interrupts disabled. It is not serious.

SEE ALSO

`tty`(4)

HISTORY

The **dmz** driver appeared in 4.3BSD.

BUGS

It should be possible to set the silo timeout with a configuration file option, as the value is a trade-off between efficiency and response time for flow control and character echo.

NAME

dn — DN-11 serial autocall unit interface

SYNOPSIS

```
dn0 at uba? csr 0160020 vector dnintr
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **dn** device provides an interface through a DEC DN-11 (or equivalent such as the Able Quadracall) to an auto-call unit (ACU). To place an outgoing call one forks a sub-process which opens the appropriate call unit file, `/dev/cua?` and writes the phone number on it. The parent process then opens the corresponding modem line `/dev/cul?`. When the connection has been established, the open on the modem line `/dev/cul?` will return and the process will be connected. A timer is normally used to timeout the opening of the modem line.

The codes for the phone numbers are:

0-9	number to be dialed
*	dial * (':' is a synonym)
#	dial # (';' is a synonym)
−	delay 20 milliseconds
<	end of phone number ('e' is a synonym)
=	delay for a second dial tone ('w' is a synonym)
f	force a hangup of any existing connection

The phone number to be dialed must be presented as one contiguous string.

By convention, even numbered call units are for 300 baud modem lines, while odd numbered units are for 1200 baud lines. For example, `/dev/cua0` is associated with a 300 baud modem line, `/dev/cul0`, while `/dev/cua1` is associated with a 1200 baud modem line, `/dev/cul1`. For devices such as the Quadracall which simulate multiple DN-11 units, the minor device indicates which outgoing modem to use.

FILES

`/dev/cua?` call units
`/dev/cul?` associated modem lines

DIAGNOSTICS

Two error numbers are of interest at open time.

`[EBUSY]` The dialer is in use.

`[ENXIO]` The device doesn't exist, or there's no power to it.

SEE ALSO

`tip`(1)

HISTORY

A **dn** driver appeared in Version 6 AT&T UNIX.

NAME

dnkbd — Domain keyboard

SYNOPSIS

```
dnkbd at frodo? offset ?
```

DESCRIPTION

The **dnkbd** driver provides support for the Domain keyboard found on HP Apollo 9000/4xx series workstations.

SEE ALSO

com(4), frodo(4)

NAME

dpclock — DP8573A real-time clock

SYNOPSIS

```
dpclock* at hpc0 offset ?
```

DESCRIPTION

The **dpclock** driver provides support for the DP8573A real-time clock (RTC). This device appears on SGI Personal Iris 4D/2x, 4D/3x and Indigo machines. Note that the kernel expects the RTC to run in UTC.

SEE ALSO

dsclock(4), intro(4)

HISTORY

The **dpclock** driver first appeared in NetBSD 2.0.

NAME

dpt — DPT EATA SCSI adapter driver

SYNOPSIS

```
dpt* at isa? port ? irq ? dma ?
dpt* at eisa? slot ?
dpt* at pci? dev ? function ?
scsibus* at dpt?
```

DESCRIPTION

The **dpt** driver provides support for third and fourth generation DPT SCSI controllers. All communication with the controllers is conducted via the EATA (Enhanced AT Bus Attachment) protocol.

DPT adapters examine and interpret all SCSI commands received before passing them to any underlying physical device(s). In this way, caching, RAID and other transformations are achieved while remaining transparent to the host operating system.

HARDWARE

The **dpt** driver provides support for the adapters listed below. Later models are supported by the **iop** driver.

DPT SmartCache III
DPT SmartCache IV
DPT SmartRAID III
DPT SmartRAID IV

FILES

`/dev/dpt`*u* control device for unit *u*

DIAGNOSTICS

None of these messages should be encountered under normal circumstances. It should be noted that the list below is not complete.

dpt%d: readcfg failed - see dpt(4)
: The EATA configuration data did not appear upon request. This may be caused by older firmware. Generally the solution is to power-cycle the affected machine.

dpt%d: spurious intr
: A spurious interrupt was received from the HBA.

dpt%d: bogus status (returned CCB id NNNN)
: A corrupt or incomplete status packet was received from the HBA.

SEE ALSO

`cd`(4), `ch`(4), `dpti`(4), `intro`(4), `iop`(4), `scsi`(4), `sd`(4), `st`(4)

The `sysutils/dptutil` package.

CAM committee standard CAM/89-004 - the EATA (Enhanced AT Bus Attachment) protocol.

HISTORY

The **dpt** driver first appeared in NetBSD 1.4.2.

BUGS

EATA adapters other than listed may function correctly with the **dpt** driver, however a definitive list is not available.

Older boards that do not support scatter-gather I/O or DMA are not supported.

ECC formatted disk and arrays (i.e. with a sector size of 528 bytes) do not work correctly with the PM2041 and certain firmware revisions of the PM3334.

NAME

dpti — DPT/Adaptec I2O RAID management interface

SYNOPSIS

dpti* **at iop? tid 0**

DESCRIPTION

The **dpti** driver attaches to DPT and Adaptec IOPs, and provides the pass-through command interface used by the management tools for these devices. **dptelog**, **dptmgr**, and **raidutil** version 3.12 as provided by Adaptec have been tested with this driver.

FILES

`/dev/dpti`*u* control device for unit *u*

SEE ALSO

`dpt`(4), `intro`(4), `iop`(4), `iopsp`(4), `ld`(4), `iopctl`(8)

The `sysutils/dptutil` and `sysutils/storage-manager` packages.

HISTORY

The **dpti** driver first appeared in NetBSD 1.5.3.

NAME

drm — Direct Rendering Manager (DRI kernel support)

SYNOPSIS

```
i915drm* at vga?
mach64drm* at vga?
mgadrm* at vga?
r128drm* at vga?
radeondrm* at vga?
savagedrm* at vga?
sisdrm* at vga?
tdfxdrm* at vga?
viadrm* at vga?

options DRM_DEBUG
```

DESCRIPTION

The Direct Rendering Manager is part of the Direct Rendering Infrastructure (see `http://dri.freedesktop.org/`) for supporting video acceleration (3d acceleration, mostly).

The **drm** drivers provide support for the following chipsets:

i915drm	Intel i915, i945
mach64drm	Mach64 (3D Rage Pro, Rage)
mgadrm	Matrox G[24]00, G[45]50
r128drm	ATI Rage 128
radeondrm	ATI Radeon
savagedrm	S3 Savage
sisdrm	SiS
tdfxdrm	3dfx (Voodoo)
viadrm	VIA

To make use of the driver, X(7) must be compiled with DRI support, Mesa DRI drivers must be installed, the appropriate `/dev/dri/card*` device must exist, and DRI must be enabled in the X configuration file.

Details for these steps:

1. X must be compiled with DRI support. On i386, this is usually the default.
2. Mesa (see `http://www.mesa3d.org/`) should be compiled for the netbsd-dri target, patch available at `http://issc.uj.ac.za/~yorick/drm/mesa.patch`
3. The device node must exist:

```
mkdir -p /dev/dri
mknod /dev/dri/card0 c 180 0
chgrp wheel /dev/dri/card0
chmod 0660 /dev/dri/card0
```

4. Enable DRI in the X configuration (either `xorg.conf` or `XF86Config`):

```
Section "Module"
        ...
        Load  "dri"
        Load  "GLcore"
        Load  "glx"
EndSection
Section "DRI"
```

```
        Group "wheel"
        Mode 0660
EndSection
```

Debugging output can be enabled and disabled by setting the `sysctl`(8) node *hw.dri.debug*. Additional information can be obtained from the `sysctl`(8) nodes *hw.dri*, *hw.dri.card0*, *hw.dri.card1*, etc.

SEE ALSO

`X`(7)

HISTORY

DRM was first available for Linux. Subsequently Eric Anholt ported the DRM kernel modules to FreeBSD. Erik Reid adapted the FreeBSD DRM kernel modules to NetBSD. As DRM continued to develop the NetBSD support was neglected. Tonnerre Lombard got the DRM modules working again, but DRM development once again left the NetBSD support behind. Finally Yorick Hardy took the FreeBSD DRM source and managed to get it compiling and working again on NetBSD, thanks largely to the efforts of all those mentioned above. Subsequently Matthias Drochner improved the DRM file hierarchy for NetBSD and committed the DRM kernel drivers.

The **drm** drivers appeared in NetBSD 5.0.

AUTHORS

Eric Anholt, Terry Barnaby, Erdi Chen, Michel Daenzer, Leif Delgass, Frank C. Earl, Rickard E. Faith, Jose Fonseca, Nicolai Haehnle, Jeff Hartmann, Thomas Hellstrom, Gareth Hughes, Felix Kuehling, Sung-Ching Lin, Kevin E. Martin, Jared D. McNeill, Daryll Strauss, Keith Whitwell

CAVEATS

Disable AIGLX if necessary (`xorg.conf`):

```
Section "ServerFlags"
        Option  "AIGLX" "off"
EndSection

Section "Extensions"
        Option "Composite" "Disable"
EndSection
```

In case of errors, `/dev/dri/card0` may be changed, make sure to recreate it in that case.

options DRM_DEBUG can slow DRI down a lot, disable it once **drm** works.

NAME

drum — paging device

DESCRIPTION

This file refers to the paging device in use by the system. This may actually be a subdevice of one of the disk drivers, but in a system with paging interleaved across multiple disk drives it provides an indirect driver for the multiple drives.

FILES

`/dev/drum`

HISTORY

The **drum** special file appeared in 3.0BSD.

BUGS

Reads from the drum are not allowed across the interleaving boundaries. Since these only occur every .5Mbytes or so, and since the system never allocates blocks across the boundary, this is usually not a problem.

NAME

dsclock — DS1286 real-time clock

SYNOPSIS

```
dsclock* at hpc0 offset ?
```

DESCRIPTION

The **dsclock** driver provides support for the DS1286 real-time clock (RTC). This device appears on SGI Indy and Indigo2 machines. Note that the kernel expects the RTC to run in UTC.

SEE ALSO

dpclock(4), intro(4)

HISTORY

The **dsclock** driver first appeared in NetBSD 1.6.

NAME

dtide — D.T. Software IDE interface driver

SYNOPSIS

```
dtide* at podulebus0 slot ?
atabus* at dtide? channel ?
```

DESCRIPTION

The **dtide** driver handles a D.T. Software IDE interface plugged into an Acorn expansion slot. It uses the standard NetBSD IDE controller driver, and hence can use the standard **atabus** driver.

The card provides two IDE channels. The external IDE connector is channel 0, while the internal connector is channel 1.

SEE ALSO

atabus(4), atapibus(4), podulebus(4), wd(4)

BUGS

The driver was derived by reverse-engineering the card and its RISC OS driver, so it may not handle the card entirely optimally.

NAME

dvbox — HP98730 ‘‘DaVinci’’ graphics device interface

DESCRIPTION

This driver is for the HP98730 and 98731 graphics device, also known as the DaVinci. This driver has not been tested with all possible combinations of frame buffer boards and scan boards installed in the device. The driver merely checks for the existence of the device and does minimal set up.

The DaVinci can be configured at either the ‘‘internal’’ address (frame buffer address 0x200000, control register space address 0x560000) or at an external select code less than 32. At the internal address it will be the ‘‘preferred’’ console device. The hardware installation manual describes the procedure for setting these values.

A user process communicates to the device initially by means of `ioctl`(2) calls. For the HP-UX `ioctl`(2) calls supported, refer to HP-UX manuals. The BSD calls supported are:

`GRFIOCGINFO`
: Get Graphics Info

  Get info about device, setting the entries in the *`grfinfo`* structure, as defined in ⟨`hpdev/grfioctl.h`⟩. For the standard 98730, the number of planes should be 4. The number of colors would therefore be 15, excluding black. If one 98732A frame buffer board is installed, there will still be 4 planes, with the 4 planes on the colormap board becoming overlay planes. With each additional 98732 frame buffer board 4 planes will be added up to a maximum of 32 planes total.

`GRFIOCON`
: Graphics On

  Turn graphics on by enabling CRT output. The screen will come on, displaying whatever is in the frame buffer, using whatever colormap is in place.

`GRFIOCOFF`
: Graphics Off

  Turn graphics off by disabling output to the CRT. The frame buffer contents are not affected.

`GRFIOCMAP`
: Map Device to user space

  Map in control registers and frame buffer space. Once the device file is mapped, the frame buffer structure is accessible. The structure describing the 98730 is defined in ⟨`hpdev/grf_dvreg.h`⟩.

FILES

`/dev/grf?`	BSD special file
`/dev/crt98730`	
`/dev/ocrt98730`	HP-UX *starbase* special files
`/dev/MAKEDEV.hpux`	script for creating HP-UX special files

EXAMPLES

This is a short segment of code showing how the device is opened and mapped into user process address space assuming that it is `grf0`:

```
struct dvboxfb *dvbox;
u_char *Addr, frame_buffer;
struct grfinfo gi;
```

```
int disp_fd;

disp_fd = open("/dev/grf0",1);

if (ioctl (disp_fd, GRFIOCGINFO, &gi) < 0) return -1;

(void) ioctl (disp_fd, GRFIOCON, 0);

Addr = (u_char *) 0;
if (ioctl (disp_fd, GRFIOCMAP, &Addr) < 0) {
(void) ioctl (disp_fd, GRFIOCOFF, 0);
return -1;
}
dvbox = (dvboxfb *) Addr;                  /* Control Registers   */
frame_buffer=(u_char *)Addr+gi.gd_regsize; /* Frame buffer memory */
```

DIAGNOSTICS

None under BSD. HP-UX CE.utilities must be used.

ERRORS

[ENODEV] no such device.

[EBUSY] Another process has the device open.

[EINVAL] Invalid ioctl specification.

SEE ALSO

ioctl(2), grf(4)

BUGS

Not tested for all configurations of scan board and frame buffer memory boards.

NAME

dwlpx — DEC DWLPA and DWLPB PCI adapter

SYNOPSIS

```
dwlpx* at kft?
pci* at dwlpx?
```

DESCRIPTION

The **dwlpx** driver provides support for DEC DWLPA and DWLPB PCI adapter found on the AlphaServer 8x00 Systems.

SEE ALSO

intro(4), kft(4), pci(4)

NAME

dz — DZ-11 serial multiplexer device interface

SYNOPSIS

```
dz0 at uba0 csr 0160100 vector dzrint dzxint
```

DESCRIPTION

A DZ-11 provides 8 communication lines with partial modem control, adequate for UNIX dialup use.

An optional argument *flags* may be supplied with the device specification in the config file indicating that the line corresponding to bit number *i* is not properly connected, and should be treated as hard-wired with carrier always present. Thus specifying `flags 0x04` for `dz0` would cause line `tty02` to be treated in this way.

Normal I/O control parameters for individual lines are managed by `ioctl`(2) calls. Line speeds may be initiated via the `ttys`(5) file, `stty`(1) or `ifconfig`(8) to name a few, see `tty`(4).

The **dz** driver monitors the rate of input on each board, and switches between the use of character-at-a-time interrupts and input silos. While the silo is enabled during periods of high-speed input, the driver polls for input 30 times per second.

FILES

`/dev/tty[0-9][0-9]`
`/dev/ttyd[0-9a-f]` dialups

DIAGNOSTICS

dz%d: silo overflow . The 64 character input silo overflowed before it could be serviced. This can happen if a hard error occurs when the CPU is running with elevated priority, as the system will then print a message on the console with interrupts disabled. It is not serious.

SEE ALSO

`stty`(1), `tty`(4), `ttys`(5), `getty`(8)

HISTORY

A **dz** driver appeared in Version 32V AT&T UNIX.

NAME

ea — Atomwide “Ether3” Ethernet driver

SYNOPSIS

```
ea* at podulebus0 slot ?
```

DESCRIPTION

The **ea** driver provides access to a 10 Mb/s Ethernet network through a card based on the SEEQ 8005 Advanced Ethernet Data Link Controller.

The cards supported by this driver are generally those supported by the “Ether3” driver under RISC OS. These include:

- Atomwide A-1005
- Atomwide A-1006
- Atomwide A-1007
- Acorn AEH54

Media selection on these cards is through links on the board, so there are no media options available through the driver.

SEE ALSO

eb(4), netintro(4), podulebus(4), ifconfig(8)

BUGS

The driver doesn’t support 8-bit cards like the A-1009, A-1010, A-1011 and A-1012.

NAME

eap — AudioPCI audio device driver

SYNOPSIS

```
eap*   at pci? dev ? function ?
audio* at audiobus?
joy*   at eap?
midi*  at eap?
options EAP_USE_BOTH_DACS
```

DESCRIPTION

The **eap** driver provides support for the Ensoniq AudioPCI and Creative Labs SoundBlaster PCI series of audio cards. All models based on the ES1370, ES1371, and ES1373 chips are supported.

By specifying:

options EAP_USE_BOTH_DACS

a second audio device is attached. You can use it simply by directing audio output to it. This way it is possible for two different programs to use the "same" audio device simultaneously.

SEE ALSO

ac97(4), audio(4), joy(4), midi(4), pci(4)

HISTORY

The **eap** device driver appeared in NetBSD 1.4.

CAVEATS

The joystick port hardware works by emulating a legacy isa(4) joystick port, bypassing the pci(4) bus method for address allocation. This is unlikely to work on PCI busses other than the primary one. There is also a possibility for conflicts with real ISA devices because the PCI bus is probed before ISA. Use with caution.

NAME

eb — ANT “EtherB” Ethernet driver

SYNOPSIS

eb* **at podulebus0 slot ?**

DESCRIPTION

The **eb** driver provides access to a 10 Mb/s Ethernet network through a card based on the SEEQ 80C04 Ethernet Data Link Controller.

The cards supported by this driver are generally those supported by the “EtherB” driver under RISC OS. These include:

- ANT EtherB network slot card
- Acorn AEH61

SEE ALSO

ea(4), netintro(4), podulebus(4), ifconfig(8)

NAME

EBus — introduction to SPARC EBus bus support and drivers

SYNOPSIS

```
ebus* at pci?
```

DESCRIPTION

The **EBus** bus is designed to provide the ability to put ISA and traditional Intel-style peripherals in a SPARC based system with a minimal amount of glue logic. Typically, it is implemented in the PCIO IC from SME, which also implements a hme(4) compatible PCI network device (network). The **EBus** has four DMA channels, similar to the DMA seen in the esp(4) SCSI DMA.

SEE ALSO

Sun Microelectronics, *Peripheral Component Interconnect Input Output Controller*, Part No.: 802-7837-01, March 1997, http://www.sun.com/oem/products/manuals/802-7837.pdf.

NAME

ec — driver for 3Com EtherLink II (3c503) ISA bus Ethernet cards

SYNOPSIS

```
ec0 at isa? port 0x250 iomem 0xd8000 irq 9
```

DESCRIPTION

The **ec** device driver supports 3Com EtherLink II (3c503) Ethernet cards for ISA bus which are based on the National Semiconductor DP8390/WD83C690 Ethernet interface chips.

MEDIA SELECTION

The EtherLink II supports two media types on a single card. All support the AUI media type. The other media is either BNC or UTP behind a transceiver. Software cannot differentiate between BNC and UTP cards.

To enable the AUI media, select the *10base5* or *aui* media type with `ifconfig`(8)'s **media** directive. To select the other media (BNC or UTP) , select the *10base2* or *bnc* media type.

DIAGNOSTICS

ec0: wildcarded IRQ is not allowed

The IRQ was wildcarded in the kernel configuration file. This is not supported.

ec0: invalid IRQ <n>, must be 3, 4, 5, or 9

An IRQ other than the above IRQ values was specified in the kernel configuration file. The EtherLink II hardware only supports the above listed IRQ values.

ec0: failed to clear shared memory at offset <off>

The memory test was unable to clear shared the interface's shared memory region. This often indicates that the card is configured at a conflicting *iomem* address.

ec0: warning - receiver ring buffer overrun

The DP8390 Ethernet chip used by this board implements a shared-memory ring-buffer to store incoming packets. The 3c503 usually has only 8K bytes of shared memory. This is only enough room for about 4 full-size (1500 byte) packets. This can sometimes be a problem, especially on the original 3c503, because these boards' shared-memory access speed is quite slow; typically only about 1MB/second. The overhead of this slow memory access, and the fact that there is only room for 4 full-sized packets means that the ring-buffer will occasionally overrun.

When an overrun occurs, the board must be reset to avoid a lockup problem in early revision DP8390 Ethernet chips. Resetting the board causes all of the data in the ring-buffer to be lost, requiring the data to be retransmitted/received, congesting the board further. Because of this, maximum throughput on these boards is only about 400-600K bytes per second.

This problem is exacerbated by NFS because the 8-bit boards lack sufficient packet buffer memory to support the default 8K byte packets that NFS and other protocols use as their default. If these cards must be used with NFS, use the `mount_nfs`(8) **-r** and **-w** options in `/etc/fstab` to limit NFS's packet size. 4K (4096) byte packets generally work.

SEE ALSO

`ifmedia`(4), `intro`(4), `isa`(4), `ifconfig`(8), `mount_nfs`(8)

NAME

ec — 3Com 3c400 Multibus Ethernet interface driver

SYNOPSIS

```
ec0 at mbmem0 addr 0xe0000 ipl 3
ec1 at mbmem0 addr 0xe2000 ipl 3
```

DESCRIPTION

The **ec** interface provides access to the 10 Mb/s Ethernet network via the 3Com 3c400 Ethernet chip set.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **ec** interface employs the Address Resolution Protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

SEE ALSO

`arp`(4), `ie`(4), `inet`(4), `intro`(4)

NAME

ec — 3Com 10 Mb/s Ethernet interface

SYNOPSIS

```
ec0 at uba0 csr 161000 vector ecrint eccollide ecxint flags 0
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **ec** interface provides access to a 10 Mb/s Ethernet network through a 3Com controller.

The hardware has 32 kilobytes of dual-ported memory on the UNIBUS. This memory is used for internal buffering by the board, and the interface code reads the buffer contents directly through the UNIBUS. The address of this memory is given in the *flags* field in the configuration file. The first interface normally has its memory at UNIBUS address 0.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **ec** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

The interface normally tries to use a "trailer" encapsulation to minimize copying data on input and output. The use of trailers is negotiated with ARP. This negotiation may be disabled, on a per-interface basis, by setting the `IFF_NOTRAILERS` flag with an `SIOCSIFFLAGS` `ioctl`(2).

The interface software implements an exponential backoff algorithm when notified of a collision on the cable. This algorithm uses a 16-bit mask and the VAX-11's interval timer in calculating a series of random backoff values. The algorithm is as follows:

1. Initialize the mask to be all 1's.
2. If the mask is zero, 16 retries have been made and we give up.
3. Shift the mask left one bit and formulate a backoff by masking the interval timer with the smaller of the complement of this mask and a 5-bit mask, resulting in a pseudo-random number between 0 and 31. This produces the number of slot times to delay, where a slot is 51 microseconds.
4. Use the value calculated in step 3 to delay before retransmitting the packet. The delay is done in a software busy loop.

DIAGNOSTICS

ec%d: send error. After 16 retransmissions using the exponential backoff algorithm described above, the packet was dropped.

ec%d: input error (offset=%d). The hardware indicated an error in reading a packet off the cable or an illegally sized packet. The buffer offset value is printed for debugging purposes.

ec%d: can't handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

SEE ALSO

`arp`(4), `inet`(4), `netintro`(4)

HISTORY

The **ec** driver appeared in 4.2BSD.

BUGS

The hardware is not capable of talking to itself. The software implements local sending and broadcast by sending such packets to the loop interface. This is a kludge.

Backoff delays are done in a software busy loop. This can degrade the system if the network experiences frequent collisions.

NAME

eca — Acorn Econet driver

SYNOPSIS

```
eca0 at ioc0 bank 2
```

DESCRIPTION

The **eca** driver provides access to an Econet network through a standard Acorn Econet module, based on the Motorola 6854 ADLC. It obtains its Econet station number from the machine's CMOS RAM.

DIAGNOSTICS

%s: Tx underrun Transmission of a frame failed because the driver couldn't get data to the ADLC fast enough. This might be caused by the network clock rate being too high, or by the current screen mode using too much of the data bus's bandwidth.

%s: collision A collision occurred while transmitting a frame. The transmission should be retried, but currently isn't.

%s: incomplete transmission A transmission failed to complete, but no obvious reason was found. This should never happen.

%s: Rx overrun Reception of a frame failed because the driver couldn't get data off the ADLC fast enough. Likely causes are the same as for `Tx underrun` above.

%s: Rx abort A frame being received was aborted before it was complete. This was probably caused by a problem at the transmitting station.

%s: CRC error A received frame failed its cyclic redundancy check and was discarded. This is usually due to electrical noise on the network.

%s: No clock The clock signal on the Econet has disappeared. This probably indicates that the machine is unplugged from the network, or that the network itself has failed.

%s: Oversized frame A frame larger than the current MTU of the interface was received and discarded.

SEE ALSO

`netintro`(4), `ifconfig`(8)

NAME

ed — Ethernet driver for DP8390-based Ethernet boards

SYNOPSIS

ed* **at zbus0**

DESCRIPTION

The **ed** interface provides access to a 10 Mb/s Ethernet network via the DP8390 Ethernet chip set.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **ed** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

HARDWARE

The **ed** interface supports the following Zorro II expansion cards:

AMIGANET Hydra Systems' Ethernet card, manufacturer 2121, product 1

LAN ROVER ASDG's Ethernet card, manufacturer 1023, product 254

SEE ALSO

`arp`(4), `inet`(4), `intro`(4), `ifconfig`(8)

HISTORY

The **ed** interface first appeared in NetBSD 1.0.

NAME

edc — IBM DASD Storage Interface ESDI disk driver

SYNOPSIS

```
edc* at mca? slot ?
ed* at edc? drive ?
```

DESCRIPTION

The **edc** driver supports IBM MCA ESDI disk controllers and disks, most commonly found in IBM PS/2 machines. Supported boards include:

IBM ESDI Fixed Disk Controller

IBM Integrated ESDI Fixed Disk & Controller

SEE ALSO

intro(4), mca(4)

HISTORY

This driver appeared in NetBSD 1.6.

AUTHORS

The driver was written by Jaromir Dolecek ⟨jdolecek@NetBSD.org⟩.

NOTES

The driver should also work properly on machines with more than 16MB of memory, using buffer bouncing to overcome 24bit MCA DMA limitation.

NAME

ef — 3Com EtherLink 16 (3c507) ISA Ethernet driver

SYNOPSIS

```
ef0 at isa? port 0x360 iomem 0xd0000 irq 7
```

DESCRIPTION

The **ef** driver supports 3Com EtherLink II (3c507) 10Mb/s Ethernet NIC based on the Intel 82586 Ethernet chip.

SEE ALSO

ai(4), elmc(4), ifmedia(4), intro(4), isa(4), ix(4), ifconfig(8)

http://www.3com.com/

AUTHORS

Rafal K. Boni

NAME

eg — 3Com EtherLink Plus (3c505) Ethernet interface device driver

SYNOPSIS

```
eg0 at isa? port 0x280 irq 9
```

DESCRIPTION

The **eg** interface provides access to a 10 Mb/s Ethernet network via the 3Com EtherLink Plus (3c505) Ethernet board.

SEE ALSO

intro(4), isa(4), ifconfig(8)

AUTHORS

The **eg** driver was written by Dean Huxley.

NAME

eh — i-cubed EtherLan 100/200/500-series (EtherH) Ethernet driver

SYNOPSIS

```
eh* at podulebus0 slot ?
```

DESCRIPTION

The **eh** driver provides access to a 10 Mb/s Ethernet network through an i-cubed EtherLan 100-series, 200-series or 500-series expansion card. It also supports the Acorn-badged variants of the same cards, with part numbers AEH75, AEH77 and AEH79.

Media selection on cards supporting multiple media can be accomplished either using `ifconfig`(8) or using links on the card. The links override the software and make the card behave as if only one medium is available. The media available, subject to the hardware on the card, are:

auto — Automatically select 10BASE-T or 10BASE2. 10BASE-T will be used if a link beat appears to be present on that interface. Otherwise, 10BASE2 will be used.

10baseT — 10BASE-T, 10Mb/s over unshielded twisted pair, RJ45 connector.

10base2 — 10BASE2, 10Mb/s over coaxial cable, BNC connector, also called Thinnet.

SEE ALSO

`ifmedia`(4), `ne`(4), `netintro`(4), `podulebus`(4), `ifconfig`(8)

BUGS

EtherLan 500-series support is untested.

The 10BASE-FL port on the EtherLan 513 will probably be identified as 10BASE2.

NAME

ehci — USB Enhanced Host Controller driver

SYNOPSIS

```
ehci* at cardbus? function ?
ehci* at pci? dev ? function ?
usb*  at ehci?
```

DESCRIPTION

The **ehci** driver provides support for the USB Enhanced Host Controller Interface, which is used by USB 2.0 controllers.

EHCI controllers are peculiar in that they can only handle the USB 2.0 protocol. This means that they normally have one or more companion controllers (i.e., ohci(4) or uhci(4)) handling USB 1.x devices. Consequently each USB connector is electrically connected to two USB controllers. The handling of this is totally automatic, but can be noticed since USB 1.x and USB 2.0 devices plugged in to the same connector appear to connect to different USB busses.

SEE ALSO

cardbus(4), ohci(4), pci(4), uhci(4), usb(4)

HISTORY

The **ehci** driver appeared in NetBSD 1.6.

BUGS

The support for hubs that are connected with high speed upstream and low or full speed downstream (i.e., for transaction translators) is limited.

There is no support (yet) for isochronous transfers.

NAME

ei — Acorn AKA25 (Ether1) Ethernet driver

SYNOPSIS

ei* at podulebus0 slot ?

DESCRIPTION

The **ei** driver provides access to a 10 Mb/s Ethernet network through an Acorn AKA25 Ethernet card (also known as “Ether1” after its RISC OS driver). The card is based on the Intel 82586.

Media selection on the AKA25 is through links on the board (LK3–LK8 and LK10), so there are no media options available through the driver.

SEE ALSO

netintro(4), podulebus(4), ifconfig(8)

NAME

eisa — Introduction to EISA bus machine-independent drivers and support

SYNOPSIS

options EISAVERBOSE

Machine-dependent; depends on the bus topology and EISA bus interface of your system. Typical EISA buses are either connected directly to the main system bus, or via an PCI to EISA bridge. See the intro(4) documentation for your system for details.

DESCRIPTION

NetBSD includes a machine-independent EISA bus subsystem and several machine-independent EISA device drivers.

Your system may support additional EISA devices. Drivers for EISA devices not listed here are machine-dependent. Consult your system's intro(4) for additional information.

SUPPORTED DEVICES

NetBSD includes machine-independent EISA drivers, sorted by device type and driver name:

Disk and tape controllers

cac	Compaq array controllers.
mlx	Mylex DAC960 and DEC SWXCR RAID controllers.

SCSI interfaces

ahb	Adaptec 174x SCSI interfaces.
ahc	Adaptec AIC 7770, 274x, and 284x SCSI interfaces.
bha	BusLogic BT-74x SCSI interfaces.
dpt	DPT SmartCache/SmartRAID III and IV SCSI interfaces.
uha	Ultrastor 24f SCSI interfaces.

Network interfaces

ep	3Com 3c579 and 3c592 10Mbit Ethernet, and 3c597 10/100Mbit Ethernet interfaces.
fea	Digital DEFEA FDDI interfaces.
le	Digital DE422 Ethernet interfaces.
tlp	Digital DE425 Ethernet interfaces.

Note that most or all EISA devices also have PCI or ISA equivalents. These are listed in pci(4), isa(4), or isapnp(4), respectively. The manual pages for each individual driver also lists the supported bus variants.

SEE ALSO

ahb(4), ahc(4), bha(4), cac(4), dpt(4), ep(4), fea(4), intro(4), le(4), mlx(4), tlp(4), uha(4)

HISTORY

The machine-independent EISA subsystem appeared in NetBSD 1.2.

NAME

el — 3Com EtherLink (3c501) Ethernet interface device driver

SYNOPSIS

```
el0 at isa? port 0x300 irq 9
```

DESCRIPTION

The **el** interface provides access to a 10 Mb/s Ethernet network via the 3Com EtherLink (3c501) Ethernet cards.

SEE ALSO

intro(4), isa(4), ifconfig(8)

AUTHORS

The **el** driver was written my Matthew Kimmel.

BUGS

The **el** driver does not currently support multicast or DMA.

NAME

elanpar — AMD Elan SC520 Programmable Address Regions

SYNOPSIS

```
elansc* at mainbus? bus ?
elanpar* at elansc?
```

DESCRIPTION

The **elanpar** driver supports the write-protect feature of the AMD Elan SC520 microcontroller's integrated Programmable Address Regions. Currently, **elanpar** protects the kernel text from being overwritten by the CPU or errant DMA.

DIAGNOSTICS

elanpar0: cpu violated write-protect window %u
elanpar0: gp violated write-protect window %u
elanpar0: pci violated write-protect window %u

A Programmable Address Region stopped either the CPU, the general-purpose bus (gp), or a PCI bus master from writing to the indicated window of write-protected memory.

elanpar0: %u bytes of kernel text are unprotected

elanpar has not write-protected *%u* bytes of the kernel text.

SEE ALSO

dmesg(8), elanpex(4), elansc(4), syslogd(8).

BUGS

elanpar leaves as many as 65535 bytes unprotected at the beginning and end of kernel text. Also, **elanpar** is not compatible with setting breakpoints using ddb(4). Disable **elanpar** using **drvctl -d** *elanpar0* before setting a breakpoint with ddb(4).

HISTORY

The **elanpar** device first appeared in NetBSD 5.0.

AUTHORS

The **elanpar** driver was written by David Young ⟨dyoung@NetBSD.org⟩.

NAME

elanpex — AMD Elan SC520 PCI Exception Instrumentation

SYNOPSIS

```
elansc* at mainbus? bus ?
elanpex* at elansc?
```

DESCRIPTION

The **elanpex** driver supports the PCI exception-reporting facilities of the AMD Elan SC520 microcontroller's integrated PCI host controller.

DIAGNOSTICS

PCI master exceptions

The host controller may originate a transaction of type *%s* on bus address *%x* that fails for the following reasons:

elanpex0: %s %x master retry timeout
elanpex0: %s %x master target abort
elanpex0: %s %x master abort
elanpex0: %s %x master system error
elanpex0: %s %x master received parity error
elanpex0: %s %x master detected parity error

Transaction types include

i/o read
i/o write
memory rd
memory wr
cfg rd
cfg wr

PCI target exceptions

The host controller may be the target of a failed transaction of type *%s* at bus address *%x*. Failures may occur for the following reasons:

elanpex0: %s %x target delayed txn timeout
elanpex0: %s %x target address parity
elanpex0: %s %x target data parity

Transaction types are alike to failed master exceptions.

SEE ALSO

dmesg(8), elanpar(4), elansc(4), syslogd(8).

HISTORY

The **elanpex** device first appeared in NetBSD 5.0.

AUTHORS

The **elanpex** driver was written by David Young ⟨dyoung@NetBSD.org⟩.

NAME

elansc — AMD Elan SC520 System Controller driver

SYNOPSIS

```
elansc* at mainbus? bus ?
gpio* at elansc?
pci* at elansc?
elanpar* at elansc?
elanpex* at elansc?
```

DESCRIPTION

The **elansc** driver supports the system controller of the AMD Elan SC520 microcontroller. The SC520 consists of an AMD Am5x86 processor core, integrated PCI host controller, and several standard on-chip devices, such as NS16550-compatible UARTs, real-time clock, and timers.

The Elan SC520 also provides several special on-chip devices. The following are supported by the **elansc** driver:

- Watchdog timer. The watchdog timer may be configured for a 1 second, 2 second, 4 second, 8 second, 16 second, or 32 second expiration period.
- PCI exceptions reporting. The SC520 microcontroller can report exceptions that occur as it acts as both a PCI bus master and a bus target. See elanpex(4).
- RAM write-protection. The SC520 microcontroller can designate write-protected regions of RAM using the Programmable Address Regions registers. See elanpar(4).
- Programmable Input/Output. The SC520 microcontroller supports 32 programmable I/O signals (PIOs) that can be used on the system board to monitor signals or control devices that are not handled by the other functions in the SC520 microcontroller. These signals can be programmed to be inputs or to be driven “high” or “low” as outputs. Pins can be accessed through the gpio(4) framework. The gpioctl(8) program allows easy manipulation of pins from userland.
- PCI host-bridge optimization. **elansc** takes advantage of a suspend/resume cycle to tune the PCI host-bridge for higher performance.

SEE ALSO

elanpar(4), elanpex(4), gpio(4), gpioctl(8), wdogctl(8)

HISTORY

The **elansc** device first appeared in NetBSD 2.0. PIO function support was added in OpenBSD 3.6, and subsequently ported to NetBSD 4.0. Support for PCI exceptions reporting and for RAM write-protection first appeared in NetBSD 5.0.

AUTHORS

The **elansc** driver was written by Jason R. Thorpe ⟨thorpej@NetBSD.org⟩.
Jasper Wallace provided the work-around for a hardware bug related to the watchdog timer in some steppings of the SC520 CPU. Support for the PIO function was added to OpenBSD 3.6 by
Alexander Yurchenko ⟨grange@openbsd.org⟩ and was ported to NetBSD by
Jeff Rizzo ⟨riz@NetBSD.org⟩.
David Young ⟨dyoung@NetBSD.org⟩ added support for PCI exceptions reporting and for RAM write-protection using the Programmable Address Regions.

NAME

elmc — 3Com EtherLink/MC (3c523) MCA Ethernet driver

SYNOPSIS

elmc* **at mca? slot ?**

DESCRIPTION

The **elmc** driver supports 3Com EtherLink/MC (3c523) 10Mb/s Ethernet NIC based on the Intel 82586 Ethernet chip.

SEE ALSO

ai(4), ef(4), ifmedia(4), intro(4), ix(4), mca(4), ifconfig(8)

http://www.3com.com/

AUTHORS

The driver was written by Jaromir Dolecek ⟨jdolecek@NetBSD.org⟩.

NAME

emuxki — Creative Labs SBLive! and PCI 512 audio device driver

SYNOPSIS

```
emuxki* at pci? dev ? function ?
audio*  at audiobus?
```

DESCRIPTION

The **emuxki** device driver supports Creative Sound Blaster Live! cards as well as the Sound Blaster PCI 512.

These Environmental Audio cards are based upon the programmable EMU10K1 digital-processing chip.

Hardware features:

- E-mu Systems, Inc. EMU10K1 music synthesis engine
- 64-voice hardware polyphony with E-mu's patented 8-point interpolation technology
- Up to 1024-voice polyphony with multi-timbre capability
- Support for real-time digital effects like reverb, chorus, flanger, pitch shifter, or distortion across any audio source
- User-selectable settings are optimized for headphones, two or four speakers
- Creative Multi Speaker Surround (CMSS) technology places any mono or stereo source in a 360 degree audio space
- Processes sample rates from 5kHz to 48kHz
- Hardware full duplex support enables simultaneous record and playback at 8 standard sample rates
- Allows multiple audio sources playback on the same speaker system

SEE ALSO

ac97(4), audio(4), joy(4), pci(4)

HISTORY

The **emuxki** device driver appeared in NetBSD 1.5.3.

AUTHORS

The **emuxki** driver was written by Yannick Montulet ⟨yannick.montulet@epitech.net⟩.

BUGS

Currently, this driver does not support multiple source recording, MIDI nor the multi-voice capability.

NAME

en — Midway-based ATM interfaces device driver

SYNOPSIS

```
en* at pci? dev ? function ?
en* at sbus? slot ? offset ?
```

DESCRIPTION

The **en** device driver supports Midway-based ATM interfaces including the Efficient Networks, Inc. ENI-155 and Adaptec 5900 and 5905.

OPTIONS

The Efficient Networks, Inc. and Adaptec interfaces use significantly different DMA engines. In order to reduce the size of the **en** driver object, exactly one of the following options may be placed in the kernel configuration file:

MIDWAY_ADPONLY
: Include support for Adaptec interfaces only.

MIDWAY_ENIONLY
: Include support for Efficient Networks, Inc. interfaces only.

HARDWARE

Cards supported by the **en** driver include:

Efficient Networks, Inc. ENI-155
Adaptec 5900 and 5905

SEE ALSO

`intro`(4), `pci`(4), `sbus`(4), `ifconfig`(8)

BUGS

The Sbus attachment of the **en** driver only currently functions on sun4c class systems.

NAME

en — Xerox 3 Mb/s Ethernet interface

SYNOPSIS

```
en0 at uba0 csr 161000 vector enrint enxint encollide
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **en** interface provides access to a 3 Mb/s Ethernet network. Due to limitations in the hardware, DMA transfers to and from the network must take place in the lower 64K bytes of the UNIBUS address space, and thus this must be among the first UNIBUS devices enabled after boot.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The station address is discovered by probing the on-board Ethernet address register, and is used to verify the protocol addresses. No packets will be sent or accepted until a network address is supplied.

The interface software implements an exponential backoff algorithm when notified of a collision on the cable. This algorithm uses a 16-bit mask and the VAX-11's interval timer in calculating a series of random backoff values. The algorithm is as follows:

1. Initialize the mask to be all 1's.
2. If the mask is zero, 16 retries have been made and we give up.
3. Shift the mask left one bit and formulate a backoff by masking the interval timer with the mask (this is actually the two's complement of the value).
4. Use the value calculated in step 3 to delay before retransmitting the packet.

The interface handles both Internet and NS protocol families. It normally tries to use a "trailer" encapsulation to minimize copying data on input and output. The use of trailers is negotiated with ARP. This negotiation may be disabled, on a per-interface basis, by setting the `IFF_NOTRAILERS` flag with an `SIOCSIFFLAGS` `ioctl`(2).

DIAGNOSTICS

en%d: output error. The hardware indicated an error on the previous transmission.

en%d: send error. After 16 retransmissions using the exponential backoff algorithm described above, the packet was dropped.

en%d: input error. The hardware indicated an error in reading a packet off the cable.

en%d: can't handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

SEE ALSO

`inet`(4), `netintro`(4)

HISTORY

The **en** driver appeared in 4.2BSD.

BUGS

The device has insufficient buffering to handle back to back packets. This makes use in a production environment painful.

The hardware does word at a time DMA without byte swapping. To compensate, byte swapping of user data must either be done by the user or by the system. A kludge to byte swap only IP packets is provided if the `ENF_SWABIPS` flag is defined in the driver and set at boot time with an `SIOCSIFFLAGS` `ioctl`(2).

NAME

envctrl — Sun Ultra Enterprise 450 environmental monitoring

SYNOPSIS

envctrl* **at ebus?**

DESCRIPTION

The **envctrl** driver provides thermal monitoring of processors and power supplies, as well as automatic fan regulation. The driver attaches to the SUNW,envctrl subsystem found on Ultra Enterprise 450 systems.

The following sensors can be queried with envstat(8):

Sensor	Units	Description
CPU0	degC	Processor 0 temperature
CPU1	degC	Processor 1 temperature
CPU2	degC	Processor 2 temperature
CPU3	degC	Processor 3 temperature
PS0	degC	Power supply 0 temperature
PS1	degC	Power supply 1 temperature
PS2	degC	Power supply 2 temperature
ambient	degC	Ambient temperature
cpufan voltage	V	Voltage used to drive processor fans
psfan voltage	V	Voltage used to drive power supply fans
ps failed	count	number of failed power supplies
fans failed	count	number of failed fans

SEE ALSO

envsys(4), envstat(8)

HISTORY

The **envctrl** driver first appeared in NetBSD 5.0.

AUTHORS

Tobias Nygren ⟨tnn@NetBSD.org⟩

BUGS

The driver doesn’t support SUNW,envctrltwo found on the Ultra Enterprise 250.

Environmental interrupts are not supported.

NAME

envsys — Environmental Systems framework (version 2)

SYNOPSIS

```
#include <sys/envsys.h>
```

DESCRIPTION

The **envsys** framework provides support to handle hardware monitor devices. Hardware monitoring chips are able to report values from different types of sensors.

The **envsys** framework consists of two parts:

1. the userland part, to receive the current sensor data and to set some properties on sensors: envstat(8).
2. The kernel part that is able to talk to the drivers providing sensor data: sysmon_envsys(9).

The **envsys** framework uses proplib(3) for communication between kernel and user space. The following ioctl(2) types are available:

ENVSYS_GETDICTIONARY (prop_dictionary_t)

This ioctl(2) is used to receive the global dictionary that is being used in the kernel by the sysmon_envsys(9) framework. It will contain an array of dictionaries per device and one dictionary per sensor plus another special dictionary that contains the properties for a device. Each sensor dictionary will have their own characteristics and values.

The following XML property list represents a virtual device “device0” with one sensor “sensor0” and all available properties set on it, plus another sensor for the “device-properties” dictionary (which contains specific properties for a device):

```
<key>device0</key>
<array>
        <dict>
                <key>allow-rfact</key>
                <true/>
                <key>avg-value</key>
                <integer>36400</integer>
                <key>battery-capacity</key>
                <string>NORMAL</string>
                <key>critical-capacity</key>
                <integer>21417</integer>
                <key>critical-max</key>
                <integer>343150000</integer>
                <key>critical-min</key>
                <integer>288150000</integer>
                <key>cur-value</key>
                <integer>406000</integer>
                <key>description</key>
                <string>CPU Temp</string>
                <string>index</string>
                <string>sensor0</string>
                <key>max-value</key>
                <integer>3894000</integer>
                <key>min-value</key>
                <integer>2894000</integer>
```

```
                <key>monitoring-state-critical</key>
                <true/>
                <key>monitoring-state-critover</key>
                <true/>
                <key>monitoring-state-critunder</key>
                <true/>
                <key>monitoring-state-state-changed</key>
                <true/>
                <key>monitoring-state-warnover</key>
                <true/>
                <key>monitoring-state-warnunder</key>
                <true/>
                <key>monitoring-supported</key>
                <true/>
                <key>state</key>
                <string>valid</string>
                <key>type</key>
                <string>Ampere hour</string>
                <key>want-percentage</key>
                <true/>
        </dict>
        <dict>
                <key>device-properties</key>
                <dict>
                        <key>refresh-timeout</key>
                        <integer>0xa</integer>
                </dict>
        </dict>
</array>
```

Let's explain some more about those objects:

allow-rfact — Set to *true* mean that the sensor is able to change the resistor factor, only used in Voltage sensors.

avg-value — Current average value in the sensor.

battery-capacity — Current capacity state for a battery capacity sensor.

critical-capacity — Critical capacity set previously by the `ENVSYS_SETDICTIONARY` `ioctl`(2). Only available on sensors with the *want-percentage* object enabled.

critical-max — Critical max limit set previously by the `ENVSYS_SETDICTIONARY` `ioctl`(2).

critical-min — Critical min limit set previously by the `ENVSYS_SETDICTIONARY` `ioctl`(2).

cur-value — Current value in the sensor.

description — Description of the sensor.

index — Index position of the sensor.

max-value	Current max value in the sensor.
min-value	Current min value in the sensor.
monitoring-state-critical	If true, the driver has enabled the flag to monitor a critical state.
monitoring-state-critical-over	If true, the driver has enabled the flag to monitor a critical over state.
monitoring-state-critical-under	If true, the driver has enabled the flag to monitor a critical under state.
monitoring-state-state-changed	If true, the driver has enabled the flag to monitor for state changes in a drive or Battery state sensor.
monitoring-state-warning-over	If true, the driver has enabled the flag to monitor a warning over state.
monitoring-state-warning-under	If true, the driver has enabled the flag to monitor a warning under state.
monitoring-supported	If true, critical capacity/max/min limits may be set by the ENVSYS_SETDICTIONARY ioctl(2).
state	Current state in the sensor.
type	Type of unit in the sensor.
want-percentage	If true, *max-value* and *cur-value* are valid and a percentage may be computed from them.

ENVSYS_REMOVEPROPS (prop_dictionary_t)

This ioctl(2) is used to remove all properties that are currently set via the ENVSYS_SETDICTIONARY ioctl. The values will be set to defaults, the ones that the driver uses.

Only one object is allowed on this dictionary:

```
<key>envsys-remove-props</key>
<true/>
```

It is a boolean object and must be set to *true* to be effective.

ENVSYS_SETDICTIONARY (prop_dictionary_t)

This ioctl(2) is used to send a dictionary with new properties that should be processed by the **envsys** framework. Only a set of predefined keywords are recognized by the kernel part. The following is the property list representation of a dictionary with all recognized and required keywords that a sensor understands:

```
<dict>
        <key>description</key>
        <string>cpu temp</string>
        <key>rfact</key>
        <integer>56000</integer>
        <key>critical-capacity</key>
        <integer>10</integer>
        <key>critical-max</key>
        <integer>3400</integer>
        <key>critical-min</key>
```

```
        <integer>2800</integer>
</dict>
```

Also if some properties in a device need to be changed, the “device-properties” dictionary must be used. At this moment only the “refresh-timeout” property is understood. This has the following structure:

```
<dict>
        <key>device-properties</key>
        <dict>
                <key>refresh-timeout</key>
                <integer>0xa</integer>
        </dict>
</dict>
```

A dictionary sent to the kernel with this `ioctl`(2) should have the following structure:

```
<dict>
        <key>device_name</key>
        <array>
                <dict>
                        <key>index</key>
                        <string>sensor0</string>
                        <key>description</key>
                        <string>cpu temp</string>
                        ...
                        Another property for this sensor
                        ...
                </dict>
                ...
                Another dictionary for device-properties or sensor
                ...
        </array>
        ...
        Another device as above
        ...
</dict>
```

The named device will be an array and will contain dictionaries, any dictionary needs to have the *index* object specifying the sensor that is required for the new properties.

If an unknown object was sent with the dictionary, `EINVAL` will be returned, or if the sensor does not support changing rfact (voltage sensors) or critical/capacity limits, `ENOTSUP` will be returned.

NOTES

When setting a critical max or min limit with the `ENVSYS_SETDICTIONARY` `ioctl`(2), the user must be aware that `sysmon_envsys`(9) expects to have a proper unit, so the value must be converted. Please see `sysmon_envsys`(9) for more information.

Also when setting a critical capacity limit, the formula to send a proper value to `sysmon_envsys`(9) is the following: *value = (value / 100)* $*$ *max value*. The max value is available in the sensor’s dictionary.

EXAMPLES

The following example shows how to change the description of `sensor0` in the `aiboost0` device with the `ENVSYS_SETDICTIONARY` `ioctl`(2):

```
int
main(void)
{
        prop_dictionary_t global_dict, sensor_dict;
        prop_array_t array;
        prop_object_t obj;
        int fd;

        global_dict = prop_dictionary_create();
        sensor_dict = prop_dictionary_create();
        array = prop_array_create();

        if (!prop_dictionary_set(global_dict, "aiboost0", array))
                err(EINVAL, "prop_dictionary_set global");

        obj = prop_string_create_cstring_nocopy("sensor0");
        if (obj == NULL ||
            !prop_dictionary_set(dict, "index", obj))
                err(EINVAL, "sensor index");

        prop_object_release(obj);

        /* new description */
        obj = prop_string_create_cstring_nocopy("CPU temp");
        if (obj == NULL ||
            !prop_dictionary_set(dict, "description", obj))
                err(EINVAL, "new description");

        prop_object_release(obj);

        if (!prop_array_add(array, sensor_dict))
                err(EINVAL, "prop_array_add");

        if ((fd = open(_DEV_SYSMON, O_RDWR)) == -1)
                err(EXIT_FAILURE, "open")

        /* we are done, send the dictionary */
        error = prop_dictionary_send_ioctl(global_dict,
                                           fd,
                                           ENVSYS_SETDICTIONARY);
        prop_object_release(array);
        prop_object_release(global_dict);
        (void)close(fd);
        return error;
}
```

SEE ALSO

envstat(8), powerd(8), sysmon_envsys(9)

HISTORY

The first *envsys* framework first appeared in NetBSD 1.5. The *envsys 2* framework first appeared in NetBSD 5.0.

AUTHORS

The (current) *envsys 2* framework was implemented by Juan Romero Pardines. Additional input on the design was provided by many NetBSD developers around the world.

The first *envsys* framework was implemented by Jason R. Thorpe, Tim Rightnour, and Bill Squier.

NAME

ep — driver for 3Com EtherLink III Ethernet interfaces

SYNOPSIS

```
ep0 at isa? port ? irq ?
ep* at isapnp?
ep* at eisa? slot ?
ep* at mca? slot ?
ep* at pci? dev ? function ?
ep* at pcmcia? function ?
```

DESCRIPTION

The **ep** device driver supports the 3Com EtherLink III family of Ethernet cards.

The 3c515 is an ISA 10/100 card with DMA capability. The chipset is similar to that of the 3c905, with some changes to make it work with the more limited ISA bus address space. This card is supported, although DMA is not used.

The EISA and PCI 3c59x devices use an older DMA engine which is not capable of multi-segment DMA. DMA on these devices is not used.

The 3c529 is a MCA device, and doesn't support DMA.

The PCI 3c90x devices have multi-segment DMA capability, which is not supported by the **ep** driver. To use the DMA capabilities of these cards, the ex(4) driver must be used.

The PCI 3c90xB devices are not supported by the **ep** driver, as they do not include support for programmed I/O. These devices are supported by the ex(4) driver.

The 3c575 is a CardBus device, and is supported by ex(4) driver.

MEDIA SELECTION

There are 3 main chipset classes supported by the **ep** driver. Each has their own media selection capabilities.

The first class is the "3c509" class. This includes the 3c509, 3c509B, 3c529, 3c579, 3c562, and 3c589. These devices can support 10BASE-T, 10BASE2, and 10BASE5. Available media will be displayed when the device is found by the kernel.

The second class is the "Vortex" class. This includes the 3c592, 3c579, 3c590, and 3c595. This class also includes the 3c900-TPO and 3c900-COMBO; they use the "Boomerang" chipset, but use Vortex-style media selection. These devices have many different media types varying by model. Some models have an external MII connector for connecting an external PHY. This is supported by means of the "manual" media type. Available media will be displayed when the device is found by the kernel.

The third class is the "Boomerang" class. This includes the 3c515, 3c905, and 3c574. These devices support media selection via MII. The 3c515 and 3c905 have an nsphy(4), and the 3c574 has a tqphy(4), for this purpose. See ifmedia(4) and mii(4) for more information.

HARDWARE

Supported cards include:

3c509	ISA 10Mbps card, in BNC and multiport variants
3c509B	ISA Plug-and-Play 10Mbps card, in BNC and multiport variants
3c515	ISA Plug-and-Play 10/100 card with UTP

3c529	MCA 10Mbps card, in UTP+AUI and BNC+AUI variants
3c556B	PCMCIA 56K modem-10/100Mbps Ethernet combo card with dongle
3c562	PCMCIA modem-10Mbps Ethernet combo card with dongle
3c574	PCMCIA 10/100Mbps card with dongle
3c579	EISA 10Mbps card, in UTP, BNC, and multiport variants
3c589	PCMCIA 10Mbps card with dongle
3c590	PCI 10Mbps multiport card with DMA capability
3c592	EISA 10Mbps card with DMA capability
3c595	PCI 10/100Mbps with different media options and DMA capability
3c597	EISA 10/100Mbps card with DMA capability
3c900	PCI 10Mbps card in 10BASE-T and multiport variants with DMA capability
3c905	PCI 10/100Mbps card in 10BASE-T, multiport, and fast variants with DMA capability

NOTES

EtherLink III cards have no jumpers to set the address. 3Com supplies software to set the address of the card in software. To find the card on the ISA bus, the kernel performs a complex scan operation at IO address 0x100. *Beware*! Avoid placing other cards at that address!

The 3Com configuration utilities can 'autosense' the active media and save it as default. The saved default medium is the medium that was active at the time the configuration utility was run. The **ep** driver does not yet re-autosense the active media at boot time. If the EEPROM autosense bit is set, the driver simply uses the media type sensed and saved when the configuration utility was run.

DIAGNOSTICS

ep0: reset (status: %x) the driver has encountered a FIFO underrun or overrun. The driver will reset the card and the packet will be lost. This is not fatal.

ep0: eeprom failed to come ready The EEPROM failed to come ready. This probably means the card is wedged.

ep0: 3c509 in test mode. Erase pencil mark! This means that someone has scribbled with pencil in the test area on the card. Erase the pencil mark and reboot. (This is not a joke).

SEE ALSO

`eisa`(4), `ex`(4), `ifmedia`(4), `intro`(4), `isa`(4), `isapnp`(4), `mca`(4), `mii`(4), `nsphy`(4), `pci`(4), `pcmcia`(4), `tqphy`(4), `ifconfig`(8)

NAME

epic — SMC83C170 (EPIC/100) Ethernet driver

SYNOPSIS

epic* **at pci? dev ? function ?**

DESCRIPTION

The **epic** driver supports network adapters based on the Standard Microsystems Corp. 83C170 Ethernet PCI Integrated Controller (EPIC/100).

SEE ALSO

ifmedia(4), intro(4), pci(4), ifconfig(8)

http://www.smc.com/

HISTORY

The **epic** driver appeared in NetBSD 1.4.

AUTHORS

Jason R. Thorpe

NAME

es — Ethernet driver for SMC91C90-based Ethernet boards

SYNOPSIS

es* **at zbus0**

DESCRIPTION

The **es** interface provides access to a 10 Mb/s Ethernet network via the SMC91C90 Ethernet chip set.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **es** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

HARDWARE

The **es** interface supports the following Zorro II expansion cards:

EthernetPLUS CEI A4066 EthernetPLUS Ethernet card, manufacturer 1053, product 10

SEE ALSO

`arp`(4), `inet`(4), `intro`(4), `ifconfig`(8)

HISTORY

The **es** interface first appeared in NetBSD 1.1.

NAME

esa — ESS Technology Allegro-1 / Maestro-3 family audio device driver

SYNOPSIS

```
esa*   at pci? dev ? function ?
audio* at audiobus?
options ESA_NUM_VOICES=4
```

DESCRIPTION

The **esa** driver provides support for the ESS Allegro-1 and Maestro-3 audio devices on the PCI bus. These devices are popular in laptop systems.

The Allegro-1 and the Maestro-3 are full-duplex devices that allow independent playback and recording at sample rates between 8000Hz and 48000Hz and are capable of playback at 8 and 16 bit in mono and stereo.

Setting options ESA_NUM_VOICES=4 allows for hardware mixing in the device driver. Note that due to microcode constraints, the **esa** driver is currently limited to 4 simultaneous voices.

SEE ALSO

ac97(4), audio(4), pci(4)

HISTORY

The **esa** device driver appeared in NetBSD 1.5.3.

AUTHORS

Jared D. McNeill ⟨jmcneill@invisible.ca⟩

NAME

esh — RoadRunner-based HIPPI interfaces device driver

SYNOPSIS

esh* **at pci? dev ? function ?**

DESCRIPTION

The **esh** device driver supports the Essential Communications RoadRunner-based HIPPI interfaces. With some modifications, the driver could be made to support the Gigabit Ethernet card based on the same chip.

The driver supports both a normal network interface and a raw HIPPI Framing Protocol (HIPPI-FP) device. The HIPPI-FP interface is accessed via the `/dev/esh`**N**/**ulpN** set of devices. There are 255 available Upper Layer Protocols in FP; these are selectable via the various device entries.

HIPPI is an 800-megabit/sec networking technology which supports extremely large packet sizes. In order to efficiently use this network, the kernel should be configured with extra mbufs, and the default socket buffer size should be increased to at least 192KB, regardless of the expected bandwidth-delay product of the network.

HIPPI ARP is not widely used, and the NetBSD stack does not support it (yet). In order to define the mappings between IP addresses and ifields (the HIPPI MAC addresses), the administrator must make link-layer entries in the routing table using the `route`(8) command:

```
route add -interface 129.99.154.101 -llinfo -link esh0:3.0.0.65
```

MEDIA SELECTION

Media selection is not yet supported for this device.

SEE ALSO

`intro`(4), `pci`(4), `eshconfig`(8), `ifconfig`(8), `route`(8)

BUGS

The card must be tuned for proper and efficient DMA operation. The appropriate values vary based on the system. The `eshconfig`(8) program is used for this.

NAME

esiop — Enhanced Symbios Logic/NCR 53c8xx SCSI driver

SYNOPSIS

```
esiop* at pci? dev ? function ?
options SIOP_SYMLED
scsibus* at esiop?
```

DESCRIPTION

The **esiop** driver provides improved support over siop(4) for the Symbios Logic/NCR 53x8xx series of SCSI controller chips:

- 53c825 and 53c825a (Fast-Wide SCSI)
- 53c875 and 53c875j (Ultra-Wide SCSI)
- 53c876 (Dual Ultra-Wide SCSI)
- 53c885 (Ultra-Wide SCSI and Ethernet)
- 53c895 (Ultra2-Wide SCSI)
- 53c896 (PCI 64bit, dual Ultra2-Wide SCSI)
- 53c1010-33 (PCI 64bit, dual Ultra160 SCSI)
- 53c1510d (PCI 64bit, dual Ultra2-wide SCSI)

Older adapters are supported by the siop(4) driver.

The SIOP_SYMLED option causes the driver to report SCSI activity on the GPIO pin 1, which is connected to the activity LED on some adapters. At this time only the 53c895 based Symbios and Tekram adapters are known to require this.

SEE ALSO

cd(4), ch(4), intro(4), pci(4), scsi(4), sd(4), siop(4), st(4), uk(4)

HISTORY

The **esiop** driver first appeared in NetBSD 1.6.

AUTHORS

The **esiop** driver was written by Manuel Bouyer ⟨Manuel.Bouyer@lip6.fr⟩ for NetBSD.

NAME

esis — End System to Intermediate System Routing Protocol

SYNOPSIS

```
#include <sys/types.h>
#include <netiso/esis.h>
```

DESCRIPTION

The ES-IS routing protocol is used to dynamically map between ISO NSAP addresses and ISO SNPA addresses; to permit End Systems (ES) and Intermediate Systems (IS) to learn of each other's existence; and to allow Intermediate Systems to inform End Systems of (potentially) better routes to use when forwarding Network Protocol Data Units (NPDUs) to a particular destination.

The mapping between NSAP addresses and SNPA addresses is accomplished by transmitting "hello" Protocol Data Units (PDUs) between the cooperating Systems. These PDUs are transmitted whenever the *configuration* timer expires. When a "hello" PDU is received, the SNPA address that it conveys is stored in the routing table for as long as the *holding time* in the PDU suggests. The default *holding time* (120 seconds) placed in the "hello" PDU, the configuration timer value, and the system type (End System or Intermediate System) may be changed by issuing an `SIOCSSTYPE` `ioctl`(2), which is defined in ⟨`sys/netiso/iso_snpac.h`⟩.

The protocol behaves differently depending on whether the System is configured as an End System or an Intermediate System.

END SYSTEM OPERATION

When an interface requests a mapping for an address not in the cache, the SNPA of any known Intermediate System is returned. If an Intermediate System is not known, then the *all end systems* multicast address is returned. It is assumed that the intended recipient of the NPDU will immediately transmit a "hello" PDU back to the originator of the NPDU.

If an NPDU is forwarded by the End System, a redirect PDU will not be generated. However, redirect PDUs received will be processed. This processing consists of adding an entry in the routing table. If the redirect is towards an Intermediate System, then an entry is made in the routing table as well. The entry in the routing table will mark the NSAP address contained in the redirect PDU as the gateway for the destination system (if an NET is supplied), or will create a route with the NSAP address as the destination and the SNPA address (embodied as a link-level struct sockaddr) as the gateway.

If the System is configured as an End System, it will report all the NSAPs that have been configured using the `ifconfig`(8) command, and no others. It is possible to have more than one NSAP assigned to a given interface, and it is also possible to have the same NSAP assigned to multiple interfaces. However, any NSAP containing an NSEL that is consistent with the *nsellength* option (default one) of any interface will be accepted as an NSAP for this System.

INTERMEDIATE SYSTEM OPERATION

When an interface requests a mapping for an address not in the routing table, an error is returned.

When an NPDU is forwarded out on the same interface that the NPDU arrived upon, a redirect PDU is generated.

MANUAL ROUTING TABLE MODIFICATION

To facilitate communications with systems which do not use ES-IS, one may add a route whose destination is a struct sockaddr_iso containing the NSAP in question, and the gateway being a link-level struct sockaddr, either by writing a special purpose program, or using the `route`(8) command e.g.:

```
route add -iface -osi 49.0.4.8.0.2b.b.83.bf  -link qe0:8.0.2b.b.83.bf
```

If the System is configured as an End System and has a single network interface which does not support multicast reception, it is necessary to manually configure the location of an IS, using the route command in a similar way. There, the destination address should be "default" (spelled out literally as 7 ASCII characters), and the gateway should be once again be a link-level struct sockaddr specifying the SNPA of the IS.

SEE ALSO

`iso`(4), `ifconfig`(8), `route`(8)

End system to Intermediate system routing exchange protocol for use in conjunction with the Protocol for providing the connectionless-mode network service, ISO, 9542.

BUGS

Redirect PDUs do not contain options from the forwarded NPDU which generated the redirect. The multicast address used on the IEEE 802.3 (Ethernet) network is taken from the National Bureau of Standards (NBS) December 1987 agreements. This multicast address is not compatible with the IEEE 802.5 (Token Ring) multicast addresses format. Therefore, broadcast addresses are used on the IEEE 802.5 subnetwork.

Researchers at the University of Wisconsin are constructing an implementation of the IS-IS routing protocol.

NBS is now known as the National Institute for Standards and Technology (NIST).

NAME

esl — ESS Technology AudioDrive (Programmed I/O mode) family audio device driver

SYNOPSIS

```
esl*   at pcmcia? function ?
audio* at audiobus?
opl*   at esl?
```

DESCRIPTION

The **esl** driver provides support for the ESS 1688 AudioDrive audio devices on a PCMCIA card.

Since there is no way to do DMA over the PCMCIA bus, the **esl** driver uses a Programmed I/O mode to drive the cards. This is very inefficient, but is the only way to drive such a card.

Cards supported by this driver are capable of 8- and 16-bit audio sample playback at rates up to 44.1kHz.

SEE ALSO

audio(4), midi(4), opl(4), pcmcia(4)

HISTORY

The **esl** device driver appeared in NetBSD 1.6.

AUTHORS

Jared D. McNeill ⟨jmcneill@invisible.ca⟩

BUGS

Putting a machine into suspend mode while this driver is active will cause the machine to freeze.

Recording is not yet supported by this driver.

NAME

esm — ESS Technology Maestro 2/2E PCI Audio Accelerator audio device driver

SYNOPSIS

```
esm*   at pci? dev ? function ?
audio* at audiobus?
```

DESCRIPTION

The **esm** device driver supports sound cards based on ESS Technology Maestro, Maestro-2, and Maestro-2E PCI AC97 Audio Accelerator chips (ES1968 and ES1978). Due to their power management capabilities, the Maestro chips are often used in laptop and notebook systems.

The Maestro is a full-duplex device that allows independent playback and recording at sample rates between 4000 Hz and 48 kHz. The sound processor is capable of handling and converting 8 and 16 bit mono and stereo data.

SEE ALSO

ac97(4), audio(4), joy(4), mpu(4), pci(4)

HISTORY

The **esm** device driver appeared in NetBSD 1.6.

BUGS

Currently, this driver only supports 16-bit mono recording; an unknown bug causes stereo recording to miss a channel. Also not supported yet are hardware volume settings, MIDI, and joystick input.

NAME

eso — ESS Technology Solo-1 PCI AudioDrive audio device driver

SYNOPSIS

```
eso*   at pci? dev ? function ?
audio* at audiobus?
joy*   at eso?
mpu*   at eso?
opl*   at eso?
```

DESCRIPTION

The **eso** device driver supports sound cards based on ESS Technology Solo-1 PCI AudioDrive chips (ES1938 and ES1946), e.g. the TerraTec 128i PCI card.

DIAGNOSTICS

eso0: can't map Audio 1 DMA into I/O space PCI autoconfiguration did not map the first audio channel DMA controller into I/O space at a suitable address, and the driver was not able map it by itself. In that case only playback functionality will be available.

eso0: reset timeout The device failed to acknowledge being reset.

eso0: RDR timeout The driver timed out reading data from a controller register.

eso0: WDR timeout The driver timed out waiting to send a command or writing to a controller register.

SEE ALSO

audio(4), joy(4), mpu(4), opl(4), pci(4)

HISTORY

The **eso** device driver appeared in NetBSD 1.5. The on-board gameport was first supported in NetBSD 1.6.

NAME

esp — NCR 53C9x, Emulex ESP406, and Qlogic FAS408 SCSI driver

SYNOPSIS

ISA bus

```
esp0 at isa? port 0x230 irq ?
```

PCMCIA

```
esp* at pcmcia? function ?
```

MCA

```
esp* at mca? slot ?
```

mac68k

```
esp0 at obio?
esp1 at obio?
```

macppc

```
esp0 at obio0 flags 0x00ff
```

sun3x

```
esp0 at obio0 addr 0x66000000 ipl 2 flags 0xff0f
```

sparc

```
dma0 at obio0 addr 0xfa001000 level 4 ( Sun 4/300 )
esp0 at obio0 addr 0xfa000000 level 4 ( Sun 4/300 )

dma0 at sbus0 slot ? offset ? ( sun4c and sun4m )
esp0 at sbus0 slot ? offset ? ( sun4c )
esp0 at dma0 ( sun4m )

dma* at sbus? slot ? offset ? ( Sbus )
esp* at sbus? slot ? offset ? ( SBus, older PROMs )
esp* at dma? ( SBus )

scsibus* at esp?
```

DESCRIPTION

The **esp** driver provides support for the NCR 53C90, 53C94 and 53C96; Emulex ESP100, ESP100A, ESP200 and ESP406; and Qlogic FAS216 and FAS408 SCSI controller chips found in a wide variety of systems and peripheral boards. This includes the Qlogic ISA and VLB SCSI host adapters, and the Sun Fast SCSI buffered Ethernet for Sbus (FSBE/S, X1053A, Sun part # 501-2015).

For Qlogic PCI SCSI host adapters, use the `isp`(4) device.

CONFIGURATION

The **esp** driver supports the following **flags** for use in `config`(1) files:

bits 0-7:	disable disconnect/reselect for the corresponding SCSI target
bits 8-15:	disable synchronous negotiation for the corresponding SCSI target
bits 16-23:	disable tagged queuing for the corresponding SCSI target

"Target" is synonymous with SCSI ID number.

Note that SCSI tape drives should be allowed to perform disconnect/reselect or performance will suffer.

SEE ALSO

cd(4), ch(4), intro(4), le(4), mca(4), pcmcia(4), scsi(4), sd(4), ss(4), st(4), uk(4)

http://www.qlc.com/
http://www.sun.com/

NAME

ess — ESS Technology AudioDrive family audio device driver

SYNOPSIS

```
ess*   at isapnp?
ess*   at pnpbios? index ?
ess*   at ofisa?
audio* at audiobus?
opl*   at ess?
```

DESCRIPTION

The **ess** driver provides support for the ESS 1788, 1888, 1887, and 888 AudioDrive audio devices.

The AudioDrive 1788 is a half-duplex device, while the 1888, 1887, and 888 are full-duplex. All are capable of 8- and 16-bit audio sample recording and playback at rates up to 44.1kHz.

The AudioDrive takes 16 I/O ports. The I/O port range, IRQ, and DRQ channels are set by the driver to the values specified in the configuration file (or for isapnp, pnpbios, or ofisa, the values assigned from the firmware). The I/O port base must be one of 0x220, 0x230, 0x240, 0x250. The IRQ must be one of 5, 7, 9, 10 (or 15 on the 1887 only). The first DRQ channel must be selected from 0, 1, 3. The second DRQ channel (used for playback by the full-duplex 1888/1887, ignored by the 1788) can additionally be set to 5. If both DRQ channels are used they must be different.

The joystick interface (if enabled) is handled by the `joy`(4) driver.

SEE ALSO

`audio`(4), `isapnp`(4), `joy`(4), `ofisa`(4), `opl`(4), `pnpbios`(4)

HISTORY

The **ess** device driver appeared in NetBSD 1.4.

BUGS

The AudioDrive devices have a SoundBlaster compatibility mode, and may be detected by the SoundBlaster driver (see `sb`(4)) rather than the AudioDrive driver. The workaround is to remove the SoundBlaster driver from the kernel configuration.

NAME

et — driver for ET4000 VME graphics cards

SYNOPSIS

```
et0 at vme0
```

DESCRIPTION

The **et** device driver supports VME graphics cards based on the Tseng ET4000 chipset. It supports the minimal ioctl's needed to run X11. The 8Kb control registers are mapped from offset 0x0 to offset 0x1fff and the 1MB frame buffer is mapped from offset 0x400000 to offset 0x4fffff. VGA compatible memory is mapped from offset 0xa0000 to offset 0xc0000.

HARDWARE

Cards supported by the **et** driver include:

Crazy Dots II VME graphics card

NOTES

The clock generator on the Crazy Dots card is an ICS 1394M. Setting the clock is done by writing to a register at offset 0x0. This address cannot be read. The values that can be written to the register and the associated clock are:

Value	*Clock*
0x00	14318
0x10	16257
0x17	20000
0x0c	24000
0x04	25175
0x14	28332
0x18	32514
0x1a	36000
0x1c	40000
0x03	44900
0x0f	50000
0x13	50344
0x07	56644
0x11	65028
0x1f	80000

FILES

`/dev/etvme`

AUTHORS

Julian Coleman

BUGS

The driver does not support a text mode, so cannot be used as a console.

NAME

etherip — EtherIP tunneling device

SYNOPSIS

```
pseudo-device etherip
```

DESCRIPTION

The **etherip** interface is a tunneling pseudo device for Ethernet frames. It can tunnel Ethernet traffic over IPv4 and IPv6 using the EtherIP protocol specified in RFC 3378.

The only difference between an **etherip** interface and a real Ethernet interface is that there is an IP tunnel instead of a wire. Therefore, to use **etherip** the administrator must first create the interface and then configure protocol and addresses used for the outer header. This can be done by using `ifconfig`(8) **create** and **tunnel** subcommands, or `SIOCIFCREATE` and `SIOCSLIFPHYADDR` ioctls.

Packet format

Ethernet frames are prepended with a EtherIP header as described by RFC 3378. The resulting EtherIP packets will be encapsulated in an outer packet, which may be either an IPv4 or IPv6 packet, with IP protocol number 97.

Ethernet address

When a **etherip** device is created, it is assigned an Ethernet address of the form f2:0b:a5:xx:xx:xx. This address can later be changed through a sysctl node.

The sysctl node is net.link.etherip.<iface>. Any string of six colon-separated hexadecimal numbers will be accepted. Reading that node will provide a string representation of the current Ethernet address.

Security

The EtherIP header of incoming packets is not checked for validity. This is because there seems to be some confusion about how such a header has to look like. For outgoing packets, the header is set up the same way as done in OpenBSD, FreeBSD, and Linux to be compatible with those systems.

Converting from previous implementation

A tunnel configured for the previous (undocumented) implementation will work with just renaming the device from gif to **etherip**.

SEE ALSO

`bridge`(4), `gif`(4), `inet`(4), `inet6`(4), `tap`(4), `ifconfig`(8)

HISTORY

The **etherip** device first appeared in NetBSD 4.0, it is based on `tap`(4), `gif`(4), and the former gif-based EtherIP implementation ported from OpenBSD.

BUGS

Probably many. There is lots of code duplication between **etherip**, `tap`(4), `gif`(4), and probably other tunnelling drivers which should be cleaned up.

NAME

ex — driver for 3Com Fast EtherLink XL (3c900, 3c905, 3c980) and similar PCI bus and cardbus Ethernet interfaces

SYNOPSIS

```
ex* at cardbus? function ?
ex* at pci? dev ? function ?
```

DESCRIPTION

3Com Ethernet and Fast Ethernet cards supported by the **ex** driver include:

3c450-TX	10/100 Ethernet
3c555	MiniPCI 10/100 Ethernet
3c575-TX	Ethernet
3c575B-TX	Ethernet
3c575CT	Ethernet
3c656	MiniPCI 10/100 Ethernet
3c656B	MiniPCI 10/100 Ethernet
3c656C	MiniPCI 10/100 Ethernet
3c900-TPO	Ethernet
3c900-COMBO	Ethernet
3c900B-TPC	Ethernet
3c900B-TPO	Ethernet
3c900B-COMBO	Ethernet
3c905-T4	10/100 Ethernet
3c905-TX	10/100 Ethernet
3c905B-COMBO	10/100 Ethernet
3c905B-FX	10/100 Ethernet
3c905B-T4	10/100 Ethernet
3c905B-TX	10/100 Ethernet
3c905CX-TX	10/100 Ethernet
3c980	Server Adapter 10/100 Ethernet
3c980C-TXM	10/100 Ethernet
3cSOHO100-TX	10/100 Ethernet

All versions of the EtherLink XL (except the older 3c900 and 3c905) support IPv4/TCP/UDP checksumming in hardware. The **ex** driver supports this feature of the chip. See `ifconfig`(8) for information on how to enable this feature.

MEDIA SELECTION

Some of these network interfaces support the Media Independent Interface (MII), a bus which can have at least one arbitrary Physical interface (PHY) chip on it. NetBSD supports MII and has separate drivers for many different PHY chips, including `ukphy`(4), a generic PHY driver that can support many PHY chips that NetBSD does not yet have a specific driver for.

Support for the PHY found on a given NIC must be configured into a NetBSD kernel `config`(1) for this driver to work properly in those cases.

See `ifmedia`(4), and `mii`(4).

DIAGNOSTICS

%s: adapter failure (%x)
%s: can’t allocate download descriptors, error = %d
%s: can’t allocate or map rx buffers
%s: can’t allocate upload descriptors, error = %d
%s: can’t create download desc. DMA map, error = %d
%s: can’t create rx DMA map %d, error = %d
%s: can’t create tx DMA map %d, error = %d
%s: can’t create upload desc. DMA map, error = %d
%s: can’t load download desc. DMA map, error = %d
%s: can’t load mbuf chain, error = %d
%s: can’t load rx buffer, error = %d
%s: can’t load upload desc. DMA map, error = %d
%s: can’t map download descriptors, error = %d
%s: can’t map upload descriptors, error = %d
%s: fifo underrun (%x) @%d
%s: jabber (%x)
%s: receive stalled
%s: too many segments,
%s: uplistptr was 0 host too slow to serve incoming packets

SEE ALSO

`cardbus`(4), `exphy`(4), `ifmedia`(4), `intro`(4), `mii`(4), `pci`(4), `ifconfig`(8)

`http://www.3com.com/`

NAME

ex — Excelan 10 Mb/s Ethernet interface

SYNOPSIS

```
ex0 at uba0 csr 164000 vector excdint
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **ex** interface provides access to a 10 Mb/s Ethernet network through an Excelan controller used as a link-layer interface.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **ex** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

The interface normally tries to use a "trailer" encapsulation to minimize copying data on input and output. The use of trailers is negotiated with ARP. This negotiation may be disabled, on a per-interface basis, by setting the `IFF_NOTRAILERS` flag with an `SIOCSIFFLAGS` `ioctl`(2).

DIAGNOSTICS

ex%d: HW %c.%c, NX %c.%c, hardware address %s. This provides firmware revisions levels, and is expected during autoconfiguration.

ex%d: can't initialize. There was a failure in allocating UNIBUS resources for the device.

ex%d: configuration failed; cc = %x. The hardware indicated an error when trying to initialize itself. The error code returned is described at length in the device Reference Manual.

ex%d: receive error %b. The hardware indicated an error in reading a packet from the cable. Specific Error bits are provided

ex%d: transmit error %b. The hardware indicated an error in transmitting a packet to the cable or an illegally sized packet. Specific Error bits are provided

ex%d: can't handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

SEE ALSO

`arp`(4), `inet`(4), `netintro`(4)

HISTORY

The **ex** driver appeared in 4.3BSD.

NAME

exphy — Driver for 3Com 3c905B-TX internal Ethernet PHY

SYNOPSIS

exphy* **at mii? phy ?**

DESCRIPTION

The **exphy** driver supports the internal PHY on 3Com 3c905B-TX 10/100 Ethernet interfaces.

SEE ALSO

ex(4), ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

faith — IPv6-to-IPv4 TCP relay capturing interface

SYNOPSIS

```
pseudo-device faith
```

DESCRIPTION

The **faith** interface captures IPv6 TCP traffic, for implementing userland IPv6-to-IPv4 TCP relay like faithd(8).

faith interfaces are dynamically created and destroyed with the ifconfig(8) **create** and **destroy** subcommands.

Special action will be taken when IPv6 TCP traffic is seen on a router, and routing table suggests to route it to **faith** interface. In this case, the packet will be accepted by the router, regardless of list of IPv6 interface addresses assigned to the router. The packet will be captured by an IPv6 TCP socket, if it has IN6P_FAITH flag turned on and it has matching address/port pairs. In result, **faith** will let you capture IPv6 TCP traffic to some specific destination addresses. Userland programs, such as faithd(8) can use this behavior to relay IPv6 TCP traffic to IPv4 TCP traffic. The program can accept some specific IPv6 TCP traffic, perform getsockname(2) to get the IPv6 destination address specified by the client, and perform application-specific address mapping to relay IPv6 TCP to IPv4 TCP.

IN6P_FAITH flag on IPv6 TCP socket can be set by using setsockopt(2), with level equals to IPPROTO_IPV6 and optname equals to IPV6_FAITH.

To handle error reports by ICMPv6, some of ICMPv6 packets routed to **faith** interface will be delivered to IPv6 TCP, as well.

To understand how **faith** can be used, take a look at source code of faithd(8).

As **faith** interface implements potentially dangerous operation, great care must be taken when configuring **faith** interface. To avoid possible misuse, sysctl(8) variable net.inet6.ip6.keepfaith must be set to 1 prior to the use of the interface. When net.inet6.ip6.keepfaith is 0, no packet will be captured by **faith** interface.

faith interface is intended to be used on routers, not on hosts.

SEE ALSO

inet(4), inet6(4), faithd(8)

Jun-ichiro itojun Hagino and Kazu Yamamoto, "An IPv6-to-IPv4 transport relay translator", *RFC 3142*, June 2001, ftp://ftp.isi.edu/in-notes/rfc3142.txt.

HISTORY

The FAITH IPv6-to-IPv4 TCP relay translator was first appeared in WIDE hydrangea IPv6 stack.

NAME

Fast IPsec — hardware-accelerated IP Security Protocols

SYNOPSIS

```
options FAST_IPSEC
options IPSEC_NAT_T
pseudo-device crypto
```

DESCRIPTION

IPsec is a set of protocols, ESP (for Encapsulating Security Payload) AH (for Authentication Header), and IPComp (for IP Payload Compression Protocol) that provide security services for IP datagrams. **Fast IPsec** is an implementation of these protocols that uses the opencrypto(9) subsystem to carry out cryptographic operations. This means, in particular, that cryptographic hardware devices are employed whenever possible to optimize the performance of these protocols.

In general, the **Fast IPsec** implementation is intended to be compatible with the KAME IPsec implementation. This documentation concentrates on differences from that software. The user should refer to ipsec(4) for basic information on setting up and using these protocols.

System configuration requires the opencrypto(9) subsystem. When the **Fast IPsec** protocols are configured for use, all protocols are included in the system. To selectively enable/disable protocols, use sysctl(8).

DIAGNOSTICS

To be added.

SEE ALSO

ipsec(4), setkey(8), sysctl(8), opencrypto(9)

HISTORY

The protocols draw heavily on the OpenBSD implementation of the IPsec protocols. The policy management code is derived from the KAME implementation found in their IPsec protocols. The **Fast IPsec** protocols are based on code which appeared in FreeBSD 4.7. The NetBSD version is a close copy of the FreeBSD original, and first appeared in NetBSD 2.0.

Support for IPv6 and IPcomp protocols has been added in NetBSD 4.0.

Support for IPSEC_NAT_T (Network Address Translator Traversal as described in RFCs 3947 and 3948) has been added in NetBSD 5.0.

BUGS

There still are some issues in the IPv6 support. In particular FAST_IPSEC does not protect packets with IPv6 extension headers.

Certain legacy authentication algorithms are not supported because of issues with the opencrypto(9) subsystem.

This documentation is incomplete.

NAME

fd, **stdin**, **stdout**, **stderr** — file descriptor files

DESCRIPTION

The files /dev/fd/0 through /dev/fd/# refer to file descriptors which can be accessed through the file system. If the file descriptor is open and the mode the file is being opened with is a subset of the mode of the existing descriptor, the call:

```
fd = open("/dev/fd/0", mode);
```

and the call:

```
fd = fcntl(0, F_DUPFD, 0);
```

are equivalent.

Opening the files /dev/stdin, /dev/stdout and /dev/stderr is equivalent to the following calls:

```
fd = fcntl(STDIN_FILENO,  F_DUPFD, 0);
fd = fcntl(STDOUT_FILENO, F_DUPFD, 0);
fd = fcntl(STDERR_FILENO, F_DUPFD, 0);
```

Flags to the open(2) call other than O_RDONLY, O_WRONLY and O_RDWR are ignored.

FILES

/dev/fd/#
/dev/stdin
/dev/stdout
/dev/stderr

SEE ALSO

tty(4)

NAME

fd, **fdc** — Sun SPARCstation i82072 or i82077 floppy disk controller driver

SYNOPSIS

```
fdc0 at mainbus0 ( sun4c )
fdc0 at obio0 ( sun4m )
fd* at fdc0
```

DESCRIPTION

This is the driver for the built-in floppy disk drive run by the Intel i82072 or i82077 controller chip found on the SPARCstation desktop systems, and other SPARC systems.

Bits [0-3] of the minor device number of the special files referring to this device encode the floppy density as follows:

0 3.5'' 1.44MB floppy diskettes.

1 3.5'' 720KB floppy diskettes.

2 3.5'' 360KB floppy diskettes.

3 3.5'' 1.2MB/NEC Japanese format floppy diskettes.

FORMATTING

The driver supports floppy disk formatting using the interfaces in ⟨`sys/fdio.h`⟩:

`FDIOCGETFORMAT` *`struct fdformat_parms`*
Fetch current formatting parameters. This gets the default parameters for the open device if no parameters have been set during the session.

`FDIOCSETFORMAT` *`struct fdformat_parms`*
Set formatting parameters. The driver saves this state and it persists while the device is open.

`FDIOCFORMAT_TRACK` *`struct fdformat_cmd`*
Format a track on the medium. If this call returns `EINVAL`, the track formatting parameters were out of range for the medium. If it returns `EIO`, there was a medium error while formatting the track.

`FDIOCSETOPTS` *`int`* Set driver options which persist until the device is closed. The options should be the logical OR of the desired values below:

`FDOPT_NORETRY` Do not retry operations on failure
`FDOPT_SILENT` Do not print error messages to the console

`FDIOCGETOPTS` *`int`* Fetch drive options.

A typical use of the formatting facilities would be to open the device, call `FDIOCGETFORMAT` to fetch the current format parameters, perhaps change a parameter or two, display the formatting details to the user, and then call `FDIOCSETFORMAT` followed by a series of calls to `FDIOCFORMAT_TRACK`.

SEE ALSO

`eject`(1), `fdformat`(1)

HISTORY

The **fd** formatting support appeared in NetBSD 1.3.

NAME

fd — Sun 3/80 i82027 floppy disk drive controller driver

SYNOPSIS

```
fdc0 at obio0 ( sun3x )
fd* at fdc0
```

DESCRIPTION

The **fd** driver is for the built-in floppy diskette drive run by the Intel i82027 controller found on the Sun 3/80.

Bits [0-3] of the minor device number of the special files referring to this device encode the floppy density as follows:

0 3.5'' 1.44MB floppy diskettes.

1 3.5'' 720KB floppy diskettes.

2 3.5'' 360KB floppy diskettes.

3 3.5'' 1.2MB/NEC Japanese format floppy diskettes.

FORMATTING

The driver supports floppy disk formatting using the interfaces in ⟨`sys/fdio.h`⟩:

`FDIOCGETFORMAT` *`struct fdformat_parms`*
: Fetch current formatting parameters. This gets the default parameters for the open device if no parameters have been set during the session.

`FDIOCSETFORMAT` *`struct fdformat_parms`*
: Set formatting parameters. The driver saves this state and it persists while the device is open.

`FDIOCFORMAT_TRACK` *`struct fdformat_cmd`*
: Format a track on the medium. If this call returns `EINVAL`, the track formatting parameters were out of range for the medium. If it returns `EIO`, there was a medium error while formatting the track.

`FDIOCSETOPTS` *`int`* Set driver options which persist until the device is closed. The options should be the logical OR of the desired values below:

`FDOPT_NORETRY`	Do not retry operations on failure
`FDOPT_SILENT`	Do not print error messages to the console

`FDIOCGETOPTS` *`int`* Fetch drive options.

A typical use of the formatting facilities would be to open the device, call `FDIOCGETFORMAT` to fetch the current format parameters, perhaps change a parameter or two, display the formatting details to the user, and then call `FDIOCSETFORMAT` followed by a series of calls to `FDIOCFORMAT_TRACK`.

SEE ALSO

`eject`(1), `fdformat`(1)

HISTORY

The **fd** formatting support appeared in NetBSD 1.3.

BUGS

Formatting appears to not work reliably on all machines.

NAME

fdc — Amiga custom chip floppy disk controller device

SYNOPSIS

```
fdc0 at mainbus0
fd* at fdc? unit ?
```

DESCRIPTION

The **fdc** device driver provides support for the mainboard floppy disk controller and floppy disk drives on Amiga systems.

The driver supports the following floppy diskette formats by using particular partitions:

880KB 3.5-inch 11 sectors/track (‘‘Amiga’’) (a)
1.76MB 3.5-inch 22 sectors/track (‘‘Amiga’’) (a)
880KB 5.25-inch 11 sectors/track (‘‘Amiga’’) (a)
720KB 3.5-inch 9 sectors/track (‘‘MS-DOS’’) (b)
1.44MB 3.5-inch 18 sectors/track (‘‘MS-DOS’’) (b)
720KB 5.25-inch 9 sectors/track (‘‘MS-DOS’’) (b)

On systems equipped with floppy disk drives capable of using high-density floppy diskettes, the driver automatically detects whether the inserted floppy diskette is either a double-density or a high-density medium.

DIAGNOSTICS

fdc: unable to allocate DMA buffer The driver found the controller, but was unable to allocate the amount of chip memory required for its DMA buffers.

SEE ALSO

`fdformat`(1), `autoconf`(4)

BUGS

In order to detect a floppy disk drive’s capability of using high-density media, a high-density diskette has to be inserted for the duration of the autoconfiguration phase of the drive.

The **fdc** driver supports neither the `fdformat`(1) facility nor the ⟨`sys/fdio.h`⟩ interface.

NAME

fdc — NEC 765 floppy disk controller driver

SYNOPSIS

```
fdc0 at isa? port 0x3f0 irq 6 drq 2
fdc* at acpi?
fdc* at pnpbios? index ?
fd*  at fdc? drive ?
```

DESCRIPTION

The **fdc** driver provides support for the NEC 765 floppy disk controller and floppy disk drives, commonly found on IBM-PC compatible systems.

The driver supports the following floppy diskette formats by using particular partitions:

1.44MB 3.5-inch (b)
1.2MB 5.25-inch (c)
360KB 5.25-inch (1.2MB drive) (d)
360KB 5.25-inch (IBM-PC drive) (e)
720KB 3.5-inch (f)
720KB 5.25-inch (g)
360KB 3.5-inch (h)

Partition *a* selects the default format for the attached floppy drive, as determined by the BIOS configuration for the diskette drive.

FORMATTING

The driver supports floppy disk formatting using the interfaces in ⟨sys/fdio.h⟩:

`FDIOCGETFORMAT` *`struct fdformat_parms`*
: Fetch current formatting parameters. This gets the default parameters for the open device if no parameters have been set during the session.

`FDIOCSETFORMAT` *`struct fdformat_parms`*
: Set formatting parameters. The driver saves this state and it persists while the device is open.

`FDIOCFORMAT_TRACK` *`struct fdformat_cmd`*
: Format a track on the medium. If this call returns `EINVAL`, the track formatting parameters were out of range for the medium. If it returns `EIO`, there was a medium error while formatting the track.

`FDIOCSETOPTS` *`int`*
: Set driver options which persist until the device is closed. The options should be the logical OR of the desired values below:
  `FDOPT_NORETRY` Do not retry operations on failure
  `FDOPT_SILENT` Do not print error messages to the console

`FDIOCGETOPTS` *`int`*
: Fetch drive options.

A typical use of the formatting facilities would be to open the device, call `FDIOCGETFORMAT` to fetch the current format parameters, perhaps change a parameter or two, display the formatting details to the user, and then call `FDIOCSETFORMAT` followed by a series of calls to `FDIOCFORMAT_TRACK`.

SEE ALSO

`fdformat`(1), `acpi`(4), `isa`(4), `pnpbios`(4)

HISTORY

The **fdc** formatting support appeared in NetBSD 1.3.

NAME

fdc — Sun SPARCstation i82072 or i82077 floppy disk controller driver

SYNOPSIS

```
fdc0 at sbus0  ( SBus based machines )
fdc0 at ebus0  ( PCI based machines )
fd* at fdc0
```

DESCRIPTION

This is the driver for the built-in floppy disk drive run by the Intel i82072 or i82077 controller chip found on the SPARCstation desktop systems, and other SPARC systems.

Bits [0-3] of the minor device number of the special files referring to this device encode the floppy density as follows:

0 3.5'' 1.44MB floppy diskettes.

1 3.5'' 720KB floppy diskettes.

2 3.5'' 360KB floppy diskettes.

3 3.5'' 1.2MB/NEC Japanese format floppy diskettes.

FORMATTING

The driver supports floppy disk formatting using the interfaces in ⟨sys/fdio.h⟩:

`FDIOCGETFORMAT` *struct fdformat_parms*
: Fetch current formatting parameters. This gets the default parameters for the open device if no parameters have been set during the session.

`FDIOCSETFORMAT` *struct fdformat_parms*
: Set formatting parameters. The driver saves this state and it persists while the device is open.

`FDIOCFORMAT_TRACK` *struct fdformat_cmd*
: Format a track on the medium. If this call returns `EINVAL`, the track formatting parameters were out of range for the medium. If it returns `EIO`, there was a medium error while formatting the track.

`FDIOCSETOPTS` *int*
: Set driver options which persist until the device is closed. The options should be the logical OR of the desired values below:

  `FDOPT_NORETRY` Do not retry operations on failure
  `FDOPT_SILENT` Do not print error messages to the console

`FDIOCGETOPTS` *int*
: Fetch drive options.

A typical use of the formatting facilities would be to open the device, call `FDIOCGETFORMAT` to fetch the current format parameters, perhaps change a parameter or two, display the formatting details to the user, and then call `FDIOCSETFORMAT` followed by a series of calls to `FDIOCFORMAT_TRACK`.

SEE ALSO

`eject`(1), `fdformat`(1)

HISTORY

The **fdc** driver first appeared in NetBSD 4.0.

BUGS

The ebus attachment does not yet work.

NAME

finsio — Fintek LPC Super I/O driver

SYNOPSIS

```
finsio0 at isa? port 0x4e
```

DESCRIPTION

The **finsio** driver provides support for the Fintek F71805F, F71806F, F71862FG, F71872, F71882 and F71883 LPC Super I/O chips.

Only the Hardware Monitor device is supported and it provides you to monitor the sensors through the envsys(4) API.

The **finsio** Super I/O Hardware Monitor supports 15 sensors (this depends on the chip ID):

Sensor	Units	Typical Use
IN0	uV DC	Internal 3.3Vcc
IN1	uV DC	Memory Vtt
IN2	uV DC	RAM Vcc
IN3	uV DC	Chipset V
IN4	uV DC	+5V
IN5	uV DC	+12V
IN6	uV DC	Vcc 1.5V
IN7	uV DC	Vcore
IN8	uV DC	Vsb
IN9	uV DC	Vbat (optional)
Temp1	uK	CPU Temperature
Temp2	uK	Motherboard Temperature
Temp3	uK	(undefined)
Fan0	RPM	CPU Fan
Fan1	RPM	User Fan1
Fan2	RPM	User Fan2

SEE ALSO

envsys(4), envstat(8)

HISTORY

The **finsio** driver first appeared in NetBSD 5.0.

AUTHORS

The **finsio** driver was written by Geoff Steckel and Juan Romero Pardines.

NAME

fl — console floppy interface

DESCRIPTION

This is a simple interface to the DEC RX01 floppy disk unit, which is part of the console LSI-11 subsystem for VAX-11/780s. Access is given to the entire floppy consisting of 77 tracks of 26 sectors of 128 bytes.

All I/O is raw; the seek addresses in raw transfers should be a multiple of 128 bytes and a multiple of 128 bytes should be transferred, as in other “raw” disk interfaces.

FILES

`/dev/floppy`

DIAGNOSTICS

None.

SEE ALSO

`arff`(8)

HISTORY

The **fl** driver appeared in 4.0BSD.

BUGS

Multiple console floppies are not supported.

If a write is given with a count not a multiple of 128 bytes then the trailing portion of the last sector will be zeroed.

NAME

floppy — Atari floppy interface

SYNOPSIS

```
fd0 at fdc0 unit 0
fd1 at fdc0 unit 1
```

DESCRIPTION

This is an interface to the builtin and external floppy drives on the Atari. Currently, there is no disklabel support for the floppy drives. Instead, the floppy interface uses the partition number embedded in the minor device number to distinguish between various floppy formats.

Currently, the following formats are supported:

Partition	Size	sides	tracks	sectors/track
a	360Kb	1	80	9
b	720Kb	2	80	9
c	1.44Mb	2	80	18

FILES

`/dev/fd[01][abc]` Block files
`/dev/rfd[01][abc]` Raw files

DIAGNOSTICS

None.

NAME

fms — Forte Media FM801 audio device driver

SYNOPSIS

```
fms*   at pci? dev ? function ?
audio* at audiobus?
mpu*   at fms?
opl*   at fms?
```

DESCRIPTION

The **fms** device driver supports the Forte Media FM801 sound card.

SEE ALSO

ac97(4), audio(4), mpu(4), opl(4), pci(4)

HISTORY

The **fms** device driver appeared in NetBSD 1.5.

NAME

fmv — Fujitsu FMV-18x Ethernet cards device driver

SYNOPSIS

```
fmv0 at isa? port 0x2a0 irq ?
fmv* at isapnp?
```

DESCRIPTION

The **fmv** driver supports the Fujitsu FMV-18x ISA network adapters based on the Fujitsu MB86964/MB86965A Ethernet controllers. Supported boards include:

Fujitsu FMV-181 ISA Ethernet

Fujitsu FMV-181A ISA Ethernet

Fujitsu FMV-182 ISA Ethernet

Fujitsu FMV-182A ISA Ethernet

Fujitsu FMV-183 ISA-PnP Ethernet

For Fujitsu FMV-186/186A/188 PCI Ethernet adapters, use fxp(4) driver.

SEE ALSO

ate(4), ifmedia(4), intro(4), isa(4), isapnp(4), mbe(4), ifconfig(8)

http://www.fujitsumicro.com/

HISTORY

The **fmv** driver appeared in NetBSD 1.4.

NAME

fpa, **fea**, **fta** — DEC FDDI interface driver

SYNOPSIS

```
fpa* at pci? dev ? function ?
fea* at eisa? slot ?
```

alpha and pmax

```
fta* at tc? slot ? offset ?
```

DESCRIPTION

Cards supported by the **fpa**, **fea** and **fta** device driver are:

DEFPA DEC PCI FDDI Controller

DEFEA DEC EISA FDDI Controller

DEFTA DEC TURBOchannel FDDI Controller

respectively. All variants of either controller are supported including the DAS and SAS configurations.

DIAGNOSTICS

fea%d: error: desired IRQ of %d does not match device's actual IRQ (%d) The device probe detected that the DEFEA board is configured for a different interrupt than the one specified in the kernel configuration file.

fea%d: error: memory not enabled! ECU reconfiguration required The device probe found that no device memory had been configured on the DEFEA. The DEFEA can be configured with no device memory, this driver requires a minimum of 1K device memory be set up. The ECU (EISA Configuration Utility) will need to be run to change the settings.

SEE ALSO

arp(4), netintro(4), ifconfig(8)

AUTHORS

The **fpa**, **fea** and **fta** device driver and manual page were written by Matt Thomas.

BUGS

Normally, the device driver will not enable the reception of SMT frames. However if the IFF_LINK1 flag is set, the device driver will enable the reception of SMT frames and pass them up to the Berkeley Packet Filter for processing.

NAME

frodo — HP Apollo 9000/4xx Frodo ASIC

SYNOPSIS

```
frodo0 at intio?
```

DESCRIPTION

The **frodo** driver supports the Frodo ASIC (a.k.a. Apollo Utility chip) found on HP Apollo 9000/4xx series workstations.

SUPPORTED DEVICES

NetBSD includes FRODO drivers, sorted by device type and driver name:

Serial devices

dnkbd	8250-like serial ports (Domain keyboard flavor)
com	8250-like serial ports (tty flavor)

SEE ALSO

com(4), dnkbd(4), intio(4)

NAME

fss, **fssbs** — file system snapshot device

SYNOPSIS

```
pseudo-device fss 4
```

DESCRIPTION

The **fss** driver provides a read-only interface to the snapshot of a currently mounted file system. Reading from a **fss** device gives the view of the file system when the snapshot was taken. It can be configured via `ioctl`(2).

IOCTLS

The `ioctl`(2) command codes below are defined in ⟨`sys/dev/fssvar.h`⟩.

The (third) argument to `ioctl`(2) should be a pointer to the type indicated.

`FSSIOCSET(struct fss_set)`

Configures a **fss** device.

```
struct fss_set {
        char *fss_mount;
        char *fss_bstore;
        blksize_t fss_csize;
};
```

The struct element *fss_mount* is the mount point of the file system. The struct element *fss_bstore* is either a regular file or a raw disk device where data overwritten on the file system will be saved. The struct element *fss_csize* is the preferred size of this data.

`FSSIOCGET(struct fss_get)`

Gets the status of a **fss** device.

```
struct fss_get {
        char fsg_mount[MNAMELEN];
        struct timeval fsg_time;
        blksize_t fsg_csize;
        blkcnt_t fsg_mount_size;
        blkcnt_t fsg_bs_size;
};
```

The struct element *fsg_mount* is the mount point of the file system. The struct element *fsg_time* is the time this snapshot was taken. The struct element *fsg_csize* is the current size of data clusters. The struct element *fsg_mount_size* is the number of clusters of the file system. The struct element *fsg_bs_size* is the number of clusters written to the backing store.

`FSSIOCCLR`

Unconfigures a **fss** device.

`FSSIOFSET(int)`

Sets the flags of a **fss** device. Possible flags are:

`FSS_UNCONFIG_ON_CLOSE`

Unconfigure the **fss** device on the last close.

`FSSIOFGET(int)`

Gets the flags of a **fss** device.

KERNEL THREADS

For each active snapshot device there is a kernel thread that handles the backing store. This thread is named *fssbsN* where *N* is the device minor number.

FILES

`/dev/rfss?`
`/dev/fss?`

SEE ALSO

`fssconfig`(8), `mount`(8), `umount`(8)

HISTORY

The **`fss`** device appeared in NetBSD 2.0.

BUGS

This driver is *experimental*. Be sure you have a backup before you use it.

NAME

fwip — IP over IEEE1394 driver

SYNOPSIS

fwip* **at ieee1394if?**

DESCRIPTION

The **fwip** driver provides standard IP over IEEE1394 based on the protocols described in RFC 2734 and RFC 3146.

The ieee1394if(4) and fwohci(4) drivers must be configured in the kernel as well.

SEE ALSO

arp(4), fwohci(4), ieee1394if(4), netintro(4), ifconfig(8), sysctl(8)

HISTORY

The **fwip** device driver first appeared in FreeBSD 5.3. It was added to NetBSD 4.0.

AUTHORS

The **fwip** driver and this manual page were written by Doug Rabson, based on earlier work by Hidetoshi Shimokawa. It was added to NetBSD 4.0 by KIYOHARA Takashi.

BUGS

This driver currently does not support the MCAP protocol for multicast IP over IEEE1394. Multicast packets are treated as broadcast packets which is sufficient for most trivial uses of multicast.

NAME

fwohci — OHCI IEEE1394 chipset device driver

SYNOPSIS

```
fwohci* at cardbus? function ?
fwohci* at pci? dev ? function ?
```

HARDWARE

The **fwohci** driver provides support for PCI/CardBus IEEE1394 interface cards. The driver supports the following IEEE 1394 OHCI chipsets:

- Adaptec AHA-894x/AIC-5800
- Apple Pangea
- Apple UniNorth
- Intel 82372FB
- IOGEAR GUF320
- Lucent / Agere FW322/323
- NEC uPD72861
- NEC uPD72870
- NEC uPD72871/2
- NEC uPD72873
- NEC uPD72874
- National Semiconductor CS4210
- Ricoh R5C551
- Ricoh R5C552
- Sony CX3022
- Sony i.LINK (CXD1947)
- Sony i.LINK (CXD3222)
- Texas Instruments PCI4410A
- Texas Instruments PCI4450
- Texas Instruments PCI4451
- Texas Instruments TSB12LV22
- Texas Instruments TSB12LV23
- Texas Instruments TSB12LV26
- Texas Instruments TSB43AA22
- Texas Instruments TSB43AB21/A/AI/A-EP
- Texas Instruments TSB43AB22/A
- Texas Instruments TSB43AB23
- Texas Instruments TSB82AA2
- VIA Fire II (VT6306)

SEE ALSO

fwip(4), ieee1394if(4), sbp(4), fwctl(8)

HISTORY

The **fwohci** device driver first appeared in FreeBSD 5.0. It was added to NetBSD 4.0.

AUTHORS

The **fwohci** device driver was written by Katsushi Kobayashi and Hidetoshi Shimokawa. It was added to NetBSD 4.0 by KIYOHARA Takashi.

BUGS

The driver allows physical access from any node on the bus by default. This means that any devices on the bus can read and modify any memory space which can be accessed by an IEEE 1394 OHCI chip. It is allowed mostly for `sbp`(4) devices. This should be changed to allow it only for specific devices. Anyway, IEEE1394 is a bus and not expected to be connected with un-trustable devices because a node can monitor all the traffic.

NAME

fxp — Intel i8255x 10/100 Ethernet device driver

SYNOPSIS

```
fxp* at cardbus? function ?
fxp* at pci? dev ? function ?
```

DESCRIPTION

The **fxp** device driver supports Ethernet interfaces based on the Intel i82557, i82558, i82559, and i82550 10/100 PCI Ethernet chips.

Certain versions of the i8255x support loading microcode which implements a receive interrupt mitigation function, known as “CPUSaver”. Use of this option can improve performance in some situations by reducing interrupt load on the host. This option is available on the following chip versions:

- i82558 step A4 (rev 4)
- i82558 step B0 (rev 5)
- i82559 step A0 (rev 8)
- i82559S step A (rev 9)
- i82550 (rev 12)
- i82550 step C (rev 13)

This option is enabled by setting the “link0” option with `ifconfig`(8).

Some chipset revisions can suffer from a receiver-side lockup bug which can be mitigated by resetting the chip every sixteen seconds without traffic. Since the probe for affected chipsets generates false positives and the workaround can cause momentary loss of responsiveness, particularly noticeable when playing audio, the workaround is not enabled by default. The boot messages will indicate if any interface may have this issue. The workaround is enabled by setting the “link1” option with `ifconfig`(8).

HARDWARE

Cards supported by the **fxp** driver include:

- Intel EtherExpress Pro 10+
- Intel EtherExpress Pro 100B
- Intel EtherExpress Pro 100+
- Intel InBusiness 10/100
- Intel PRO/100 S

MEDIA SELECTION

Media selection is supported via MII. See `ifmedia`(4) and `mii`(4) for more information.

EtherExpress Pro 10+ boards may use a Seeq 80c24 AutoDUPLEX(tm) media interface. Boards with these chips do not support media selection, as the 80c24 has no programming interface, and no way to read link status. These boards claim a media of "manual" since they self-configure based on the configuration of the link partner (hub or switch).

DIAGNOSTICS

fxp0: WARNING: SCB timed out! The driver timed out waiting for the chip’s command interface to become ready.

fxp0: too many segments, aborting The driver encountered a packet that included too many DMA segments, and was not able to allocate a new buffer to transmit the packet from. The packet has been dropped.

fxp0: too many segments, retrying The driver encountered a packet that included too many DMA segments, and allocated a new buffer to transmit the packet from.

fxp0: can't load mbuf chain, error = %d The driver was unable to load a transmit DMA map, and has reported the errno value.

fxp0: device timeout The device failed to generate a transmit complete interrupt for the last packet transmitted. The device has been reset.

fxp0: can't load rx buffer, error = %d The driver was unable to load the DMA map for a receive buffer, and has reported the errno value. This error is currently fatal, and will panic the system.

fxp0: fxp_mdi_read: timed out The MDIO failed to become ready during an MII read operation.

fxp0: fxp_mdi_write: timed out The MDIO failed to become ready during an MII write operation.

fxp0: May need receiver lock-up workaround The interface may need to be periodically reset to workaround a receiver lock-up bug.

SEE ALSO

`cardbus`(4), `ifmedia`(4), `intro`(4), `mii`(4), `pci`(4), `ifconfig`(8)

NAME

g2bus — G2 bus support

SYNOPSIS

```
g2bus0 at shb?
```

DESCRIPTION

The **g2bus** driver provides support for the Dreamcast G2 bus layer, to which the audio subsystem and the expansion slot are connected.

SEE ALSO

gapspci(4)

HISTORY

The **g2bus** device driver appeared in NetBSD 1.6.

NAME

gapspci — GAPS PCI bridge support

SYNOPSIS

```
gapspci* at g2bus?
pci*     at gapspci?
```

DESCRIPTION

The **gapspci** driver provides support for the GAPS PCI bridge. The SEGA Broadband Adapter (model HIT-0400/HIT-0401) is composed of this bridge and an rtk(4) Ethernet controller.

SEE ALSO

g2bus(4), pci(4), rtk(4)

HISTORY

The **gapspci** device driver appeared in NetBSD 1.6.

NAME

gb — HP98700 ‘‘Gatorbox’’ graphics device interface

DESCRIPTION

This driver is for the HP98700 and 98710 graphics devices, also known as the Gatorbox. The term ‘‘Gator’’ will often be used, and it is not to be confused with ‘‘Gator’’ used in reference to an HP 9837 or 200/237 machine. Also, the term Gatorbox is used for the 98700 alone, with the 98701 frame buffer memory or with the 98710 accelerator installed. This driver merely checks for the existence of the device and does minimal set up, as it is expected the applications will initialize the device to their requirements.

The 98700 can be used as the only graphics device on a system, in which case it will be used as the system console. It can also be installed as a secondary display device. For the first case, the HP 98287A M.A.D. interface card should be set to internal control space. This will put the frame buffer at the DIO address 0x200000 and the control registers at 0x560000. At this address it will be the ‘‘preferred’’ console device (see `cons`(4)). For use as a secondary device, the 98287A should be set to frame buffer address 0x300000, and to an external select code.

It should be noted that this configuration will conflict with the 98547 display card which has a 2 megabyte frame buffer starting at address 0x200000. The 98700 should only be installed as a secondary device in a machine with a 1 bit 98544 display card or 4 bit 98545 card. The *98700H Installation Guide* contains further configuration information.

The `ioctl`(2) calls supported by the BSD system for the Gatorbox are:

`GRFIOCGINFO`
: Get Graphics Info

  Get info about device, setting the entries in the *grfinfo* structure, as defined in ⟨`hpdev/grfioctl.h`⟩. For the standard 98700, the number of planes should be 4. The number of colors would therefore be 15, excluding black. With the 98701 option installed there will be another 4 planes for a total of 8, giving 255 colors.

`GRFIOCON`
: Graphics On

  Turn graphics on by enabling CRT output. The screen will come on, displaying whatever is in the frame buffer, using whatever colormap is in place.

`GRFIOCOFF`
: Graphics Off

  Turn graphics off by disabling output to the CRT. The frame buffer contents are not affected.

`GRFIOCMAP`
: Map Device to user space

  Map in control registers and framebuffer space. Once the device file is mapped, the frame buffer structure is accessible. The frame buffer structure describing the 98700 is given in ⟨`hpdev/grf_gbreg.h`⟩.

`GRFIOCUNMAP`
: Unmap Device

  Unmap control registers and framebuffer space.

  For further information about the use of ioctl see the man page.

FILES

`/dev/grf?` BSD special file
`/dev/crt98700` HP-UX *starbase* special file

EXAMPLES

A small example of opening, mapping and using the device is given below. For more examples of the details on the behavior of the device, see the device dependent source files for the X Window System, in the `/usr/src/new/X/libhp.fb` directory.

```
struct gboxfb *gbox;
u_char *Addr, frame_buffer;
struct grfinfo gi;
int disp_fd;

disp_fd = open("/dev/grf0",1);

if (ioctl (disp_fd, GRFIOCGINFO, &gi) < 0) return -1;

(void) ioctl (disp_fd, GRFIOCON, 0);

Addr = (u_char *) 0;
if (ioctl (disp_fd, GRFIOCMAP, &Addr) < 0) {
(void) ioctl (disp_fd, GRFIOCOFF, 0);
return -1;
}
gbox = (gboxfb *) Addr;                          /* Control Registers   */
frame_buffer = (u_char *) Addr + gi.gd_regsize; /* Frame buffer memory */
```

DIAGNOSTICS

None under BSD. HP-UX The CE.utilities/Crtadjust programs must be used.

ERRORS

`[ENODEV]` no such device.

`[EBUSY]` Another process has the device open.

`[EINVAL]` Invalid ioctl specification.

SEE ALSO

`ioctl`(2), `grf`(4)

NAME

gbus — internal bus on AlphaServer CPU modules

SYNOPSIS

gbus* at tlsb? node ? offset ?

DESCRIPTION

The **gbus** driver provides support for the internal bus on AlphaServer CPU modules.

The following devices are supported by the **gbus** driver:

mcclock DS1287 real-time clock

SEE ALSO

intro(4), mcclock(4), tlsb(4)

NAME

gcscide — IDE Controller for the AMD CS5535 Companion device

SYNOPSIS

```
gcscide* at pci? dev ? function ? flags 0x0000
```

DESCRIPTION

The **gcscide** driver supports the IDE controller of the AMD CS5535 Companion device, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **gcscide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

NAME

gcscpcib — AMD CS553[56] PCI-ISA bridge with timecounter, watchdog timer, and GPIO

SYNOPSIS

```
gcscpcib* at pci? dev ? function ?
gpio* at gcscpcib?
isa*  at gcscpcib?
```

DESCRIPTION

The **gcscpcib** driver provides support for the AMD Geode CS5535 and CS5536 Companion chip. This device contains a number of components, including a PCI-ISA bridge. Besides the core functionality, the **gcscpcib** driver implements a 32-bit 3.5 MHz timecounter, a watchdog timer device, and a GPIO device. The watchdog timer may be configured via the `wdogctl`(8) utility and the GPIO pins can be manipulated via the `gpioctl`(8) utility.

SEE ALSO

`gpio`(4), `intro`(4), `isa`(4), `pci`(4), `pcib`(4), `gpioctl`(8), `wdogctl`(8)

HISTORY

The **gcscpcib** driver first appeared in NetBSD 5.0.

AUTHORS

The **gcscpcib** driver was written by Michael Shalayeff and Yojiro UO. The GPIO parts were added by Marc Balmer ⟨mbalmer@openbsd.org⟩.

NAME

gdrom — GD-ROM disk driver

SYNOPSIS

```
gdrom0 at shb?
```

DESCRIPTION

The **gdrom** driver provides support for the Dreamcast CD-ROM ("GD-ROM") storage device.

FILES

`/dev/gdrom0`*p*	block mode GD-ROM disk partition *p*
`/dev/rgdrom0`*p*	raw mode GD-ROM disk partition *p*

HISTORY

The **gdrom** device driver appeared in NetBSD 1.6.

NAME

gem — ERI/GEM/GMAC Ethernet device driver

SYNOPSIS

```
gem* at pci? dev ? function ?
gem* at sbus? slot ? offset ?
```

Configuration of PHYs may also be necessary. See `mii`(4).

DESCRIPTION

The **gem** driver provides support for the GMac Ethernet hardware found mostly in the last Apple PowerBooks G3s and most G4-based Apple hardware, as well as many Sun UltraSPARCs.

Cards supported by this driver include:

- Sun GEM gigabit ethernet (including SX fibre variants)
- Sun ERI 10/100
- Apple GMAC

The GEM family supports hardware checksumming to assist in computing IPv4 TCP checksums. The **gem** driver supports this feature of the chip. See `ifconfig`(8) for information on how to enable this feature.

SEE ALSO

`bmtphy`(4), `ifmedia`(4), `intro`(4), `mii`(4), `ifconfig`(8)

Sun Microsystems, *GEM Gigabit Ethernet ASIC Specification*, http://www.sun.com/processors/manuals/ge.pdf.

HISTORY

The **gem** device driver appeared in NetBSD 1.6.

AUTHORS

The **gem** driver was written by Eduardo Horvath ⟨eeh@NetBSD.org⟩. The man page was written by Thomas Klausner ⟨wiz@NetBSD.org⟩.

BUGS

The hardware checksumming support does not support IPv4 UDP, although this was allowed prior to NetBSD 5. Also, the hardware IPv4 TCP receive checksumming support has bugs, so this is disabled.

On the SX fibre variants of the hardware, the link will stay down if there is a duplex mismatch. Also, packet transmission may fail when in **half-duplex** mode.

NAME

genfb — generic framebuffer console driver

SYNOPSIS

```
genfb* at pci?
genfb* at sbus?
wsdisplay* at genfb?
```

DESCRIPTION

The **genfb** driver provides support for generic framebuffers that have no native driver. All it needs are some parameters to describe the framebuffer and an address.

PCI

When attaching to a pci(4) bus the driver is configured via device properties:

`width (uint32)`
: Width in pixels.

`height (uint32)`
: Height in pixels.

`stride (uint32)`
: Line size in bytes.

`depth (uint32)`
: Bits per pixel.

`is_console (bool)`
: If true, **genfb** will try to become the system console.

`address (uint32)`
: Bus address of the framebuffer.

SBus

When attaching to sbus(4) all those parameters are retrieved from the firmware.

SEE ALSO

pci(4), sbus(4), wscons(4), wsdisplay(4)

BUGS

There is no way to change the color map even when the firmware supports it. The pci(4) bus frontend has only been tested on macppc and requires machine dependent code to pass the properties mentioned above. So far only macppc provides them.

NAME

gentbi — Driver for generic 1000BASE-X ten-bit interfaces

SYNOPSIS

gentbi* at mii? phy ?

DESCRIPTION

The **gentbi** driver supports generic ten-bit interfaces that implement the 1000BASE-X protocol, including 1000BASE-SX, 1000BASE-LX, and 1000BASE-CX.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

geodecntr — AMD Geode SC1100 High Resolution Counter

SYNOPSIS

```
geodegcb*  at pci? dev ? function ?
geodecntr* at geodegcb?
```

DESCRIPTION

The **geodecntr** driver supports the AMD SC1100 controllers and provides a timecounter implementation for the 27MHz counter.

The other timing sources in the SC1100 suffer from powermanagement features that prohibit i8254 and TSC to be used for time keeping. This **geodecntr** driver makes the 27MHz counter available for the timecounter(9) framework.

SEE ALSO

timecounter(9)

NAME

geodeide — AMD Geode IDE disk controllers driver

SYNOPSIS

geodeide* **at pci? dev ? function ? flags 0x0000**

DESCRIPTION

The **geodeide** driver supports the AMD Geode CS5530A and SC1100 IDE controllers, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **geodeide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

The SC1100 controller requires 4-byte aligned data transfers and cannot handle transfers of exactly 64 kilobytes.

The CS5530 multifunction chip/core's IDE section claims to be capable of UDMA mode 2 (33.3MB/s) but in practice using that mode swamps the controller so badly that **geodeide** limits the UDMA negotiation to mode 1 (25MB/s) so that the other functions of this chip continue to work.

The IDE DMA engine in the CS5530 can only do transfers on cache-line (16-byte) boundaries. Attempts to perform DMA on any other alignment will crash the system. This problem may also exist in the SC1100 since the CS5530 was its direct predecessor, and it is not clear that National Semiconductor fixed any bugs in it.

The **geodeide** driver will reject attempts to DMA to buffers not aligned to the required boundary. The wd(4) disk driver will back off to PIO mode to accomplish these transfer requests, at reduced system performance.

NAME

geodewdog — AMD Geode SC1100 Watchdog Timer driver

SYNOPSIS

```
geodegcb*  at pci? dev ? function ?
geodewdog* at geodegcb?
```

DESCRIPTION

The **geodewdog** driver supports the AMD Geode SC1100 microcontroller's integrated watchdog timer. The watchdog timer may be configured for expiration periods between one and 16776 seconds (more than 4.5 hours).

SEE ALSO

elansc(4), wdogctl(8)

HISTORY

The **geodewdog** device first appeared in NetBSD 4.0.

AUTHORS

The **geodewdog** driver was written by David Young ⟨dyoung@NetBSD.org⟩. It was derived from the elansc(4) driver by
Jason R. Thorpe ⟨thorpej@NetBSD.org⟩.

NAME

gif — generic tunnel interface

SYNOPSIS

```
pseudo-device gif
```

DESCRIPTION

The **gif** interface is a generic tunneling pseudo device for IPv4 and IPv6. It can tunnel IPv[46] traffic over IPv[46]. Therefore, there can be four possible configurations. The behavior of **gif** is mainly based on RFC 2893 IPv6-over-IPv4 configured tunnel. **gif** can also tunnel ISO traffic over IPv[46] using EON encapsulation.

To use **gif**, the administrator must first create the interface and then configure protocol and addresses used for the outer header. This can be done by using `ifconfig`(8) **create** and **tunnel** subcommands, or `SIOCIFCREATE` and `SIOCSIFPHYADDR` ioctls. Also, administrator needs to configure protocol and addresses used for the inner header, by using `ifconfig`(8). Note that IPv6 link-local address (those start with `fe80::`) will be automatically configured whenever possible. You may need to remove IPv6 link-local address manually using `ifconfig`(8), when you would like to disable the use of IPv6 as inner header (like when you need pure IPv4-over-IPv6 tunnel). Finally, use routing table to route the packets toward **gif** interface.

gif can be configured to be ECN friendly. This can be configured by `IFF_LINK1`.

ECN friendly behavior

gif can be configured to be ECN friendly, as described in `draft-ietf-ipsec-ecn-02.txt`. This is turned off by default, and can be turned on by `IFF_LINK1` interface flag.

Without `IFF_LINK1`, **gif** will show a normal behavior, like described in RFC 2893. This can be summarized as follows:

Ingress	Set outer TOS bit to `0`.
Egress	Drop outer TOS bit.

With `IFF_LINK1`, **gif** will copy ECN bits (`0x02` and `0x01` on IPv4 TOS byte or IPv6 traffic class byte) on egress and ingress, as follows:

Ingress	Copy TOS bits except for ECN CE (masked with `0xfe`) from inner to outer. set ECN CE bit to `0`.
Egress	Use inner TOS bits with some change. If outer ECN CE bit is `1`, enable ECN CE bit on the inner.

Note that the ECN friendly behavior violates RFC 2893. This should be used in mutual agreement with the peer.

Packet format

Every inner packet is encapsulated in an outer packet. The inner packet may be IPv4, IPv6, or ISO CLNP. The outer packet may be IPv4 or IPv6, and has all the usual IP headers, including a protocol field that identifies the type of inner packet.

When the inner packet is IPv4, the protocol field of the outer packet is 4 (`IPPROTO_IPV4`). When the inner packet is IPv6, the protocol field of the outer packet is 41 (`IPPROTO_IPV6`). When the inner packet is ISO CNLP, the protocol field of the outer packet is 80 (`IPPROTO_EON`).

Security

Malicious party may try to circumvent security filters by using tunneled packets. For better protection, **gif** performs martian filter and ingress filter against outer source address, on egress. Note that martian/ingress filters are no way complete. You may want to secure your node by using packet filters. Ingress filter can be turned off by IFF_LINK2 bit.

EXAMPLES

Configuration example:

```
Host X--NetBSD A  ----------------tunnel---------- cisco D------Host E
           \                                          |
            \                                        /
             +-----Router B--------Router C---------+
```

On NetBSD system A (NetBSD):

```
# route add default B
# ifconfig gifN create
# ifconfig gifN A netmask 0xffffffff tunnel A D up
# route add E 0
# route change E -ifp gif0
```

On Host D (Cisco):

```
Interface TunnelX
 ip unnumbered D   ! e.g. address from Ethernet interface
 tunnel source D   ! e.g. address from Ethernet interface
 tunnel destination A
ip route C <some interface and mask>
ip route A mask C
ip route X mask tunnelX
```

or on Host D (NetBSD):

```
# route add default C
# ifconfig gifN D A
```

If all goes well, you should see packets flowing.

If you want to reach Host A over the tunnel (from the Cisco D), then you have to have an alias on Host A for e.g. the Ethernet interface like: **ifconfig** *<etherif> alias Y* and on the cisco **ip** *route Y mask tunnelX*.

SEE ALSO

etherip(4), inet(4), inet6(4), ifconfig(8)

C. Perkins, "IP Encapsulation within IP", *RFC 2003*, October 1996, ftp://ftp.isi.edu/in-notes/rfc2003.txt.

R. Gilligan and E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers", *RFC 2893*, August 2000, ftp://ftp.isi.edu/in-notes/rfc2893.txt.

Sally Floyd, David L. Black, and K. K. Ramakrishnan, *IPsec Interactions with ECN*, December 1999, draft-ietf-ipsec-ecn-02.txt.

F. Baker and P. Savola, "Ingress Filtering for Multihomed Networks", *RFC 3704*, March 2004, ftp://ftp.isi.edu/in-notes/rfc3704.txt.

STANDARDS

IPv4 over IPv4 encapsulation is compatible with RFC 2003. IPv6 over IPv4 encapsulation is compatible with RFC 2893.

HISTORY

The **gif** device first appeared in WIDE hydrangea IPv6 kit.

BUGS

There are many tunneling protocol specifications, defined differently from each other. **gif** may not interoperate with peers which are based on different specifications, and are picky about outer header fields. For example, you cannot usually use **gif** to talk with IPsec devices that use IPsec tunnel mode.

The current code does not check if the ingress address (outer source address) configured to **gif** makes sense. Make sure to configure an address which belongs to your node. Otherwise, your node will not be able to receive packets from the peer, and your node will generate packets with a spoofed source address.

If the outer protocol is IPv6, path MTU discovery for encapsulated packet may affect communication over the interface.

In the past, **gif** had a multi-destination behavior, configurable via `IFF_LINK0` flag. The behavior was obsoleted and is no longer supported.

NAME

gio — SGI's Graphics I/O (GIO) bus (an early PCI-like bus)

SYNOPSIS

```
gio0 at imc0
gio0 at pic0
```

DESCRIPTION

The **gio** bus is a bus for connecting high-speed peripherals to the main memory and CPU. The devices themselves are typically (but not necessarily) connected to the hpc(4) peripheral controller, and memory and CPU are accessed through the imc(4) (Indy Memory Controller) or pic(4) (Processor Interface Controller). The **gio** bus is found on the Personal Iris 4D/3x, Indigo, Indy, Challenge S, Challenge M, and Indigo2 machines and exists in three incarnations: GIO32, GIO32-bis, and GIO64.

SEE ALSO

giopci(4), grtwo(4), hpc(4), imc(4), light(4), newport(4), pic(4)

HISTORY

The **gio** driver first appeared in NetBSD 1.5.

CAVEATS

Challenge S systems may use only one **gio** DMA-capable expansion card, despite having two slots. Cards based on the hpc(4) controller, such as the GIO32 scsi and E++ Ethernet adapters, must be placed in slot 1 (closest to the side of the case). All other cards must be placed in slot 0 (adjacent to the memory banks).

Indigo2 and Challenge M systems contain either three or four GIO64 connectors, depending on the model. However, in both cases only two electrically distinct slots are present. Therefore, distinct expansion cards may not share physical connectors associated with the same slot. Refer to the PCB stencils to determine the association between physical connectors and slots.

BUGS

Systems employing the imc(4) may experience spurious SysAD bus parity errors when using expansion cards, which do not drive all data lines during a CPU PIO read. The only workaround is to disable SysAD parity checking when using such cards.

NAME

giopci — Attachment for PCI devices bridged to the GIO bus

SYNOPSIS

giopci* **at gio? slot?**

DESCRIPTION

The **giopci** driver provides support for the machine-independent PCI subsystem (described in pci(4)) such that it may attach various GIO expansion boards featuring PCI chipsets sitting behind various special GIO<->PCI bridges.

The following boards are presently supported:

Phobos G100/G130/G160 Fast Ethernet	tlp(4)
Set Engineering GIO Fast Ethernet	tl(4)

SEE ALSO

gio(4), pci(4), tl(4), tlp(4)

HISTORY

The **giopci** driver first appeared in NetBSD 4.0.

NAME

glxsb — Geode LX Security Block crypto accelerator

SYNOPSIS

glxsb* **at pci?**

DESCRIPTION

The **glxsb** driver supports the security block of the Geode LX series processors. The Geode LX is a member of the AMD Geode family of integrated x86 system chips.

Driven by periodic checks for available data from the generator, **glxsb** supplies entropy to the `random`(4) driver for common usage.

glxsb also supports acceleration of AES-CBC operations for `crypto`(4).

SEE ALSO

`crypto`(4), `intro`(4), `pci`(4), `random`(4), `crypto`(9)

HISTORY

The **glxsb** device first appeared in NetBSD 4.0.

AUTHORS

The **glxsb** driver was written for OpenBSD by Tom Cosgrove. It was ported to NetBSD by Jared D. McNeill ⟨jmcneill@NetBSD.org⟩.

BUGS

The **glxsb** driver only provides random numbers and AES acceleration. Since it does not provide HMACs, IPSec will not currently use it; it will however be used by OpenSSH.

NAME

glxtphy — Driver for Level One LXT-1000 10/100/1000 Ethernet PHY

SYNOPSIS

```
glxtphy* at mii? phy ?
```

DESCRIPTION

The **glxtphy** driver supports the Level One LXT-1000 10/100/1000 Ethernet PHY. This PHY is found on a variety of CAT5 Gigabit Ethernet interfaces.

SEE ALSO

`ifmedia`(4), `intro`(4), `mii`(4), `ifconfig`(8)

NAME

gm — Apple GMac Ethernet device driver

SYNOPSIS

```
gm* at pci? dev ? function ?
```

Configuration of PHYs may also be necessary. See mii(4).

DESCRIPTION

The **gm** driver provides support for the GMac Ethernet hardware found mostly in the newest Apple PowerBooks G3s and most G4-based Apple hardware.

This driver is now obsolete. You should be using the gem(4) driver.

Wireless hardware is not supported by this driver, but by wi(4).

SEE ALSO

bm(4), bmtphy(4), brgphy(4), gem(4), mc(4), mii(4), tlp(4), wi(4)

HISTORY

The **gm** device driver appeared in NetBSD 1.5 and is obsoleted by the gem(4) driver.

AUTHORS

The **gm** driver was written by Tsubai Masanari ⟨tsubai@NetBSD.org⟩. The man page was written by Thomas Klausner ⟨wiz@NetBSD.org⟩.

NAME

gphyter — Driver for National Semiconductor DP83861 10/100/1000 Ethernet PHYs

SYNOPSIS

gphyter* at mii? phy ?

DESCRIPTION

The **gphyter** driver supports the National Semiconductor DP83861 and DP83891 (Gig PHYTER) 10/100/1000 Ethernet PHYs. These PHYs are found on a variety of CAT5 Gigabit Ethernet interfaces.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

cec — support for the IEEE488 GPIB

SYNOPSIS

```
gpib* at cec?
```

DESCRIPTION

The **cec** driver supports the IEEE488 general-purpose interface bus (GPIB).

FILES

`/dev/gpib?` GPIB device special file

SEE ALSO

`cs80bus`(4), `ct`(4), `mt`(4), `rd`(4)

HISTORY

The **cec** driver appeared in NetBSD 2.0.

BUGS

The user-level interface to the gpib driver current doesn't do anything useful. The GPIB subsystem places the unfortunate constraint that the local GPIB controller is always the controller in charge (CIC).

NAME

gpio — General Purpose Input/Output

SYNOPSIS

```
gpio* at elansc?
gpio* at gcscpcib?
gpio* at gscpcib?
gpio* at nsclpcsio?
gpio* at ppbus?

#include <sys/types.h>
#include <sys/gpio.h>
#include <sys/ioctl.h>
```

DESCRIPTION

The **gpio** device attaches to the GPIO controller and provides a uniform programming interface to its pins.

Each GPIO controller with an attached **gpio** device has an associated device file under the `/dev` directory, e.g. `/dev/gpio0`. Access from userland is performed through `ioctl`(2) calls on these devices.

IOCTL INTERFACE

The following structures and constants are defined in the ⟨`sys/gpio.h`⟩ header file:

`GPIOINFO (struct gpio_info)`

Returns information about the GPIO controller in the *`gpio_info`* structure:

```
struct gpio_info {
        int gpio_npins;              /* total number of pins available */
};
```

`GPIOPINREAD (struct gpio_pin_op)`

Returns the input pin value in the *`gpio_pin_op`* structure:

```
struct gpio_pin_op {
        int gp_pin;           /* pin number */
        int gp_value;         /* value */
};
```

The *`gp_pin`* field must be set before calling.

`GPIOPINWRITE (struct gpio_pin_op)`

Writes the output value to the pin. The value set in the *`gp_value`* field must be either `GPIO_PIN_LOW` (logical 0) or `GPIO_PIN_HIGH` (logical 1). On return, the *`gp_value`* field contains the old pin state.

`GPIOPINTOGGLE (struct gpio_pin_op)`

Toggles the pin output value, i.e. changes it to the opposite. *`gp_value`* field is ignored and on return contains the old pin state.

`GPIOPINCTL (struct gpio_pin_ctl)`

Changes pin configuration flags with the new ones provided in the *`gpio_pin_ctl`* structure:

```
struct gpio_pin_ctl {
        int gp_pin;           /* pin number */
        int gp_caps;          /* pin capabilities (read-only) */
        int gp_flags;         /* pin configuration flags */
};
```

The *gp_flags* field is a combination of the following flags:

GPIO_PIN_INPUT	input direction
GPIO_PIN_OUTPUT	output direction
GPIO_PIN_INOUT	bi-directional
GPIO_PIN_OPENDRAIN	open-drain output
GPIO_PIN_PUSHPULL	push-pull output
GPIO_PIN_TRISTATE	output disabled
GPIO_PIN_PULLUP	internal pull-up enabled
GPIO_PIN_PULLDOWN	internal pull-down enabled
GPIO_PIN_INVIN	invert input
GPIO_PIN_INVOUT	invert output

Note that the GPIO controller may not support all of these flags. On return the *gp_caps* field contains flags that are supported. If no flags are specified, the pin configuration stays unchanged.

FILES

/dev/gpio*u* GPIO device unit *u* file.

SEE ALSO

ioctl(2), gpioctl(8)

HISTORY

The **gpio** device first appeared in OpenBSD 3.6 and NetBSD 4.0.

AUTHORS

The **gpio** driver was written by Alexander Yurchenko ⟨grange@openbsd.org⟩. **gpio** and was ported to NetBSD by Jared D. McNeill ⟨jmcneill@NetBSD.org⟩.

BUGS

Event capabilities are not supported.

NAME

gpioow — 1-Wire bus bit-banging through GPIO pin

SYNOPSIS

```
gpioow* at gpio? offset 0 mask 0x1
onewire* at gpioow?
```

DESCRIPTION

The **gpioow** driver allows bit-banging a 1-Wire bus as a master using one GPIO pin. The pin is used as a data signal. The GPIO pin must be able to drive an output and read an input.

The pin number is specified in the kernel configuration with the *offset* locator. The *mask* locator should always be 0x1.

SEE ALSO

gpio(4), intro(4), onewire(4)

HISTORY

The **gpioow** driver first appeared in OpenBSD 4.0 and NetBSD 4.0.

AUTHORS

The **gpioow** driver was written by Alexander Yurchenko ⟨grange@openbsd.org⟩ and was ported to NetBSD by Jeff Rizzo ⟨riz@NetBSD.org⟩.

NAME

gre — encapsulating network device

SYNOPSIS

```
pseudo-device gre
```

DESCRIPTION

The **gre** network interface pseudo device encapsulates datagrams into IP. These encapsulated datagrams are routed to a destination host, where they are decapsulated and further routed to their final destination. The "tunnel" appears to the inner datagrams as one hop.

gre interfaces are dynamically created and destroyed with the `ifconfig`(8) **create** and **destroy** subcommands.

This driver currently supports the following modes of operation:

GRE encapsulation (IP protocol number 47)
: Encapsulated datagrams are prepended an outer datagram and a GRE header. The GRE header specifies the type of the encapsulated datagram and thus allows for tunneling other protocols than IP like e.g. AppleTalk. GRE mode is also the default tunnel mode on Cisco routers. This is also the default mode of operation of the **gre***X* interfaces.

GRE in UDP encapsulation
: Encapsulated datagrams are prepended a GRE header, and then they are sent over a UDP socket. Userland may create the socket and "delegate" it to the kernel using the `GRESSOCK` `ioctl`(2). If userland does not supply a socket, then the kernel will create one using the addresses and ports supplied by `ioctl`(2)s `SIOCSLIFPHYADDR`, `GRESADDRD`, and/or `GRESADDRS`.

MOBILE encapsulation (IP protocol number 55)
: Datagrams are encapsulated into IP, but with a shorter encapsulation. The original IP header is modified and the modifications are inserted between the so modified header and the original payload. Like `gif`(4), only for IP in IP encapsulation.

The **gre***X* interfaces support a number of `ioctl`(2)s, such as:

GRESADDRS:
: Set the IP address of the local tunnel end. This is the source address set by or displayed by ifconfig for the **gre***X* interface.

GRESADDRD:
: Set the IP address of the remote tunnel end. This is the destination address set by or displayed by ifconfig for the **gre***X* interface.

GREGADDRS:
: Query the IP address that is set for the local tunnel end. This is the address the encapsulation header carries as local address (i.e. the real address of the tunnel start point.)

GREGADDRD:
: Query the IP address that is set for the remote tunnel end. This is the address the encapsulated packets are sent to (i.e. the real address of the remote tunnel endpoint.)

GRESPROTO:
: Set the operation mode to the specified IP protocol value. The protocol is passed to the interface in (struct ifreq)->ifr_flags. The operation mode can also be given as

  link0 link2 IPPROTO_UDP

link0 -link2 IPPROTO_GRE

-link0 -link2
IPPROTO_MOBILE

to `ifconfig`(8).

GREGPROTO:
Query operation mode.

GRESSOCK:
Delegate a socket from userland to a tunnel interface in UDP encapsulation mode. The file descriptor for the socket is passed in (struct ifreq)->ifr_value.

Note that the IP addresses of the tunnel endpoints may be the same as the ones defined with `ifconfig`(8) for the interface (as if IP is encapsulated), but need not be, as e.g. when encapsulating AppleTalk.

EXAMPLES

Configuration example:

```
Host X-- Host A  ----------------tunnel---------- cisco D------Host E
          \                                          |
           \                                        /
             +------Host B----------Host C----------+
```

On host A (NetBSD):

```
# route add default B
# ifconfig greN create
# ifconfig greN A D netmask 0xffffffff linkX up
# ifconfig greN tunnel A D
# route add E D
```

On Host D (Cisco):

```
Interface TunnelX
 ip unnumbered D   ! e.g. address from Ethernet interface
 tunnel source D   ! e.g. address from Ethernet interface
 tunnel destination A
ip route C <some interface and mask>
ip route A mask C
ip route X mask tunnelX
```

OR On Host D (NetBSD):

```
# route add default C
# ifconfig greN create
# ifconfig greN D A
# ifconfig tunnel greN D A
```

If all goes well, you should see packets flowing ;-)

If you want to reach Host A over the tunnel (from Host D (Cisco)), then you have to have an alias on Host A for e.g. the Ethernet interface like:

```
ifconfig <etherif> alias Y
```

and on the cisco

```
ip route Y mask tunnelX
```

A similar setup can be used to create a link between two private networks (for example in the 192.168 subnet) over the Internet:

```
192.168.1.* --- Router A  -------tunnel-------- Router B --- 192.168.2.*
                    \                              /
                     \                            /
                       +----- the Internet ------+
```

Assuming router A has the (external) IP address A and the internal address 192.168.1.1, while router B has external address B and internal address 192.168.2.1, the following commands will configure the tunnel:

On router A:

```
# ifconfig greN create
# ifconfig greN 192.168.1.1 192.168.2.1
# ifconfig greN tunnel A B
# route add -net 192.168.2 -netmask 255.255.255.0 192.168.2.1
```

On router B:

```
# ifconfig greN create
# ifconfig greN 192.168.2.1 192.168.1.1
# ifconfig greN tunnel B A
# route add -net 192.168.1 -netmask 255.255.255.0 192.168.1.1
```

To setup the same tunnel as above, but using GRE in UDP encapsulation instead of GRE encapsulation, set flags *link0* and *link2*, and specify source and destination UDP ports.

On router A:

```
# ifconfig greN create
# ifconfig greN link0 link2
# ifconfig greN 192.168.1.1 192.168.2.1
# ifconfig greN tunnel A,port-A B,port-B
# route add -net 192.168.2 -netmask 255.255.255.0 192.168.2.1
```

On router B:

```
# ifconfig greN create
# ifconfig greN link0 link2
# ifconfig greN 192.168.2.1 192.168.1.1
# ifconfig greN tunnel B,port-B A,port-A
# route add -net 192.168.1 -netmask 255.255.255.0 192.168.1.1
```

Along these lines, you can use GRE tunnels to interconnect two IPv6 networks over an IPv4 infrastructure, or to hook up to the IPv6 internet via an IPv4 tunnel to a Cisco router.

```
2001:db8:1::/64 -- NetBSD A  -----tunnel----- Cisco B --- IPv6 Internet
                    \                              /
                     \                            /
                       +----- the Internet ------+
```

The example will use the following addressing:
NetBSD
A has the IPv4 address A and the IPv6 address 2001:db8:1::1 (connects to internal network 2001:db8:1::/64).
Cisco B has external IPv4 address B.
All the IPv6 internet world is behind B, so A wants to route 0::0/0

```
(the IPv6 default route) into the tunnel.
The GRE tunnel will use a transit network: 2001:db8:ffff::1/64 on
the
NetBSD
side, and ::2/64 on the Cisco side.
Then the following commands will configure the tunnel:

On router A
(NetBSD):

   # ifconfig greN create
   # ifconfig greN inet6 2001:db8:ffff::1/64
   # ifconfig greN tunnel A B
   # route add -inet6 2001:db8:ffff::/64 2001:db8:ffff::2 -ifp greN
   # route add -inet6 0::0/0 2001:db8:ffff::2 -ifp greN

On router B (Cisco):

   Interface TunnelX
     tunnel mode gre ip
     ipv6 address 2001:db8:ffff::2/64   ! transfer network
     tunnel source B                    ! e.g. address from LAN interface
     tunnel destination A               ! where the tunnel is connected to
   ipv6 route 2001:db8::/64 TunnelX     ! route this network through tunnel
```

NOTES

The MTU of **gre***X* interfaces is set to 1476 by default to match the value used by Cisco routers. This may not be an optimal value, depending on the link between the two tunnel endpoints. It can be adjusted via `ifconfig`(8).

There needs to be a route to the decapsulating host that does not run over the tunnel, as this would be a loop. (This is not relevant for IPv6-over-IPv4 tunnels, of course.)

In order to tell `ifconfig`(8) to actually mark the interface as up, the keyword “up” must be given last on its command line.

The kernel must be set to forward datagrams by either option *GATEWAY* in the kernel config file or by issuing the appropriate option to `sysctl`(8).

SEE ALSO

`atalk`(4), `gif`(4), `inet`(4), `ip`(4), `netintro`(4), `options`(4), `protocols`(5), `ifconfig`(8), `sysctl`(8)

A description of GRE encapsulation can be found in RFC 1701 and RFC 1702.

A description of MOBILE encapsulation can be found in RFC 2004.

AUTHORS

Heiko W.Rupp ⟨hwr@pilhuhn.de⟩
David Young ⟨dyoung@NetBSD.org⟩ (GRE in UDP encapsulation, bug fixes)

BUGS

The GRE RFCs are not yet fully implemented (no GRE options).

The MOBILE encapsulation appears to have been broken since it was first added to NetBSD, until August 2006. It is known to interoperate with another **`gre`** in MOBILE mode, however, it has not been tested for interoperability with any other implementation of RFC 2004.

The NetBSD base system does not (yet) contain a daemon for automatically establishing a UDP tunnel between a host behind a NAT router and a host on the Internet.

NAME

grf — Amiga graphics frame buffer device

SYNOPSIS

```
grf0 at grfcc0
grf1 at grfrt0
grf2 at grfrh0
grf3 at grfcl?
grf4 at grful0
grf5 at grfcv0
grf6 at grfet?
grf7 at grfcv3d0
```

DESCRIPTION

This is frame buffer device for Amiga. There is fixed mapping to the common video hardware. If the device is configured, the frame buffers are mapped as follows:

grf0	custom chips
grf1 & grf2	Retina Z2/Z3
grf3	Cirrus boards
grf4	A2410 video
grf5	CyberVision 64
grf6	ET4000 boards (Tseng - oMniBus, Domino, Merlin)
grf7	CyberVision 64/3D

SEE ALSO

`grfcl`(4), `grfcv`(4), `grfcv3d`(4), `grfet`(4), `grfrh`(4), `grfrt`(4), `grful`(4), `hp300/grf`(4), `ite`(4)

BUGS

This manpage should provide programming details.

NAME

grf — HP graphics frame buffer device interface

SYNOPSIS

```
grf* at dvbox?
grf* at gbox?
grf* at hyper?
grf* at rbox?
grf* at topcat?
```

DESCRIPTION

This is a generic description of the frame buffer device interface. The devices to which this applies are the 98544, 98545 and 98547 Topcat display cards (also known as HP300H devices), the 98548, 98549 and 98550 Catseye display cards, the 98700 Gatorbox graphics box, the 98720 Renaissance graphics box, and the 98730 DaVinci graphics box.

Use of the devices can be effectively approached from two directions. The first is through HP-UX *Starbase* routines, the second is by direct control in the BSD environment. In order to use the Starbase libraries, code must be compiled in an HP-UX environment, either by doing so on an HP-UX machine and transferring the binaries to the BSD machine, or by compilation with the use of the **hpux** command. Applications using Starbase libraries have been run successfully on BSD machines using both of these compilation techniques.

Direct compilation, such as that used for the X Window System servers, has also been successful. Examples of some frame buffer operations can be found in the device dependent X Window system sources, for example the `/usr/src/new/X/libhp.fb` directory. These files contain examples of device dependent color map initialization, frame buffer operations, bit moving routines etc.

The basic programming of the **grf**? devices involves opening the device file, mapping the control registers and frame buffer addresses into user space, and then manipulating the device as the application requires. The address mapping is controlled by an `ioctl`(2) call to map the device into user space, and an unmap call when finished. The ioctls supported by BSD are:

`GRFIOCGINFO`

Get Graphics Info

Get info about device, setting the entries in the *grfinfo* structure, as defined in <hpdev/grfioctl.h>:

```
struct  grfinfo {
        int     gd_id;          /* HPUX identifier */
        caddr_t gd_regaddr;     /* control registers physaddr */
        int     gd_regsize;     /* control registers size */
        caddr_t gd_fbaddr;      /* frame buffer physaddr */
        int     gd_fbsize;      /* frame buffer size */
        short   gd_colors;      /* number of colors */
        short   gd_planes;      /* number of planes */
/* new stuff */
        int     gd_fbwidth;     /* frame buffer width */
        int     gd_fbheight;    /* frame buffer height */
        int     gd_dwidth;      /* displayed part width */
        int     gd_dheight;     /* displayed part height */
        int     gd_pad[6];      /* for future expansion */
};
```

`GRFIOCON`
: Graphics On

  Turn graphics on by enabling CRT output. The screen will come on, displaying whatever is in the frame buffer, using whatever colormap is in place.

`GRFIOCOFF`
: Graphics Off

  Turn graphics off by disabling output to the CRT. The frame buffer contents are not affected.

`GRFIOCMAP`
: Map Device to user space

  Map in control registers and framebuffer space. Once the device file is mapped, the frame buffer structure is accessible.

`GRFIOCUNMAP`
: Unmap Device

  Unmap control registers and framebuffer space.

For further information about the use of ioctl see the man page.

FILES

`/dev/grf?` BSD interface special files
`/dev/*crt*` HP-UX *starbase* interface special files

EXAMPLES

This short code fragment is an example of opening some graphics device and mapping in the control and frame buffer space:

```
#define GRF_DEV <some_graphics_device>  /* /dev/grfN */
{
        struct fbstruct *regs;  /*  fbstruct = gboxfb, rboxfb, etc. */
        u_char *Addr, frame_buffer;
        struct grfinfo gi;
        int disp_fd;

        disp_fd = open(GRF_DEV,1);
        if (ioctl (disp_fd, GRFIOCGINFO, &gi) < 0) return -1;
        (void) ioctl (disp_fd, GRFIOCON, 0);

        Addr = (u_char *) 0;
        if (ioctl (disp_fd, GRFIOCMAP, &Addr) < 0) {
                (void) ioctl (disp_fd, GRFIOCOFF, 0);
                return -1;
        }
        regs = (fbstruct *) Addr;               /* Control Registers   */
        frame_buffer = (u_char *) Addr + gi.gd_regsize; /* Frame buffer mem */
}
```

DIAGNOSTICS

None under BSD. HP-UX The CE.utilities/Crtadjust programs must be used for each specific device.

ERRORS

[ENODEV] no such device.

[EBUSY] Another process has the device open.

[EINVAL] Invalid ioctl specification.

SEE ALSO

ioctl(2), dvbox(4), gbox(4), hil(4), hyper(4), rbox(4), topcat(4)

NAME

grfcl — 8/16/24/32-bit color graphics driver

SYNOPSIS

```
grfcl* at zbus0
grf3 at grfcl?
ite3 at grf3
```

DESCRIPTION

The **grfcl** device driver supports graphics boards with the Cirrus Logic chipset. It supports the minimal ioctl's needed to run X11.

HARDWARE

The **grfcl** interface supports the following ZorroII/III expansion cards:

Picasso II/II+ A Zorro II only card with the Cirrus Logic 5426, 5428 or 5429 chipset. From Village Tronic, manufacturer 2167, product 11 for the memory base and 12 for the register base (if your board has a product of 13, it is in segmented mode which is not supported by NetBSD). This board supports 8/15/16/24 bit graphics modes.

Picasso IV A Zorro II/III card with the Cirrus Logic 5446 chipset. From Village Tronic, manufacturer 2167, in Zorro II mode with the product 21 for the memory base1, 22 for the memory base2 and 23 for the register base, in Zorro III mode with the product 24 for the entire board. This board supports 8/15/16/24 bit graphics modes.

Piccolo A Zorro II/III card with the Cirrus Logic 5426 chipset. From IBH, manufacturer 2195, product 5 for the memory base and 6 for the register base. This board supports 8/15/16/24 bit graphics modes.

Piccolo SD64 A Zorro II/III card with the Cirrus Logic 5434 chipset. From IBH, manufacturer 2195, product 10 for the memory base and 11 for the register base. This board supports 8/15/16/24/32 bit graphics modes.

Spectrum A Zorro II/III card with the Cirrus Logic 5426 chipset. From GVP, manufacturer 2193, product 1 for the memory base and 2 for the register base. This board supports 8/15/16/24 bit graphics modes.

FILES

`/dev/grf3` graphics interface special file
`/dev/ttye3` console interface special file for the internal terminal emulator

SEE ALSO

`console`(4), `ite`(4), `grfconfig`(8)

HISTORY

The **grfcl** interface first appeared in NetBSD 1.0.

BUGS

grfcl will not work with Zorro II cards in the segmented mode and it does not support the sync-on-green flag from `grfconfig`(8).

NAME

grfcv — 8/15/16/24/32-bit color graphics driver

SYNOPSIS

```
grfcv0 at zbus0
grf5 at grfcv0
ite5 at grf5
```

DESCRIPTION

The **grfcv** device driver supports graphics boards with the S3 Trio64 chipset. It supports the minimal ioctl's needed to run X11.

HARDWARE

The **grfcv** interface supports the following ZorroIII expansion card:

CyberVision 64 A Zorro III only card. From phase5, manufacturer 8512, product 34.

FILES

`/dev/grf5` graphics interface special file
`/dev/ttye5` console interface special file for the internal terminal emulator

SEE ALSO

`console`(4), `grfcv3d`(4), `ite`(4), `grfconfig`(8)

HISTORY

The **grfcv** interface first appeared in NetBSD 1.1.

BUGS

grfcv does not support the sync-on-green flag from `grfconfig`(8).

NAME

grfcv3d — 8/15/16/24/32-bit color graphics driver

SYNOPSIS

```
grfcv3d0 at zbus0
grf7 at grfcv3d0
ite7 at grf7
```

DESCRIPTION

The **grfcv3d** device driver supports graphics boards with the S3 Virge chipset. It supports the minimal ioctl's needed to run X11.

HARDWARE

The **grfcv3d** interface supports the following ZorroII/III expansion card:

CyberVision 64/3D A Zorro II/III card. From phase5, manufacturer 8512, product 67.

FILES

`/dev/grf7` graphics interface special file
`/dev/ttye7` console interface special file for the internal terminal emulator

SEE ALSO

`console`(4), `grfcv`(4), `ite`(4), `grfconfig`(8)

HISTORY

The **grfcv3d** interface first appeared in NetBSD 1.3.

BUGS

grfcv3d does currently only work in Zorro III mode and does not support the sync-on-green flag from `grfconfig`(8).

NAME

grfet — 8/16/24/32-bit color graphics driver

SYNOPSIS

```
grfet* at zbus0
grf6 at grfet?
ite6 at grf6
```

DESCRIPTION

The **grfet** device driver supports graphics boards with the Tseng ET4000 or ET4000W32 chipset and a Sierra 11483 or Brooktree BT482 DAC. It supports the minimal ioctl's needed to run X11.

HARDWARE

The **grfet** interface supports the following ZorroII/III expansion cards:

Domino A Zorro II only card with the Tseng ET4000 chipset and a Sierra 11483 DAC. From Village Tronic, manufacturer 2167, product 1 for the memory base and 2 for the register base.

Merlin A Zorro II/III card with the Tseng ET4000W32 chipset and a Brooktree BT482 DAC. From Expert, manufacturer 2117, with the product 3 for the memory base and 4 for the register base.

oMnibus A Zorro II only Zorro/AT bridge board with a separate VGA card usually with a Tseng ET4000 chipset and a Sierra 11483 DAC, but a few others are also supported. Once made by Oliver Bausch and sold by Armax, manufacturer 2181, product 0 for the memory base and also 0 for the register base.

FILES

`/dev/grf6` graphics interface special file
`/dev/ttye6` console interface special file for the internal terminal emulator

SEE ALSO

`console`(4), `ite`(4), `grfconfig`(8)

HISTORY

The **grfet** interface first appeared in NetBSD 1.2.

BUGS

grfet does not support the sync-on-green flag from `grfconfig`(8).

NAME

grfrh — 8/16/24-bit color graphics driver

SYNOPSIS

```
grfrh0 at zbus0
grf2 at grfrh0
ite2 at grf2
```

DESCRIPTION

The **grfrh** device driver supports graphics boards with the NCR 77C32BLT chipset. It supports the minimal ioctl's needed to run X11.

HARDWARE

The **grfrh** interface supports the following expansion cards:

Retina BLT Z3	A Zorro III only card. From Macro System, manufacturer 18260, product 16.
Altais	A DraCo local bus only card. From Macro System, manufacturer 18260, product 19.

FILES

`/dev/grf2`	graphics interface special file
`/dev/ttye2`	console interface special file for the internal terminal emulator

SEE ALSO

`console`(4), `ite`(4), `grfconfig`(8)

HISTORY

The **grfrh** interface first appeared in NetBSD 1.0.

BUGS

grfrh does not allow setting graphics modes with `grfconfig`(8).

NAME

grfrt — 8-bit color graphics driver

SYNOPSIS

```
grfrt0 at zbus0
grf1 at grfrt0
ite1 at grf1
```

DESCRIPTION

The **grfrt** device driver supports graphics boards with the NCR 77C22E+ chipset. It supports the minimal ioctl's needed to run X11.

HARDWARE

The **grfrt** interface supports the following ZorroII expansion card:

Retina Z2 A Zorro II only card. From Macro System, manufacturer 18260, product 6.

FILES

`/dev/grf1` graphics interface special file
`/dev/ttye1` console interface special file for the internal terminal emulator

SEE ALSO

`console`(4), `ite`(4), `grfconfig`(8)

HISTORY

The **grfrt** interface first appeared in NetBSD 1.0.

BUGS

grfrt does not allow setting graphics modes with `grfconfig`(8).

NAME

grful — 8-bit color graphics driver

SYNOPSIS

```
grful* at zbus0
grf4 at grful?
ite4 at grf4
```

DESCRIPTION

The **grful** is a driver for the A2410, a TMS34010 based color graphics board. It supports the minimal ioctl's needed to run X11, but doesn't provide a mappable framebuffer, so special X servers are needed.
A standard ITE terminal emulator is supported on one of the overlay planes.

HARDWARE

The **grful** interface supports the following ZorroII expansion card:

A2410 A Zorro II only card with the TMS34010 processor. From the University of Lowell, manufacturer 1030, product 0.

FILES

`/dev/grf4`	graphics interface special file
`/dev/grfim4`	graphics interface special file for accessing images
`/dev/grfov4`	graphics interface special file for overlays
`/dev/ttye4`	console interface special file for the internal terminal emulator

SEE ALSO

`console`(4), `ite`(4), `grfconfig`(8)

HISTORY

The **grful** interface first appeared in NetBSD 1.1.

BUGS

grful does not allow setting graphics modes with `grfconfig`(8).

NAME

grtwo — SGI GR2 graphics controller

SYNOPSIS

```
grtwo* at gio? slot ?
wsdisplay* at grtwo? console ?
```

DESCRIPTION

The **grtwo** driver supports the SGI GR2 series of graphics controllers, which are found on Indigo, Crimson, and some Personal Iris series machines.

SEE ALSO

gio(4), light(4), newport(4), wscons(4)

HISTORY

The **grtwo** driver first appeared in NetBSD 2.0.

AUTHORS

Christopher SEKIYA wrote this driver.

BUGS

This driver has not been extensively tested on the many different GR2 series offerings. It is unlikely to run without modification on Crimson machines.

NAME

gsc — General System Connect bus on HP 9000/700 machines

SYNOPSIS

```
gsc* at lasi?
gsc* at asp?
gsc* at wax?
```

DESCRIPTION

This is the core I/O bus for all HP 9000/700 workstations. All I/O subsystems connect to this bus. The devices can be either on separate chips, expansion cards or on an integrated megacell, like the LASI MBA.

The GSC bus is a 32-bit wide, address and data multiplexed bus. In its "standard" implementation it has a maximum throughput of 160MB/s, the "2x" implementation reaches up to 250MB/s. Some HPPA CPUs directly attach to this bus, namely the PA7100LC and PA7300LC.

As for the expansion cards, there are different form-factors, depending on bus-speed (standard or 2x) and specific models. The standard formfactor is the "EISA form-factor"; cards that look like typical EISA cards with a different connector (100-pin female EBBL). The Series 712 have their own special type of GSC expansion cards, called the "GIO form-factor", which is quite small and mostly has only one VLSI chip on it (in most cases LASI/WAX). Newer systems sometimes feature the "HSC formfactor", which is a 1U-VME card-like expansion card with a 100-pin male pin+socket connector. Mixing cards with different speeds is supported but downgrades the performance of the whole I/O-subsystem.

SUPPORTED DEVICES

The system includes machine-dependent GSC drivers, sorted by driver name:

com(4)	RS-232 ports.
gsckbc(4)	PC-style keyboard controller.
harmony(4)	CS4215/AD1849 audio.
hil(4)	Human Interface Loop, for HP-proprietary input devices like keyboard and mice.
iee(4)	Intel i82596 DX/CA 32-bit LAN coprocessor.
lpt(4)	Centronics printer port.
oosiop(4)	Symbios/NCR 53C700 SCSI I/O Processor.
osiop(4)	Symbios/NCR 53C710 SCSI I/O Processor.

Some of these GSC devices also have PCI, EISA, or ISA equivalents. These are listed in pci(4), eisa(4), or isa(4),

SEE ALSO

asp(4), cpu(4), intro(4), lasi(4), wax(4)

HISTORY

The **gsc** driver appeared in OpenBSD 2.6. It was ported to NetBSD 1.6 by Matthew Fredette.

NAME

gsckbc — GSC PS/2 keyboard and mouse interface

SYNOPSIS

```
gsckbc* at gsc?
pckbd*  at gsckbc?
pms*    at gsckbc?
```

DESCRIPTION

The **gsckbc** device is a machine dependent front end to the pckbport(9) interface. It attaches to the GSC PS/2 keyboard and mouse interface found in LASI chips.

DIAGNOSTICS

gsckbc_attach: can't map I/O space The driver was not able to map the device registers during attachment. The device will not be usable.

can't find master device An error occured during attachment of the keyboard port so the mouse port can't be attached too.

SEE ALSO

gsc(4), intro(4), io(4), lasi(4), pckbport(9)

HISTORY

The **gsckbc** driver appeared in NetBSD 2.0.

AUTHORS

Jochen Kunz

BUGS

Actually the two PS/2 ports are a single device and share a single interrupt. The firmware lists them as individual devices in the firmware device tree. This illusion is kept to map the firmware device tree as close as possible to the kernel device tree. The first device is caled master, gets the interrupt and the other is the slave. Assumption: Master attaches first, gets the interrupt and has lower HPA. So it is important that the master device, usually the keyboard port, attaches first to make the slave, usually the mouse port, usable.

NAME

gscpcib — National Semiconductor Geode SC1100 PCI-ISA bridge

SYNOPSIS

```
gscpcib* at pci?
isa* at gscpcib?
gpio* at gscpcib?
```

DESCRIPTION

The **gscpcib** driver provides support for the National Semiconductor Geode SC1100 System-on-Chip PCI-ISA bridge. This device is a PCI-ISA bridge, but with some additional sub-devices. Besides the core pcib(4) functionality, the **gscpcib** driver provides support for the GPIO interface of this device.

The device has 64 I/O pins which can be accessed through the gpio(4) framework. The gpioctl(8) program provides a convenient way to manipulate these pins from userland.

SEE ALSO

gpio(4), intro(4), isa(4), pci(4), pcib(4), gpioctl(8)

HISTORY

The **gscpcib** driver first appeared in OpenBSD 3.6 and NetBSD 4.0.

AUTHORS

The **gscpcib** driver was written by Alexander Yurchenko ⟨grange@openbsd.org⟩ and was ported to NetBSD by Jared D. McNeill ⟨jmcneill@NetBSD.org⟩.

NAME

gsip — National Semiconductor DP83820 Gigabit Ethernet driver

SYNOPSIS

gsip* **at pci? dev ? function ?**

Configuration of PHYs may also be necessary. See mii(4).

DESCRIPTION

The **gsip** device driver supports Gigabit Ethernet interfaces based on the National Semiconductor DP83820 Gigabit Ethernet chips.

The National Semiconductor DP83820 is found on NetGear GA-622, Asante FriendlyNet GigaNIX, D-Link DGE-500T, SMC 9452TX and 9462TX, Accton EN1407-T, Planex GN-1000TE, ARK SOHO GA2000T and GA2500T, and other low-cost Gigabit Ethernet cards. It uses an external PHY or an external 10-bit interface.

The DP83820 supports VLAN tag insertion/removal in hardware. The **gsip** driver supports this feature of the chip.

The DP83820 supports IPv4/TCP/UDP checksumming in hardware. The **gsip** driver supports this feature of the chip. See ifconfig(8) for information on how to enable this feature.

The DP83820 chip is a close relative of the DP83815 10/100 Ethernet chip, which is supported by the sip(4) driver, hence the **gsip** name.

SEE ALSO

arp(4), ifmedia(4), mii(4), netintro(4), pci(4), vlan(4), ifconfig(8)

HISTORY

The **gsip** driver first appeared in NetBSD 1.6.

AUTHORS

The **gsip** driver was written by Jason R. Thorpe ⟨thorpej@NetBSD.org⟩.

BUGS

The **gsip** driver does not support the 10-bit interface, which is required in order to support fiber-optic media.

NAME

gt — PCI bridge

SYNOPSIS

```
gt* at mainbus? addr 0x14000000
pci* at gt?
```

DESCRIPTION

The **gt** driver provides support for the PCI bridge.

SEE ALSO

intro(4), pci(4)

NAME

gtsc — GVP low level SCSI interface

SYNOPSIS

```
gtsc0 at gvpbus0
```

DESCRIPTION

The Amiga architecture uses a common machine independent scsi sub-system provided in the kernel source. The machine independent drivers that use this code access the hardware through a common interface. (see scsibus(4)) This common interface interacts with a machine dependent interface, such as **gtsc**, which then handles the hardware specific issues.

The **gtsc** interface handles things such as DMA and interrupts as well as actually sending commands, negotiating synchronous or asynchronous transfers and handling disconnect/reconnect of SCSI targets. The hardware that **gtsc** uses is based on the WD33c93 SCSI chip.

DIAGNOSTICS

sbicwait TIMEO @%d with asr=x%x csr=x%x The 33c93 code (sbic) has been waiting too long for a SCSI chip operation to complete. %d is the line in the source file amiga/dev/sbic.c at which the SCSI chip timed-out. Asr and csr are status registers within the SCSI chip.

gtsc%d: abort %s: csr = 0x%02x, asr = 0x%02x A SCSI operation %s was aborted due to an error.

gtsc%d: csr == 0x%02i A error has occurred within the SCSI chip code.

gtsc%d: unexpected phase %d in icmd from %d The target described by ‘from %d’ has taken the SCSI bus into a phase which is not expected during polled IO.

gtsc%d: unexpected phase %d in icmd from %d The target described by ‘from %d’ has taken the SCSI bus into a phase which is not expected during DMA IO setup.

SEE ALSO

scsibus(4)

HISTORY

The **gtsc** interface first appeared in NetBSD 1.0

NAME

gus — Gravis UltraSound/UltraSound MAX audio device driver

SYNOPSIS

```
gus0 at isa? port 0xPPP irq X drq Y drq2 Z
audio* at audiobus?
```

DESCRIPTION

The **gus** driver provides support for the Gravis UltraSound (GUS) and GUS MAX audio cards. Both cards have on-board memory which is used for seamless playback of samples. They can play back 8- or 16-bit samples at up to 44.1kHz. They can record 8-bit samples at up to 44.1kHz. The UltraSound MAX is a full-duplex sound device, and if configured with two DRQ channels can be used for simultaneous playback and recording. The I/O port base is jumper-selected, and may be chosen from 0x210-0x260 in steps of 0x10. (The normal setting is 0x220.) The GUS takes 16 ports at its base address and 8 ports at its base address + 0x100.

The IRQ is software programmed, so you may select any IRQ from the set {3,5,7,9,11,12,15}. The DRQ lines are software programmed, and may be chosen from {1,3,5,6,7}. The drq2 field in the configuration file line specifies a second DRQ line for recording. If there is no drq2 field in the config file, the playback channel will be used for recording DMA and only half-duplex mode will be available.

The Gravis UltraSound MAX has an additional CODEC onboard which is addressed with four ports at an offset of 0x10C from the base ports (0x31C-0x36C).

SEE ALSO

audio(4)

REFERENCES

Gravis UltraSound Low-Level Toolkit, Revision 2.01, 20 May 1993, published by Advanced Gravis and Forte Technologies.

HISTORY

The **gus** device driver appeared in NetBSD 1.1.

BUGS

The full-duplex features of the GUS MAX have not been fully tested, and full-duplex on the original GUS may not be possible at all.

Only two voices on the GF1 synthesizer chip are used by this driver (for left and right channels).

Manipulating the mixer while audio samples are playing can lead to device driver confusion (and maybe even a system panic).

Manipulating the mixer device seems to create pregnant system pauses, probably due to excessive interrupt masking.

The joystick and MIDI port interfaces are not supported.

NAME

guspnp — Am78C201 audio device driver

SYNOPSIS

```
guspnp* at isapnp?
audio*  at audiobus?
```

There should be no limit caused by the driver on the number of drivers or cards active in the system.

DESCRIPTION

The **guspnp** driver provides support for audio subsystems using the Interwave (Am78C20x) family of ICs, usually the Gravis Ultrasound Plug and Play. Unlike the **gus** driver guspnp driver does not require any local memory for the IC, but uses the codec for both playback and recording. The **guspnp** driver can simultaneously playback and record 8- and 16-bit samples at frequencies from 5.51kHz to 48kHz.

The **guspnp** driver relies on **isapnp** to allocate suitable resources for it. This version of the driver only uses the first logical device of the five the Interwave IC has. The four unused logical devices are the ATAPI CD-ROM device, PnP Joystick device, legacy soundcard emulation device (SoundBlaster) and MIDI serial device. Support for at least ATAPI CD-ROM and Joystick is being worked on. This version of the driver will use 1 IRQ and 2 DRQs.

HARDWARE

Cards supported by the **guspnp** driver include:

Gravis Ultrasound PNP, and compatibles

SEE ALSO

audio(4), gus(4), isapnp(4)

REFERENCES

Interwave(tm) IC Am78C201/202 Programmer's Guide Rev. 2. 1996. Advanced Micro Devices.

HISTORY

The **guspnp** driver appeared in NetBSD 1.3.

AUTHORS

Kari Mettinen ⟨Kari.Mettinen@helsinki.fi⟩, University of Helsinki.

BUGS

Sometimes you can cause a hiss on either left or right channel, or both. You can usually make it disappear by playing random data, however this might not be a very nice thing to your audio equipment, but it is the only way I have found out to be effective.

Only the Codec is used in this version of the driver, therefore only 2 channels are supported (left and right). Also sound quality is probably worse at lower kHz compared to playing through the synthesizer which does interpolation.

If the implementation has a 'bad' oscillator, using frequencies 44.8kHz and 38.4kHz will result in incorrect playback frequency. The author has a GUS PnP Pro which displays this behavior.

Other members of the Interwave family have not been tested and don't have the glue needed to make them work. Should someone need to implement it, not many changes in the existing code are needed. Output voltage control in register CFIG2 [7] should be set differently for some other members of the family.

Other architectures than i386 haven’t been tested. The bus_space abstraction has been used from the beginning, so it should work.

NAME

haltwo — SGI HAL2 audio controller

SYNOPSIS

```
haltwo0 at hpc0 offset ?
```

DESCRIPTION

haltwo is the audio controller found on the Indy and Indigo2 machines.

SEE ALSO

audio(4), hpc(4)

HISTORY

The **haltwo** driver first appeared in NetBSD 2.0.

CAVEATS

The driver lacks support for most mixer operations and recording.

NAME

harmony — CS4215/AD1849 audio interface

SYNOPSIS

```
harmony* at gsc?
audio*   at harmony?
```

DESCRIPTION

The **harmony** device uses the Crystal Semiconductor CS4215 16-Bit Multimedia Audio Codec or Analog Devices AD1849 SoundPort(R) Stereo Codec chip to implement the audio device interface described in audio(4). This device is found on most HP PA-RISC workstations. The **harmony** has a maximum precision of 16 bits and has a stereo input and stereo output.

On HP 9000/712 models **harmony** also provides two additional channels for an add-on card with two fax/voice modems.

One of the hardware registers reflects the state of the CHI bus that is used to communicate with the codec and thus being sampled at a low accuracy secondary frequency (such as timeout(9)) produces poor quality random bit stream that is fed into the entropy pool of rnd(4).

MACHINES

An incomplete list of machines that feature **harmony** audio:

- 712/*
- 715/*
- 725/*
- 735/*
- 755/*
- B132L[+], B160L, B180L+
- C100, C110, C132L, C160[L], C180, C200, C240, C360
- J200, J210[XC], J280, J282, J2240

SEE ALSO

ioctl(2), audio(4), gsc(4), intro(4), rnd(4)

HISTORY

Support for **harmony** first appeared in OpenBSD 3.3. It was ported to NetBSD 1.6 by Chuck Silvers.

CAVEATS

To trigger entropy collection CHI bus has to be programmed into the data mode that happens once a single buffer of data has been played or recorded.

NAME

dtide — HCCS IDE interface driver

SYNOPSIS

```
hcide* at podulebus0 slot ?
ata* at hcide? channel ?
```

DESCRIPTION

The **dtide** driver handles an HCCS IDE interface plugged into an Acorn expansion slot. It uses the standard NetBSD IDE controller driver, and hence can use the standard **atabus** driver.

The card provides three IDE channels. The internal 44-way IDE connector is channel 0, the internal 40-way connector is channel 1 and the external connector is channel 2.

SEE ALSO

atabus(4), atapibus(4), podulebus(4), wd(4)

BUGS

The driver was derived by reverse-engineering the card, so it may not handle the card entirely optimally.

NAME

hdh — ACC IF-11/HDH IMP network interface

SYNOPSIS

```
pseudo-device imp
hdh0 at uba0 csr 166740 vector hdhintr
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

NOTE: At the moment, NetBSD does not support IMP, so this manual page is not relevant.

The **hdh** device provides an HDLC Host (HDH) interface to an IMP. It is normally used when participating in the DARPA Internet. The controller itself is not accessible to users, but instead provides the hardware support to the IMP interface described in `imp`(4). The configuration entry for the IMP must also include the *pseudo-device* as shown above in the **SYNOPSIS**.

DIAGNOSTICS

hdh%d: not alive. The initialization routine was entered even though the device did not autoconfigure. This indicates a system problem.

hdh%d: cannot get chan %d uba resources. Insufficient UNIBUS resources existed to initialize the device. This is likely to be a shortage of UNIBUS mapping registers.

hdh%d: LINE UP. This indicates that both the HDLC and HDH protocols have declared the link to the IMP alive.

hdh%d: LINE DOWN. This indicates that the link to the IMP has died.

hdh%d: TIMEOUT.
hdh%d: HOST DATA ERROR.
hdh%d: IMP SEQUENCE ERROR.
hdh%d: HOST SEQUENCE ERROR. These errors indicate that an HDH protocol error has been detected.

hdh%d: cannot get supervisor cmnd buffer. This error indicates that an *mbuf* could not be allocated to send a command to the IF-11/HDH.

Any other error message from hdh%d: indicates a serious error detected by either the driver or the IF-11/HDH firmware.

SEE ALSO

`netintro`(4)

HISTORY

The **hdh** driver appeared in 4.3BSD.

NAME

hifn — Hifn 7751/7951/7811/7955/7956 crypto accelerator

SYNOPSIS

hifn* **at pci? dev ? function ?**

DESCRIPTION

The **hifn** driver supports various cards containing the Hifn 7751, 7951, 7811, 7955, and 7956 chipsets, such as

Invertex AEON	No longer being made. Came as 128KB SRAM model, or 2MB DRAM model.
Hifn 7751	Reference board with 512KB SRAM.
PowerCrypt	See `http://www.powercrypt.com/`. Comes with 512KB SRAM.
XL-Crypt	See `http://www.powercrypt.com/`. Only board based on 7811 (which is faster than 7751 and has a random number generator).
NetSec 7751	See `http://www.netsec.net/`. Supports the most IPsec sessions, with 1MB SRAM.
Soekris Engineering vpn1201 and vpn1211	See `http://www.soekris.com/`. Contains a 7951 and supports symmetric and random number operations.
Soekris Engineering vpn1401 and vpn1411	See `http://www.soekris.com/`. Contains a 7955 and supports symmetric and random number operations.

The **hifn** driver registers itself to accelerate DES, Triple-DES, AES (7955 and 7956 only), ARC4, MD5, MD5-HMAC, SHA1, and SHA1-HMAC operations for `opencrypto`(9), and thus for `fast_ipsec`(4) and `crypto`(4).

The Hifn 7951, 7811, 7955, and 7956 may also supply data to the kernel `rnd`(4) subsystem.

SEE ALSO

`crypto`(4), `fast_ipsec`(4), `intro`(4), `rnd`(4), `opencrypto`(9)

HISTORY

The **hifn** device driver appeared in OpenBSD 2.7. The **hifn** device driver was imported to FreeBSD 5.0, back-ported to FreeBSD 4.8, and subsequently imported into NetBSD 2.0.

CAVEATS

The Hifn 9751 shares the same PCI ID. This chip is basically a 7751, but with the cryptographic functions missing. Instead, the 9751 is only capable of doing compression. Since we do not currently attempt to use any of these chips to do compression, the 9751-based cards are not useful.

Support for the 7955 and 7956 is incomplete; the asymmetric crypto facilities are to be added and the performance is suboptimal.

Supplying data to the kernel `rnd`(4) subsystem has been disabled, pending verification that the on-chip RNG is statistically adequate.

BUGS

The 7751 chip starts out at initialization by only supporting compression. A proprietary algorithm, which has been reverse engineered, is required to unlock the cryptographic functionality of the chip. It is possible for vendors to make boards which have a lock ID not known to the driver, but all vendors currently just use the obvious ID which is 13 bytes of 0.

NAME

hil — Human Interface Link device driver

SYNOPSIS

hil* **at intio?**

DESCRIPTION

The Human Interface Link (HIL) is the interface used by the Series 300 computers to connect devices such as keyboards, mice, control knobs, and ID modules to the machine.

Special files `/dev/hil[1-7]` refer to physical HIL devices 1 through 7. `/dev/hil0` is an artifact of a never-completed interface and is not currently useful for anything. In the current implementation, only one keyboard can be used for text-mode interaction via the `ite`(4) interface; other keyboards, if any, are accessible only via their `/dev/hil`*N* interfaces, as described here.

The device file that corresponds to a particular HIL device is determined by the order of the devices on the loop. For instance, if an ID module is the second physical device on the loop, then `/dev/hil2` is the special file that should be used for communication with that module.

Communication with an HIL device is begun with an *open* system call. A process may open a device already opened by another process unless the existing process is operating in HP-UX compatibility mode, in which case it requires exclusive use of the device, or another process has the device open and is using HP-UX style device access (see `HILIOCHPUX` below).

In the current implementation, HP-UX style access is the only supported access method. This interface uses `read`(2) calls to receive packets of data representing events. (An interface using a memory area shared between the kernel and the user process was partially implemented, and remnants of it can be found in the include files and the driver, but it does not work and probably will be completely dropped in favor of a console interface providing better cross-port consistency.)

To receive events from a device, a user process uses `open`(2) to open the device, then uses the `HILIOCHPUX` ioctl (see below) to request HP-UX style access. Then data obtained with `read`(2) consists of a stream of packets, each of which has a five-byte header consisting of a one-byte length (including the header) and a four-byte timestamp, which is measured in hundredths of a second since some fixed reference point. The timestamp is stored in host-native byte order; copying it into a ‘long int’ variable with `memcpy`(3) or equivalent will give a useful result. Following this header is zero or more bytes, as received from the device. This manual page documents this data only minimally; see PACKET FORMAT below.

`select`(2) may be used in the usual way to detect input data.

`ioctl`(2) is used to control the HIL device. The ioctl commands (which unfortunately are defined in an include file, `/usr/src/sys/arch/hp300/dev/hilioctl.h`, not normally installed under `/usr/include`), are as follows. Many of these are functionally identical to ioctls HP-UX provides.

`HILIOCID` Identify and Describe

The device will return up to 11 bytes of information describing the type and characteristics of the device. At the very least, 2 bytes of information, the device ID, and the Describe Record Header will be returned. Identical to the HP-UX `HILID` ioctl.

`HILIOCSC` Report Security Code

Request the security code record from a device. The security code can vary from 1 byte to 15, and is only supported by some HIL devices. Identical to the HP-UX `HILSC` ioctl.

`HILIOCRN` Report Name

An ascii string of up to 15 bytes in length that describes the device is returned. Identical to the HP-UX `HILRN` ioctl.

`HILIOCRS` Report Status

An ascii string of up to 15 bytes in length that describes the current status of the device is returned. Identical to the HP-UX `HILRS` ioctl.

`HILIOCED` Extended Describe

Additional information of up to 15 bytes is returned describing the device. This ioctl is similar to HILIOCID, which must be used first to determine if the device supports extended describe. Identical to the HP-UX `HILED` ioctl.

`HILIOCAROFF`
Disable Auto Repeat

Turn off auto repeat on the keyboard while it is in cooked mode. Identical to the HP-UX `HILDKR` ioctl.

`HILIOCAR1` Enable Auto Repeat

Turn on auto repeat on the keyboard while it is in raw mode. The repeat rate is set to 1/30th of a second. Identical to the HP-UX `HILER1` ioctl.

`HILIOCAR2` Enable Auto Repeat

Turn on auto repeat on the keyboard while it is in raw mode. The repeat rate is set to 1/60th of a second. Identical to the HP-UX `HILER2` ioctl.

The following ioctls are specific to this implementation:

`HILIOCBEEP`
Beep

Generate a keyboard beep as defined by *`arg`*. *`arg`* is a pointer to two bytes of information; the first is the duration of the beep (microseconds), the second is the frequency of the beep (Hertz).

`HILIOCHPUX`
Use HP-UX Read Interface

Use HP-UX semantics for gathering data from this device. This call must be used before anything can be read from the descriptor.

PACKET FORMAT

When reading data from a device, events are received as data packets, with a header as described above. Here we provide (very rudimentary) documentation for the format of the device-dependent data, for at least one type of mouse and keyboard (specifically, the 46060A two-button mechanical mouse and 46021A keyboard); other mice and keyboards may or may not be similar.

The first byte of data is a bitmask. Only two bits have known meaning:

`0x02` The data portion contains mouse motion deltas, two signed 8-bit quantities, X delta first.

`0x40` The data portion contains a key or mouse button up/down event, one byte. The low bit is 0 if the event is a ‘down’ event, 1 if an ‘up’ event. The other seven bits identify the key or button, according to the table below.

If both bits are set, the mouse deltas appear before the button byte.

The known identifying values for key/button events are (only the 0xfe bits are listed):

0x04 ‘Extend char’ (right-hand).
0x06 ‘Extend char’ (left-hand).
0x08 ‘Shift’ (right-hand).
0x0a ‘Shift’ (left-hand).
0x0c ‘CTRL’
0x0e ‘Break’ / ‘Reset’
0x10 4, on the numeric keypad.
0x12 8, on the numeric keypad.
0x14 5, on the numeric keypad.
0x16 9, on the numeric keypad.
0x18 6, on the numeric keypad.
0x1a 7, on the numeric keypad.
0x1c The comma key on the numeric keypad.
0x1e ‘Enter’, on the numeric keypad.
0x20 1, on the numeric keypad
0x22 The slash key on the numeric keypad.
0x24 2, on the numeric keypad.
0x26 The plus key on the numeric keypad.
0x28 3, on the numeric keypad.
0x2a The asterisk key on the numeric keypad.
0x2c 0, on the numeric keypad.
0x2e The minus key on the numeric keypad.
0x30 B
0x32 V
0x34 C
0x36 X
0x38 Z
0x3e ‘ESC’ / ‘DEL’
0x42 The second (counting from the left) of the four unmarked keys at the top right.
0x46 The third (counting from the left) of the four unmarked keys at the top right.
0x48 The period key on the numeric keypad.
0x4a The leftmost of the four unmarked keys at the top right.
0x4c The ‘Tab’ key on the numeric keypad.
0x4e The rightmost of the four unmarked keys at the top right.
0x50 H
0x52 G
0x54 F
0x56 D
0x58 S
0x5a A
0x5e ‘Caps’
0x60 U
0x62 Y
0x64 T
0x66 R
0x68 E
0x6a W
0x6c Q

0x6e ‘Tab’
0x70 The digit-7 / ampersand key.
0x72 The digit-6 / circumflex key.
0x74 The digit-5 / percent-sign key.
0x76 The digit-4 / dollar-sign key.
0x78 The digit-3 / hash-mark key.
0x7a The digit-2 / at-sign key.
0x7c The digit-1 / exclamation-point key.
0x7e The backquote / tilde key.
0x80 Left mouse button.
0x82 Right mouse button.
0x90 ‘Menu’
0x92 ‘F4’
0x94 ‘F3’
0x96 ‘F2’
0x98 ‘F1’
0x9c ‘Stop’
0x9e ‘Enter’ / ‘Print’
0xa0 ‘System’ / ‘User’
0xa2 ‘F5’
0xa4 ‘F6’
0xa6 ‘F7’
0xa8 ‘F8’
0xac ‘Clear line’
0xae ‘Clear display’
0xb0 The digit-8 / asterisk key.
0xb2 The digit-9 / left-parenthesis key.
0xb4 The digit-0 / right-parenthesis key.
0xb6 The minus-sign / underscore key.
0xb8 The equal-sign / plus-sign key.
0xba ‘Back space’
0xbc ‘Insert line’
0xbe ‘Delete line’
0xc0 I
0xc2 O
0xc4 P
0xc6 The left-bracket / left-brace key.
0xc8 The right-bracket / right-brace key.
0xca The backslash / vertical-bar key.
0xcc ‘Insert char’
0xce ‘Delete char’
0xd0 J
0xd2 K
0xd4 L
0xd6 The semicolon / colon key.
0xd8 The single-quote / double-quote key.
0xda ‘Return’
0xdc The arrow key pointing up and left.
0xde ‘Prev’

0xe0 M
0xe2 The period / less-than key.
0xe4 The comma / greater-than key.
0xe6 The slash / question-mark key.
0xea ‘Select’
0xee ‘Next’
0xf0 N
0xf2 The spacebar.
0xf8 The left-pointing arrow key.
0xfa The down-pointing arrow key.
0xfc The up-pointing arrow key.
0xfe The right-pointing arrow key.

FILES

`/dev/hil[1-7]` Individual HIL loop devices.

ERRORS

`[ENODEV]` No such HIL loop device.

`[ENXIO]` HIL loop is inoperative.

`[EBUSY]` Another HP-UX process has the device open, or another BSD process has the device open, and is using it in HP-UX mode.

`[EINVAL]` Invalid `ioctl`(2) specification.

BUGS

Requiring `HILIOCHPUX` to be able to read anything is gross. It is probably not worth fixing, though, because the whole hil/ite mechanism will probably be replaced with a more-or-less port-independent scheme in the foreseeable future.

NAME

hk — RK6-11/ RK06 and RK07 disk interface

SYNOPSIS

```
hk0 at uba? csr 0177440 vector rkintr
rk0 at hk0 drive 0
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **hk** driver is a typical block-device disk driver; block device I/O is described in physio(4).

The script MAKEDEV(8) should be used to create the special files; if a special file needs to be created by hand consult mknod(8).

DISK SUPPORT

Special file names begin with 'hk' and 'rhk' for the block and character files respectively. The second component of the name, a drive unit number in the range of zero to seven, is represented by a '?' in the disk layouts below. The last component is the file system partition which is designated by a letter from 'a' to 'h'. and corresponds to a minor device number set: zero to seven, eight to 15, 16 to 23 and so forth for drive zero, drive two and drive three respectively. The location and size (in sectors) of the partitions for the RK06 and RK07 drives are as follows:

RK07 partitions

disk	start	length	cyl	
	hk?a	0	15884	0-240
	hk?b	15906	10032	241-392
	hk?c	0	53790	0-814
	hk?d	25938	15884	393-633
	hk?f	41844	11792	634-814
	hk?g	25938	27786	393-813

RK06 partitions

disk	start	length	cyl	
	hk?a	0	15884	0-240
	hk?b	15906	11154	241-409
	hk?c	0	27126	0-410

On a dual RK-07 system partition hk?a is used for the root for one drive and partition hk?g for the /usr file system. If large jobs are to be run using hk?b on both drives as swap area provides a 10Mbyte paging area. Otherwise partition hk?c on the other drive is used as a single large file system.

FILES

/dev/hk[0-7][a-h]	block files
/dev/rhk[0-7][a-h]	raw files

DIAGNOSTICS

hk%d%c: hard error %sing fsbn %d[-%d] cs2=%b ds=%b er=%b. An unrecoverable error occurred during transfer of the specified filesystem block number(s), which are logical block numbers on the indicated partition. The contents of the cs2, ds and er registers are printed in octal and symbolically with bits decoded. The error was either unrecoverable, or a large number of retry attempts (including offset positioning and drive recalibration) could not recover the error.

rk%d: write locked. The write protect switch was set on the drive when a write was attempted. The write operation is not recoverable.

rk%d: not ready. The drive was spun down or off line when it was accessed. The i/o operation is not recoverable.

rk%d: not ready (came back!). The drive was not ready, but after printing the message about being not ready (which takes a fraction of a second) was ready. The operation is recovered if no further errors occur.

rk%d%c: soft ecc reading fsbn %d[-%d]. A recoverable ECC error occurred on the specified sector(s) in the specified disk partition. This happens normally a few times a week. If it happens more frequently than this the sectors where the errors are occurring should be checked to see if certain cylinders on the pack, spots on the carriage of the drive or heads are indicated.

hk%d: lost interrupt. A timer watching the controller detected no interrupt for an extended period while an operation was outstanding. This indicates a hardware or software failure. There is currently a hardware/software problem with spinning down drives while they are being accessed which causes this error to occur. The error causes a UNIBUS reset, and retry of the pending operations. If the controller continues to lose interrupts, this error will recur a few seconds later.

SEE ALSO

hp(4), uda(4), up(4), syslogd(8)

HISTORY

The **hk** driver appeared in 4.1BSD.

BUGS

The write(2) function scribbles on the tail of incomplete blocks.

DEC-standard error logging should be supported.

A program to analyze the logged error information (even in its present reduced form) is needed.

The partition tables for the file systems should be read off of each pack, as they are never quite what any single installation would prefer, and this would make packs more portable.

The RK07 g partition size in rk.c disagrees with that in /etc/disktab.

NAME

hme — Sun Microelectronics STP2002-STQ Ethernet interfaces device driver

SYNOPSIS

```
hme* at pci? dev ? function ?
hme* at sbus? slot ? offset ?
```

DESCRIPTION

The **hme** driver supports Sun Microelectronics STP2002-STQ Fast Ethernet interfaces.

HARDWARE

The **hme** driver supports the on-board Ethernet interfaces of many Sun UltraSPARC workstation and server models.

Cards supported by the **hme** driver include:

Sun PCI SunSwift Adapter ("SUNW,hme")
Sun SBus SunSwift Adapter ("hme" and "SUNW,hme")
Sun PCI Sun100BaseT Adapter 2.0 ("SUNW,hme")
Sun SBus Sun100BaseT 2.0 ("SUNW,hme")
Sun PCI Quad FastEthernet Controller ("SUNW,qfe")
Sun SBus Quad FastEthernet Controller ("SUNW,qfe")

The STP2002 family supports hardware checksumming to assist in computing IPv4 TCP/UDP checksums. The **hme** driver supports this feature of the chip. See `ifconfig`(8) for information on how to enable this feature.

SEE ALSO

`ifmedia`(4), `intro`(4), `mii`(4), `ukphy`(4), `ifconfig`(8)

Sun Microelectronics, *STP2002QFP Fast Ethernet, Parallel Port, SCSI (FEPS) User's Guide*, April 1996, http://mediacast.sun.com/users/Barton808/media/STP2002QFP-FEPs_UG.pdf.

HISTORY

The **hme** driver first appeared in NetBSD 1.5.

AUTHORS

The **hme** driver was written by Paul Kranenburg ⟨pk@NetBSD.org⟩.

NAME

hp — MASSBUS disk interface

SYNOPSIS

```
hp0 at mba0 drive 0
hp* at mba? drive ?
```

DESCRIPTION

The **hp** driver is a generic MASSBUS disk driver which handles the standard DEC controllers. It is typical of a block-device disk driver; block I/O is described in `physio`(4).

The script `MAKEDEV`(8) should be used to create the special files; if a special file needs to be created by hand consult `mknod`(8). It is recommended as a security precaution to not create special files for devices which may never be installed.

The first sector of each disk contains both a first-stage bootstrap program and a disk label containing geometry information and partition layouts (see `disklabel`(5). This sector is normally write-protected, and disk-to-disk copies should avoid copying this sector. The label may be updated with `disklabel`(8), which can also be used to write-enable and write-disable the sector. The next 15 sectors contain a second-stage bootstrap program.

DISK SUPPORT

During autoconfiguration or whenever a drive comes on line for the first time, or when a drive is opened after all partitions are closed, the first sector of the drive is examined for a disk label. If a label is found, the geometry of the drive and the partition tables are taken from it. If no label is found, a fake label is created by the driver, enough so that a real label can be written.

The hp?a partition is normally used for the root file system, the hp?b partition as a paging area, and the hp?c partition for pack-pack copying (it maps the entire disk). On disks larger than about 205 Megabytes, the hp?h partition is inserted prior to the hp?d or hp?g partition; the hp?g partition then maps the remainder of the pack.

FILES

`/dev/hp[0-7][a-h]` block files
`/dev/rhp[0-7][a-h]` raw files

DIAGNOSTICS

hp%d%c: hard error %sing fsbn %d [of %d-%d] (hp%d bn %d cn %d tn %d sn %d) mbsr=%b er1=%b er2=%b. An unrecoverable error occurred during transfer of the specified filesystem block number, which is a logical block number on the indicated partition. If the transfer involved multiple blocks, the block range is printed as well. The parenthesized fields list the actual disk sector number relative to the beginning of the drive, as well as the cylinder, track and sector number of the block. The MASSBUS status register is printed in hexadecimal and with the error bits decoded if any error bits other than MBEXC and DTABT are set. In any case the contents of the two error registers are also printed in octal and symbolically with bits decoded. (Note that er2 is what old RP06 manuals would call RPER3; the terminology is that of the RM disks). The error was either unrecoverable, or a large number of retry attempts (including offset positioning and drive recalibration) could not recover the error.

hp%d%c: soft ecc reading fsbn %d [of %d-%d] (hp%d bn %d cn %d tn %d sn %d). A recoverable ECC error occurred on the specified sector of the specified disk partition. If the transfer involved multiple blocks, the block range is printed as well. The parenthesized fields list the actual disk sector number relative to the beginning of the drive, as well as the cylinder, track and sector number of the block. This happens normally a few times a week. If it happens more frequently than this the sectors where the errors are occurring should be checked to see if certain cylinders on the pack, spots on the carriage of the drive or heads are indi-

cated.

SEE ALSO

physio(4), up(4), disklabel(5), MAKEDEV(8), disklabel(8), mknod(8)

HISTORY

The **hp** driver appeared in 4.0BSD.
A new **hp** driver showed up in NetBSD 1.2.

BUGS

DEC-standard bad144(8) bad-block handling should be used.

DEC-standard error logging should be supported.

A program to analyze the logged error information (even in its present reduced form) is needed.

NAME

hpc — SGI High performance Peripheral Controller

SYNOPSIS

```
hpc0 at gio0 addr 0x1fb80000
hpc1 at gio0 addr 0x1fb00000
hpc2 at gio0 addr 0x1fb98000
hpc3 at gio0 addr 0x1fb90000
```

DESCRIPTION

hpc interfaces the peripherals connected to it to the gio(4) bus. **hpc** is found on the Personal Iris 4D/3x, Indigo, Indy, Challenge S, Challenge M, and Indigo2 machines.

There are three different numerical revisions of the **hpc** controller. Revisions 1 and 1.5 exist on Personal Iris 4D/3x and Indigo machines, as well as GIO32bis expansion cards such as the E++ SEEQ-based Ethernet adapter. Revision 1.5 supports bi-endian operation. Revision 3 exists on Indy, Challenge S, Indigo2, and Challenge M systems. It is possible to have an on-board HPC3 as well as HPC1.5-based GIO32bis adapters in the Indy and Challenge S systems. Additionally, the Challenge S may have a secondary HPC3 if the IOPLUS (a.k.a. ''mezzanine'') board is installed.

HARDWARE

dsclock	DS1286-based RTC
dpclock	DP8573A-based RTC
haltwo	HAL2 audio controller
sq	Seeq 8003 and 80C03 Ethernet controllers
wdsc	WD33c93 SCSI controller
zsc	Zilog Z8530 UART

SEE ALSO

gio(4), imc(4), pic(4)

HISTORY

The **hpc** driver first appeared in NetBSD 1.6 with support for Revision 3. Revision 1 and 1.5 support was added in NetBSD 2.0.

BUGS

hpc Revisions 1 and 1.5 support DMA buffer pointers of only 28 bits and may therefore only address 256 megabytes of memory. The R4k Indigo and Indy are the only systems that support sufficient memory to illustrate this drawback. A software workaround is not currently implemented. Revision 3, with 32 bit pointers, does not have this limitation.

NAME

hpib — Built-in and 98625 HP-IB interface

SYNOPSIS

```
nhpib* at dio? scode ?
hpibbus*    at nhpib?
fhpib* at dio? scode ?
hpibbus*    at fhpib?
```

DESCRIPTION

The **hpib** driver supports the built-in and 98625 HP-IB interface.

SEE ALSO

ct(4), ppi(4), rd(4)

NAME

hpqlb — HP Quick Launch Buttons

SYNOPSIS

hpqlb* **at acpi?**

DESCRIPTION

Many HP notebook computers come with hotkeys. The **hpqlb** driver provides support for all hotkeys generating keycodes.

The following hotkeys which do not generate keycodes are:

- Display Switch
- Brightness Up
- Brightness Down

SEE ALSO

acpi(4)

HISTORY

The **hpqlb** driver appeared in NetBSD 5.0.

AUTHORS

This driver was written for NetBSD by Christoph Egger.

NAME

hptide — Triones/Highpoint IDE disk controllers driver

SYNOPSIS

```
hptide* at pci? dev ? function ? flags 0x0000
```

DESCRIPTION

The **hptide** driver supports the Triones/Highpoint HPT366, HPT370, HPT370A, HPT372 and HPT374 IDE controllers, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **hptide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

The timings used for the PIO and DMA modes for controllers listed above are for a PCI bus running at 30 or 33 MHz. This driver may not work properly on overclocked systems.

NAME

ht — TM-03/ TE-16, TU-45, TU-77 MASSBUS mag tape device interface:

SYNOPSIS

```
ht0 at mba? drive ?
tu0 at ht0 slave 0
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The TM-03 transport combination provides a standard tape drive interface as described in mtio(4). All drives provide both 800 and 1600 BPI; the TE-16 runs at 45 IPS, the TU-45 at 75 IPS, while the TU-77 runs at 125 IPS and autoloads tapes.

DIAGNOSTICS

tu%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

tu%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

tu%d: can't change density in mid-tape. An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.

tu%d: hard error bn%d mbsr=%b er=%b ds=%b. A tape error occurred at block *bn*; the ht error register and drive status register are printed in octal with the bits symbolically decoded. Any error is fatal on non-raw tape; when possible the driver will have retried the operation which failed several times before reporting the error.

SEE ALSO

mt(1), tar(1), mt(4), mtio(4), physio(4), tm(4), ts(4), ut(4)

HISTORY

An **ht** driver appeared in Version 6 AT&T UNIX.

BUGS

May hang if physical (non-data) errors occur.

NAME

hy — Network Systems Hyperchannel interface

SYNOPSIS

```
hy0 at uba0 csr 0172410 vector hyint
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **hy** interface provides access to a Network Systems Corporation Hyperchannel Adapter.

The network to which the interface is attached is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The host's address is discovered by reading the adapter status register. The interface will not transmit or receive packets until the network number is known.

DIAGNOSTICS

hy%d: unit number 0x%x port %d type %x microcode level 0x%x. Identifies the device during autoconfiguration.

hy%d: can't handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

hy%d: can't initialize. The interface was unable to allocate UNIBUS resources. This is usually due to having too many network devices on an 11/750 where there are only 3 buffered data paths.

hy%d: NEX - Non Existent Memory. Non existent memory error returned from hardware.

hy%d: BAR overflow. Bus address register overflow error returned from hardware.

hy%d: Power Off bit set, trying to reset. Adapter has lost power, driver will reset the bit and see if power is still out in the adapter.

hy%d: Power Off Error, network shutdown. Power was really off in the adapter, network connections are dropped. Software does not shut down the network unless power has been off for a while.

hy%d: RECVD MP > MPSIZE (%d). A message proper was received that is too big. Probable a driver bug. Shouldn't happen.

hy%d: xmit error − len > hy_olen [%d > %d]. Probable driver error. Shouldn't happen.

hy%d: DRIVER BUG − INVALID STATE %d. The driver state machine reached a non-existent state. Definite driver bug.

hy%d: watchdog timer expired. A command in the adapter has taken too long to complete. Driver will abort and retry the command.

hy%d: adapter power restored. Software was able to reset the power off bit, indicating that the power has been restored.

SEE ALSO

`inet`(4), `netintro`(4)

HISTORY

The **hy** interface appeared in 4.2BSD.

BUGS

If the adapter does not respond to the status command issued during autoconfigure, the adapter is assumed down. A reboot is required to recognize it.

The adapter power fail interrupt seems to occur sporadically when power has, in fact, not failed. The driver will believe that power has failed only if it can not reset the power fail latch after a “reasonable” time interval. These seem to appear about 2-4 times a day on some machines. There seems to be no correlation with adapter rev level, number of ports used etc. and whether a machine will get these “bogus powerfails”. They don’t seem to cause any real problems so they have been ignored.

NAME

hyper — A1096A ‘‘Hyperion’’ graphics device interface

SYNOPSIS

hyper* at dio? scode ?

DESCRIPTION

This driver is for the Hyperion monochrome framebuffer.

SEE ALSO

grf(4), ite(4)

NAME

iavc — isdn4bsd AVM B1 driver

SYNOPSIS

iavc* **at pci?**

DESCRIPTION

The **iavc** driver is used to glue the AVM family of active cards to the isdncapi(4) driver and the *isdn4bsd* package. Currently only the AVM B1 PCI is supported. Support for the AVM B1 ISA and the AVM T1 PCI cards should be quite easy to add, since support already exists in the FreeBSD version of the driver.

To use this driver, you must first fetch the firmware file **b1.t4** from ftp://ftp.avm.de/ and load it to the card using isdnd(8).

SEE ALSO

isdncapi(4), isdnd(8)

STANDARDS

CAPI 2.0 (http://www.capi.org/)

AUTHORS

The **iavc** device driver was written by Juha-Matti Liukkonen ⟨jml@cubical.fi⟩ (Cubical Solutions Ltd, Finland) for FreeBSD and ported to NetBSD by Antti Kantee ⟨pooka@cubical.fi⟩. This manpage was written by Hellmuth Michaelis ⟨hm@FreeBSD.org⟩.

NAME

ibus — pmax internal I/O space driver

SYNOPSIS

```
ibus0 at mainbus0
ibus0 at tc? slot ? offset ?
```

DESCRIPTION

ibus is a virtual device corresponding to the pmax internal I/O space found on DEC 2100, 3100, 5100 and 5000/200 systems.

Internal I/O space spans the pmax physical address space, and is mapped permanently in the kernel virtual space at the very early time of the kernel startup procedure.

ibus driver manages the internal I/O space of pmax.

- Address range management to avoid confliction of address space of which devices probe by touching hardware port is difficult.
- Interrupt vector management.
- bus_space(9) and bus_dma(9) implementation.
- Other utility functions.

ibus is always required to run the NetBSD kernel.

SEE ALSO

intro(4), bus_dma(9), bus_space(9)

NAME

ichlpcib — Intel ICH LPC Interface Bridge

SYNOPSIS

```
ichlpcib* at pci? dev ? function ?
hpet0     at ichlpcib?
isa0      at ichlpcib?
```

DESCRIPTION

The **ichlpcib** driver supports the Intel ICH LPC Interface Bridges on compatible chipsets. Supported functions include:

- Watchdog timer. The watchdog timer may be configured for a 2 seconds (on ICH6 or newer) and 4 seconds (on ICH5 or older) min period and for a 39 seconds (on ICH5 or older) or 613 seconds max period (on ICH6 or newer).
- Power Management timer. A 24-bit timer available to be used by the timecounters framework.
- SpeedStep. In some systems the SpeedStep function is also available, and can be used to switch between high and low frequency (to reduce power consumption) via the *machdep.speedstep_state* `sysctl`(8) node. A value of 0 will use the low frequency (low power) and a 1 will enable the high frequency (full power).

SEE ALSO

`sysctl`(8), `wdogctl`(8)

HISTORY

The **ichlpcib** driver first appeared in NetBSD 3.0.

AUTHORS

The **ichlpcib** driver was written by Minoura Makoto and Matthew R. Green.

NAME

ichsmb — Intel ICH SMBus controller

SYNOPSIS

```
ichsmb* at pci?
iic* at ichsmb?
```

DESCRIPTION

The **ichsmb** driver provides support for the Intel ICH SMBus host interface to be used with the iic(4) framework.

Supported chipsets:

- Intel ICH, ICH2, ICH3, ICH4, ICH4-M, ICH5, ICH5R, ICH6, ICH6-M, ICH6R, ICH7, ICH8, C-ICH, 6300ESB and 6321ESB.

SEE ALSO

iic(4), intro(4), pci(4)

HISTORY

The **ichsmb** driver first appeared in OpenBSD 4.0. This driver imports from OpenBSD 3.9.

AUTHORS

The **ichsmb** driver was written by Alexander Yurchenko ⟨grange@openbsd.org⟩.

BUGS

The driver doesn't support I2C commands with a data buffer size of more than 2 bytes.

NAME

icmp — Internet Control Message Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

int
socket(AF_INET, SOCK_RAW, proto);
```

DESCRIPTION

ICMP is the error and control message protocol used by IP and the Internet protocol family. It may be accessed through a "raw socket" for network monitoring and diagnostic functions. The *proto* parameter to the socket call to create an ICMP socket is obtained from getprotobyname(3). ICMP sockets are connectionless, and are normally used with the sendto(2) and recvfrom(2) calls, though the connect(2) call may also be used to fix the destination for future packets (in which case the read(2) or recv(2) and write(2) or send(2) system calls may be used).

Outgoing packets automatically have an IP header prepended to them (based on the destination address). Incoming packets are received with the IP header and options intact.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
[ENOTCONN]	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
[ENOBUFS]	when the system runs out of memory for an internal data structure;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

recv(2), send(2), inet(4), intro(4), ip(4)

Internet Control Message Protocol, RFC, 792, September 1981.

Requirements for Internet Hosts -- Communication Layers, RFC, 1122, October 1989.

HISTORY

The **icmp** protocol appeared in 4.3BSD.

NAME

icmp6 — Internet Control Message Protocol for IPv6

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/icmp6.h>

int
socket(AF_INET6, SOCK_RAW, IPPROTO_ICMPV6);
```

DESCRIPTION

ICMPv6 is the error and control message protocol used by IPv6 and the IPv6 protocol family (see ip6(4) and inet6(4)). It may be accessed through a "raw socket" for network monitoring and diagnostic functions.

The *proto* parameter to the socket(2) call to create an ICMPv6 socket may be obtained from getprotobyname(3). ICMPv6 sockets are connectionless, and are normally used with the sendto(2) and recvfrom(2) calls, though the connect(2) call may also be used to fix the destination for future packets (in which case read(2) or recv(2) and write(2) or send(2) system calls may be used).

Outgoing packets automatically have an IPv6 header prepended to them (based on the destination address). Incoming packets on the socket are received with the IPv6 header and any extension headers removed.

Types

ICMPv6 messages are classified according to the type and code fields present in the ICMPv6 header. The abbreviations for the types and codes may be used in rules in pf.conf(5). The following types are defined:

Num	Abbrev.	Description
1	unreach	Destination unreachable
2	toobig	Packet too big
3	timex	Time exceeded
4	paramprob	Invalid IPv6 header
128	echoreq	Echo service request
129	echorep	Echo service reply
130	groupqry	Group membership query
130	listqry	Multicast listener query
131	grouprep	Group membership report
131	listenrep	Multicast listener report
132	groupterm	Group membership termination
132	listendone	Multicast listerner done
133	routersol	Router solicitation
134	routeradv	Router advertisement
135	neighbrsol	Neighbor solicitation
136	neighbradv	Neighbor advertisement
137	redir	Shorter route exists
138	routrrenum	Route renumbering
139	fqdnreq	FQDN query
139	niqry	Node information query
139	wrureq	Who-are-you request
140	fqdnrep	FQDN reply

140	nirepNode information reply
140	wrurepWho-are-you reply
200	mtracerespmtrace response
201	mtracemtrace messages

The following codes are defined:

Num		Abbrev.	TypeDescription
0	noroute-unr	unreach	No route to destination
1	admin-unr	unreach	Administratively prohibited
2	beyond-unr	unreach	Beyond scope of source address
2	notnbr-unr	unreach	Not a neighbor (obselete)
3	addr-unr	unreach	Address unreachable
4	port-unr	unreach	Port unreachable
0	transit	timex	Time exceeded in transit
1	reassemb	timex	Time exceeded in reassembly
0	badhead	paramprob	Erroneous header field
1	nxthdr	paramprob	Unrecognized next header
2		redir	Unrecognized option
0	redironlink	redir	Redirection to on-link node
1	redirrouter	redir	Redirection to better router

Headers

All ICMPv6 messages are prefixed with an ICMPv6 header. This header corresponds to the *icmp6_hdr* structure and has the following definition:

```
struct icmp6_hdr {
        uint8_t icmp6_type;    /* type field */
        uint8_t icmp6_code;    /* code field */
        uint16_t       icmp6_cksum;   /* checksum field */
        union {
                uint32_t icmp6_un_data32[1]; /* type-specific */
                uint16_t icmp6_un_data16[2]; /* type-specific */
                uint8_t  icmp6_un_data8[4];  /* type-specific */
        } icmp6_dataun;
} __packed;

#define icmp6_data32   icmp6_dataun.icmp6_un_data32
#define icmp6_data16   icmp6_dataun.icmp6_un_data16
#define icmp6_data8    icmp6_dataun.icmp6_un_data8
#define icmp6_pptr     icmp6_data32[0]        /* parameter prob */
#define icmp6_mtu      icmp6_data32[0]        /* packet too big */
#define icmp6_id       icmp6_data16[0]        /* echo request/reply */
#define icmp6_seq      icmp6_data16[1]        /* echo request/reply */
#define icmp6_maxdelay icmp6_data16[0]        /* mcast group membership*/
```

icmp6_type describes the type of the message. Suitable values are defined in ⟨netinet/icmp6.h⟩. *icmp6_code* describes the sub-type of the message and depends on *icmp6_type*. *icmp6_cksum* contains the checksum for the message and is filled in by the kernel on outgoing messages. The other fields are used for type-specific purposes.

Filters

Because of the extra functionality of ICMPv6 in comparison to ICMPv4, a larger number of messages may be potentially received on an ICMPv6 socket. Input filters may therefore be used to restrict input to a subset of the incoming ICMPv6 messages so only interesting messages are returned by the `recv`(2) family of calls to an application.

The *`icmp6_filter`* structure may be used to refine the input message set according to the ICMPv6 type. By default, all messages types are allowed on newly created raw ICMPv6 sockets. The following macros may be used to refine the input set:

`void ICMP6_FILTER_SETPASSALL`(*`struct icmp6_filter *filterp`*)
Allow all incoming messages. *filterp* is modified to allow all message types.

`void ICMP6_FILTER_SETBLOCKALL`(*`struct icmp6_filter *filterp`*)
Ignore all incoming messages. *filterp* is modified to ignore all message types.

`void ICMP6_FILTER_SETPASS`(*`int type`*, *`struct icmp6_filter *filterp`*)
Allow ICMPv6 messages with the given *`type`*. *filterp* is modified to allow such messages.

`void ICMP6_FILTER_SETBLOCK`(*`int type`*, *`struct icmp6_filter *filterp`*)
Ignore ICMPv6 messages with the given *`type`*. *filterp* is modified to ignore such messages.

`int ICMP6_FILTER_WILLPASS`(*`int type`*, *`const struct icmp6_filter *filterp`*)
Determine if the given filter will allow an ICMPv6 message of the given type.

`int ICMP6_FILTER_WILLBLOCK`(*`int type`*, *`const struct icmp6_filter *filterp`*)
Determine if the given filter will ignore an ICMPv6 message of the given type.

The `getsockopt`(2) and `setsockopt`(2) calls may be used to obtain and install the filter on ICMPv6 sockets at option level `IPPROTO_ICMPV6` and name `ICMPV6_FILTER` with a pointer to the *`icmp6_filter`* structure as the option value.

SEE ALSO

`getsockopt`(2), `recv`(2), `send`(2), `setsockopt`(2), `socket`(2), `getprotobyname`(3), `inet6`(4), `ip6`(4), `netintro`(4)

W. Stevens and M. Thomas, *Advanced Sockets API for IPv6*, RFC 2292, February 1998.

A. Conta and S. Deering, *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, RFC 2463, December 1998.

NAME

icp — ICP-Vortex and Intel Storage RAID driver

SYNOPSIS

```
icp* at pci? dev ? function ?
ld* at icp? unit ?
icpsp* at icp? unit ?
scsibus* at icpsp?
```

DESCRIPTION

The **icp** driver provides support for the newer (post 1997) ICP-Vortex GDT and Intel Storage RAID controllers.

The **ld** driver provides access to logical devices (disk arrays) presented by the controller. The **icpsp** driver provides direct access to other SCSI devices attached to the controller, such as tape or CD-ROM drives.

DIAGNOSTICS

pci_mem_find: expected mem type 00000000, found 00000002

This message may be displayed during autoconfiguration if the controller's memory is mapped below 1MB in physical address space. This can be disabled either through changing a setting in the controller's BIOS utility, or changing the position of a jumper on the board. See your controller's documentation for more information.

SEE ALSO

intro(4), ld(4), scsi(4)

HISTORY

The **icp** driver first appeared in NetBSD 1.6, and was based on the FreeBSD and OpenBSD drivers of the same name.

BUGS

ISA & EISA front-ends and a management interface are needed.

Older PCI boards are not yet supported.

NAME

icsphy — Driver for Integrated Circuit Systems ICS1890/1893 10/100 Ethernet PHYs

SYNOPSIS

icsphy* **at mii? phy ?**

DESCRIPTION

The **icsphy** driver supports the Integrated Circuit Systems ICS1890 and ICS1893 10/100 Ethernet PHYs. These PHYs are found on a variety of high-performance Ethernet interfaces.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

ie — Ethernet driver for Ether1 podule Ethernet interfaces

SYNOPSIS

```
ie0 at podulebus?
```

DESCRIPTION

The **ie** interface provides access to a 10 Mb/s Ethernet network.

HARDWARE

The Ethernet cards supported by the **ie** interface are:

Acorn Ether1 podule

SEE ALSO

ea(4), eb(4), eh(4), em(4), es(4), intro(4), ifconfig(8)

NAME

ie — Ethernet driver for Intel 82596 Ethernet chip

SYNOPSIS

```
ie0 at pcctwo? ipl 3
```

DESCRIPTION

The **ie** interface provides access to a 10 Mb/s Ethernet network via the Intel 82596 chip fitted to Motorola MVME167 and MVME177 single board computers.

SEE ALSO

intro(4), le(4), pcctwo(4), ifconfig(8)

NAME

ie — SPARC Intel 82586 ethernet interface

SYNOPSIS

```
ie0 at obio0 addr 0xf6000000 level 6            (sun4/200 on-board)
ie0 at obio0 addr 0x06000000 level 6            (sun4/100 on-board)
ie1 at vmes0 addr 0xffe88000 level 5 vect 0x75  (VME)
ie2 at vmes0 addr 0xff31ff02 level 5 vect 0x76  (VME)
ie3 at vmes0 addr 0xff35ff02 level 5 vect 0x77  (VME)
ie4 at vmes0 addr 0xff2dff02 level 5 vect 0x7c  (VME)
```

DESCRIPTION

The **ie** interface provides access to the 10 Mb/s Ethernet network via the Intel 82586 Ethernet chip set. The **ie** is found as an on-board interface on Sun 4/100 and 4/200 workstations. The **ie** also exists as a VME card.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **ie** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

SEE ALSO

`arp`(4), `ifmedia`(4), `inet`(4), `intro`(4), `ifconfig`(8)

NAME

ie — Intel 82586 Ethernet interface driver

SYNOPSIS

```
ie0 at obio0 addr 0x7f0800 ipl 3
ie0 at mbmem0 addr 0x88000 ipl 3
ie1 at mbmem0 addr 0x8c000 ipl 3
ie1 at vme0 addr 0xe88000,0xe00000 len -1,0x40000 irq 3 vect 0x75
```

DESCRIPTION

The **ie** interface provides access to the 10 Mb/s Ethernet network via the Intel 82586 Ethernet chip set.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **ie** interface employs the Address Resolution Protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

SEE ALSO

`arp`(4), `ec`(4), `inet`(4), `intro`(4)

NAME

ie — Intel 82586 Ethernet interface driver

SYNOPSIS

```
ie0 at obio0 addr 0x0C0000 ipl 3
ie1 at vme2  addr 0xe88000 ipl 3 vect 0x75
ie* at sebuf?
```

DESCRIPTION

The **ie** interface provides access to the 10 Mb/s Ethernet network via the Intel 82586 Ethernet chip set.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **ie** interface employs the Address Resolution Protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

SEE ALSO

`arp`(4), `inet`(4), `intro`(4), `le`(4)

NAME

iee — Intel i82596 10MBit/s Ethernet interface

SYNOPSIS

hp700

```
iee* at gsc?
```

DESCRIPTION

The **iee** device provides access to the Intel i82596 10MBit/s Ethernet interface. **iee** supports IEEE 802.1Q Virtual LANs. **iee** operates the i82596 in 32-Bit Linear Mode, opposed to ie(4) that drives the i82596 only in i82586 compatibility mode.

HARDWARE

There is currently only an MD frontend for the hp700 GSC bus.

hp700

The **iee** interface suports Intel i82596DX on ASP/ASP2 based machines and the i82596CA macrocell in LASI based machines. GSC expansion cards may work, but are not tested. Media selection on hp700 is done either manually via hardware jumper or automatically depending on machine model.

DIAGNOSTICS

iee%d: iee_intr: receive error %d, rfd_status=0x%.4x, rfd_count=0x%.4x. An error during frame reception occured. The frame was dropped.

iee%d: iee_intr: can't allocate mbuf.
iee%d: iee_intr: can't alloc mbuf cluster. A frame was received, but dropped due to lack of memory.

iee%d: iee_intr: receive ring buffer overrun Many frames where received under high load and the receive ring buffer was to small to store them all. Frame reception was restarted from scratch. Most likely there where frames lost.

iee%d: iee_intr: scb_status=0x%x scb_cmd=0x%x faild command %d: cb_status[%d]=0x%.4x cb_cmd[%d]=0x%.4x A transmit or setup command was not executed successfully.

iee%d: iee_intr: crc_err=%d Number of frames with a CRC error received.

iee%d: iee_intr: align_err=%d Number of unaligned frames received.

iee%d: iee_intr: resource_err=%d Number of good frames dropped because the receive ring buffer was full.

iee%d: iee_intr: overrun_err=%d Number of frames lost because the system bus was not available for DMA.

iee%d: iee_intr: rcvcdt_err=%d Number of collisions detected during frame reception.

iee%d: iee_intr: short_fr_err=%d Number of frames received that where shorter then the minimum frame length.

iee%d: iee_start: failed to load DMA map A mbuf(9) chain with too many elements could not be setup for transmission. The mbuf(9) chain will be merged into a single mbuf(9) cluster and retransmitted.

iee%d: iee_start: can't allocate mbuf.
iee%d: iee_start: can't load TX DMA map. Said error occurred during merging the mbuf(9) chain into a mbuf(9) cluster. The frame was dropped.

iee%d: iee_init: can't create TX DMA map
iee%d: iee_init: can't allocate mbuf
iee%d: iee_init: can't allocate mbuf cluster
iee%d: iee_init: can't create RX DMA map
iee%d: iee_init: can't load RX DMA map There was no memory free to allocate resources when the operator tried to bring the interface up. The interface will not come up. Try again later.

iee%d: iee_watchdog: transmit timeout %d
iee%d: iee_watchdog: setup timeout %d The hardware didn't respond to a transmit or setup command within five seconds. The interface will be reset and restarted.

iee%d: iee_gsc_cmd: timeout n=%d Timeout at sending a channel attention command to the chip.

iee%d: iee_gsc_reset timeout bussy=0x%x Timeout at resetting the chip. Possible errors during `autoconf`(4).

iee%d: iee_gsc_attach: can't map I/O space The driver failed to map the I/O ports of the chip. The device will not be attached.

iee%d: iee_gsc_attach: can't allocate %d bytes of DMA memory
iee%d: iee_gsc_attach: can't map DMA memory
iee%d: iee_gsc_attach: can't create DMA map
iee%d: iee_gsc_attach: can't load DMA map The driver failed to get the shared DMA memory for the chip. The device will not be attached.

SEE ALSO

`arp`(4), `ifmedia`(4), `inet`(4), `intro`(4), `vlan`(4), `ifconfig`(8)

HISTORY

The **`iee`** driver appeared in NetBSD 2.0.

AUTHORS

Jochen Kunz

BUGS

None. ;-)

NAME

ieee1394if — IEEE1394 High-performance Serial Bus

SYNOPSIS

ieee1394if* **at fwohci?**

DESCRIPTION

NetBSD provides machine-independent bus support and raw drivers for IEEE1394 interfaces.

The **ieee1394if** driver consists of two layers: the controller and the bus layer. The controller attaches to a physical bus (like pci(4)). The **ieee1394if** bus attaches to the controller. Additional drivers can be attached to the bus.

Up to 63 devices, including the host itself, can be attached to a IEEE1394 bus. The root node is dynamically assigned with a PHY device function. Also, the other IEEE1394 bus specific parameters, e.g., node ID, cycle master, isochronous resource manager and bus manager, are dynamically assigned, after bus reset is initiated. On the **ieee1394if** bus, every device is identified by an EUI 64 address.

FILES

/dev/fw0.0
/dev/fwmem0.0

SEE ALSO

fwip(4), fwohci(4), pci(4), sbp(4), fwctl(8), sysctl(8)

HISTORY

The **ieee1394if** driver first appeared in FreeBSD 5.0, as **firewire**. It was added to NetBSD 4.0 under its present name.

AUTHORS

The **ieee1394if** driver was written by Katsushi Kobayashi and Hidetoshi Shimokawa for the FreeBSD project. It was added to NetBSD 4.0 by KIYOHARA Takashi.

BUGS

See fwohci(4) for security notes.

NAME

ieee80211 — standard interface to IEEE 802.11 devices

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/ethernet.h>
#include <net/if_ieee80211.h>
```

DESCRIPTION

This section describes the standard interface to configuration and status information on IEEE 802.11 devices. Most devices support options not configurable by this interface. They must be set by their respective, specific control program. The interface is via one of the following `ioctl`(2) calls on a socket:

`SIOCG80211` Get configuration or status information.

`SIOCS80211` Set configuration information.

These requests are made via a modified *`ifreq`* structure. This structure is defined as follows:

```
struct ieee80211req {
        char            i_name[IFNAMSIZ];      /* if_name, e.g. "wi0" */
        uint16_t        i_type;                /* req type */
        int16_t         i_val;                 /* Index or simple value */
        int16_t         i_len;                 /* Index or simple value */
        void            *i_data;               /* Extra data */
};
```

For `SIOCG80211` the following values of *i_type* are valid:

`IEEE80211_IOC_SSID`
: Returns the requested SSID by copying it into the buffer pointed to by *i_data* and setting *i_len* to the length. If *i_val* is ≥ 0 then the request refers to the configured value for that slot. Generally, 0 is the only valid value, but some interfaces support more SSIDs. If *i_val* is −1 then the request refers to the currently active value.

`IEEE80211_IOC_NUMSSIDS`
: Returns the number of SSIDs this card supports. In most cases, this is 1, but some devices such as `an`(4) support more.

`IEEE80211_IOC_WEP`
: Returns the current WEP status in *i_val*. Valid values are `IEEE80211_WEP_NOSUP`, `IEEE80211_WEP_ON`, `IEEE80211_WEP_OFF`, and `IEEE80211_WEP_MIXED`. Respectively, these values mean unsupported, mandatory for all devices, off, and on, but not required for all devices.

`IEEE80211_IOC_WEPKEY`
: Returns the requested WEP key via *i_data* and its length via *i_len*. If the device does not support returning the WEP key or the user is not root then the key may be returned as all zeros. Technically this is a valid key, but it is the kind of key an idiot would put on his luggage so we use it as a special value. Generally, only four WEP keys are allowed, but some devices support more. If so, the first four (0-3) are the standard keys stored in volatile storage and the others are device specific.

`IEEE80211_IOC_NUMWEPKEYS`
: Returns the number of WEP keys supported by this device, generally 4. A device that does not support WEP may either report 0 or simply return `EINVAL`.

`IEEE80211_IOC_WEPTXKEY`
Returns the WEP key used for transmission.

`IEEE80211_IOC_AUTHMODE`
Returns the current authentication mode in *i_val*. Valid values are `IEEE80211_AUTH_NONE`, `IEEE80211_AUTH_OPEN`, and `IEEE80211_AUTH_SHARED`.

`IEEE80211_IOC_STATIONNAME`
Returns the station name via *i_data* and its length via *i_len*. While all known devices seem to support this in some way or another, they all do it differently and it appears to not have anything to do with the actual IEEE 802.11 standard so making up an answer may be necessary for future devices.

`IEEE80211_IOC_CHANNEL`
Returns the current direct sequence spread spectrum channel in use.

`IEEE80211_IOC_POWERSAVE`
Returns the current powersaving mode. Valid values are `IEEE80211_POWERSAVE_NOSUP`, `IEEE80211_POWERSAVE_OFF`, `IEEE80211_POWERSAVE_ON`, `IEEE80211_POWERSAVE_CAM`, `IEEE80211_POWERSAVE_PSP`, and `IEEE80211_POWERSAVE_PSP_CAM`. Currently, `IEEE80211_POWERSAVE_ON` is defined to be equal to `IEEE80211_POWERSAVE_CAM`, but this may be incorrect.

`IEEE80211_IOC_POWERSAVESLEEP`
Returns the powersave sleep time in msec in *i_val*.

For `SIOCS80211` the following values of *i_type* are valid:

`IEEE80211_IOC_SSID`
Set the desired SSID for infrastructure and ad-hoc modes to value given by *i_data* and *i_len*. The length should be no longer than 32 characters.

`IEEE80211_IOC_WEP`
Set the current WEP mode to the value given in *i_val*. Valid values are the same as those for this value above. Devices which do not support all modes may choose to either return `EINVAL` or choose a reasonable alternate (supported) setting.

`IEEE80211_IOC_WEPKEY`
Set the WEP key indicated by *i_val* to the value given by *i_data* and *i_len*. Generally, valid values of *i_len* are 0, 5, and 13 though not all devices with WEP support have support for 13-byte keys.

`IEEE80211_IOC_WEPTXKEY`
Set the WEP key used for transmission to the value in *i_val*. Not all values which are valid for setting keys may be valid for setting transmit keys due to strange device interfaces.

`IEEE80211_IOC_AUTHMODE`
Set the current authorization mode to the value given in *i_val*. Valid values are given above. Not all devices support this.

`IEEE80211_IOC_STATIONNAME`
Set the station name to the value given by *i_data* and *i_len*. The standard does not appear to deal with this feature so the range of valid values may vary from device to device.

`IEEE80211_IOC_CHANNEL`
Set the desired ad-hoc channel to the value given by *i_val*. On some devices this has an impact on infrastructure mode as well. Valid values are 1-14, but 0 should be allowed and should return the device to the default value. May devices support this directly by converting any invalid value to the default value.

`IEEE80211_IOC_POWERSAVE`
: Set the current powersaving mode to the value given in *i_val*. Valid values are the same as those for this value above. Devices which do not support all modes may choose to either return `EINVAL` or choose a reasonable alternate (supported) setting. Most devices only support CAM mode.

`IEEE80211_IOC_POWERSAVESLEEP`
: Set the powersave sleep time in msec to the value in *i_val*.

SEE ALSO

`ioctl`(2), `an`(4), `ray`(4), `wi`(4), `ifconfig`(8)

HISTORY

The **`ieee80211`** manual appeared in FreeBSD 4.3.

NAME

ifmedia — network interface media settings

SYNOPSIS

```
#include <sys/socket.h>
#include <net/if.h>
#include <net/if_media.h>
```

DESCRIPTION

The **ifmedia** interface provides a consistent method for querying and setting network interface media and media options. The media is typically set using the ifconfig(8) command.

There are currently four link types supported by **ifmedia**:

IFM_ETHER	Ethernet
IFM_TOKEN	Token Ring
IFM_FDDI	FDDI
IFM_IEEE80211	IEEE802.11 Wireless LAN

The following sections describe the possible media settings for each link type. Not all of these are supported by every device; refer to your device's manual page for more information.

The lists below provide the possible names of each media type or option. The first name in the list is the canonical name of the media type or option. Additional names are acceptable aliases for the media type or option.

COMMON MEDIA TYPES AND OPTIONS

The following media types are shared by all link types:

IFM_AUTO	Autoselect the best media. [autoselect, auto]
IFM_MANUAL	Jumper or switch on device selects media. [manual]
IFM_NONE	Deselect all media. [none]

The following media options are shared by all link types:

IFM_FDX	Place the device into full-duplex mode. This option only has meaning if the device is normally not full-duplex. [full-duplex, fdx]
IFM_HDX	Place the device into half-duplex mode. This option only has meaning if the device is normally not half-duplex. [half-duplex, hdx]
IFM_FLOW	Hardware flow control support. [flowcontrol, flow]
IFM_FLAG0	Driver-defined flag. [flag0]
IFM_FLAG1	Driver-defined flag. [flag1]
IFM_FLAG2	Driver-defined flag. [flag2]
IFM_LOOP	Place the device into hardware loopback mode. [loopback, hw-loopback, loop]

MEDIA TYPES AND OPTIONS FOR ETHERNET

The following media types are defined for Ethernet:

IFM_HPNA_1	HomePNA 1.0, 1Mb/s. [HomePNA1, HPNA1]
IFM_10_T	10BASE-T, 10Mb/s over unshielded twisted pair, RJ45 connector. [10baseT, UTP, 10UTP]

`IFM_10_2` 10BASE2, 10Mb/s over coaxial cable, BNC connector, also called Thinnet. [10base2, BNC, 10BNC]

`IFM_10_5` 10BASE5, 10Mb/s over 15-wire cables, DB15 connector, also called AUI. [10base5, AUI, 10AUI]

`IFM_10_STP` 10BASE-STP, 10Mb/s over shielded twisted pair, DB9 connector. [10baseSTP, STP, 10STP]

`IFM_10_FL` 10BASE-FL, 10Mb/s over fiber optic cables. [10baseFL, FL, 10FL]

`IFM_100_TX` 100BASE-TX, 100Mb/s over unshielded twisted pair, RJ45 connector. [100baseTX, 100TX]

`IFM_100_FX` 100BASE-FX, 100Mb/s over fiber optic cables. [100baseFX, 100FX]

`IFM_100_T4` 100BASE-T4, 100Mb/s over 4-wire (category 3) unshielded twisted pair, RJ45 connector. [100baseT4, 100T4]

`IFM_100_T2` 100BASE-T2. [100baseT2, 100T2]

`IFM_100_VG` 100VG-AnyLAN. [100baseVG, 100VG]

`IFM_1000_SX` 1000BASE-SX, 1Gb/s over multi-mode fiber optic cables. (short waves) [1000baseSX, 1000SX]

`IFM_1000_LX` 1000BASE-LX, 1Gb/s over single-mode fiber optic cables. (long waves) [1000baseLX, 1000LX]

`IFM_1000_CX` 1000BASE-CX, 1Gb/s over shielded twisted pair. (twinax) [1000baseCX, 1000CX]

`IFM_1000_T` 1000BASE-T, 1Gb/s over category 5 unshielded twisted pair, 802.3ab, RJ45 connector. [1000baseT, 1000T]

`IFM_10G_LR` 10GBASE-LR, 10Gb/s over single-mode fiber optic cables. [10GbaseLR, 10GLR]

The following media option is defined for Ethernet:

`IFM_ETH_MASTER` Configure a 1000BASE-T PHY as the clock master for a 1000BASE-T link. This option has no effect (shows current status only) if the media is `IFM_AUTO`.

`IFM_ETH_TXPAUSE` Configure the device to send PAUSE (flow control) frames. This option has no effect (shows current status only) if the media is `IFM_AUTO`.

`IFM_ETH_RXPAUSE` Configure the device to receive PAUSE (flow control) frames. This option has no effect (shows current status only) if the media is `IFM_AUTO`.

MEDIA TYPES AND OPTIONS FOR TOKEN RING

The following media types are defined for Token Ring:

`IFM_TOK_STP4` 4Mb/s, shielded twisted pair, DB9 connector. [DB9/4Mbit, 4STP]

`IFM_TOK_STP16` 16Mb/s, shielded twisted pair, DB9 connector. [DB9/16Mbit, 16STP]

`IFM_TOK_UTP4` 4Mb/s, unshielded twisted pair, RJ45 connector. [UTP/4Mbit, 4UTP]

`IFM_TOK_UTP16` 16Mb/s, unshielded twisted pair, RJ45 connector. [UTP/16Mbit, 16UTP]

The following media options are defined for Token Ring:

`IFM_TOK_ETR` Early token release. [EarlyTokenRelease, ETR]

`IFM_TOK_SRCRT` Enable source routing features. [SourceRouting, SRCRT]

`IFM_TOK_ALLR` All routes vs. single route broadcast. [AllRoutes, ALLR]

MEDIA TYPES AND OPTIONS FOR FDDI

The following media types are defined for FDDI:

`IFM_FDDI_SMF` Single-mode fiber. [Single-mode, SMF]
`IFM_FDDI_MMF` Multi-mode fiber. [Multi-mode, MMF]
`IFM_FDDI_UTP` Unshielded twisted pair, RJ45 connector. [UTP, CDDI]

The following media options are defined for FDDI:

`IFM_FDDI_DA` Dual-attached station vs. Single-attached station. [dual-attach, das]

MEDIA TYPES AND OPTIONS FOR IEEE802.11 WIRELESS LAN

The following media types are defined for IEEE802.11 Wireless LAN:

`IFM_IEEE80211_FH1` Frequency Hopping 1Mbps. [FH1]
`IFM_IEEE80211_FH2` Frequency Hopping 2Mbps. [FH2]
`IFM_IEEE80211_DS1` Direct Sequence 1Mbps. [DS1]
`IFM_IEEE80211_DS2` Direct Sequence 2Mbps. [DS2]
`IFM_IEEE80211_DS5` Direct Sequence 5Mbps. [DS5]
`IFM_IEEE80211_DS11` Direct Sequence 11Mbps. [DS11]
`IFM_IEEE80211_DS22` Direct Sequence 22Mbps. [DS22]
`IFM_IEEE80211_OFDM6` Orthogonal Frequency Division Multiplexing 6Mbps. [OFDM6]
`IFM_IEEE80211_OFDM9` Orthogonal Frequency Division Multiplexing 9Mbps. [OFDM9]
`IFM_IEEE80211_OFDM12` Orthogonal Frequency Division Multiplexing 12Mbps. [OFDM12]
`IFM_IEEE80211_OFDM18` Orthogonal Frequency Division Multiplexing 18Mbps. [OFDM18]
`IFM_IEEE80211_OFDM24` Orthogonal Frequency Division Multiplexing 24Mbps. [OFDM24]
`IFM_IEEE80211_OFDM36` Orthogonal Frequency Division Multiplexing 36Mbps. [OFDM36]
`IFM_IEEE80211_OFDM48` Orthogonal Frequency Division Multiplexing 48Mbps. [OFDM48]
`IFM_IEEE80211_OFDM54` Orthogonal Frequency Division Multiplexing 54Mbps. [OFDM54]
`IFM_IEEE80211_OFDM72` Orthogonal Frequency Division Multiplexing 72Mbps. [OFDM72]

The following media options are defined for IEEE802.11 Wireless LAN:

`IFM_IEEE80211_ADHOC` Ad-hoc (IBSS) mode. [adhoc, ibss]
In some drivers, it may be used with the `IFM_FLAG0` [flag0] media option to specify non-standard ad-hoc demo mode.
`IFM_IEEE80211_HOSTAP` Access Point mode [hostap]
`IFM_IEEE80211_MONITOR` Monitor mode [monitor]
`IFM_IEEE80211_TURBO` Turbo mode [turbo]

SEE ALSO

`netintro`(4), `ifconfig`(8)

HISTORY

The **`ifmedia`** interface first appeared in BSD/OS 3.0. The implementation that appeared in NetBSD 1.3 was written by Jonathan Stone and Jason R. Thorpe to be compatible with the BSDI API. It has since gone through several revisions which have extended the API while maintaining backwards compatibility with the original API.

Support for the **IEEE802.11 Wireless LAN** link type was added in NetBSD 1.5.

NAME

ifpci — AVM Fritz!PCI ISDN card driver

SYNOPSIS

ifpci* **at pci?**

DESCRIPTION

The **ifpci** driver supports the popular Fritz!PCI card from AVM. While this card uses the same brand as its predecessors, those use a different chipset and are driven by the isic(4) driver.

The PCI subsystem configures the card automatically, there are no options for the driver.

SEE ALSO

isic(4), isdnd(8)

STANDARDS

CCITT Recommendation I.430

AUTHORS

The **ifpci** driver was written by Gary Jennejohn for FreeBSD. It was ported to NetBSD by Martin Husemann ⟨martin@NetBSD.org⟩.

NAME

igsfb — IGA 1682 and CyberPro series graphics cards

SYNOPSIS

```
igsfb* at pci?
wsdisplay* at igsfb?
```

DESCRIPTION

The **igsfb** driver provides support for IGA 1682 and CyberPro series of graphics cards by InteGraphics Systems, now Tvia. The **igsfb** driver operates these cards in linear framebuffer mode, not in VGA mode.

SEE ALSO

wscons(4)

HISTORY

The **igsfb** driver first appeared in NetBSD 1.6.

BUGS

There's currently no way to change resolution, frequency, etc. The driver is hardcoded to init the card to 1024×768, 8bpp at 60Hz.

NAME

iha — Initio INIC-940/950 based PCI SCSI host adapter driver

SYNOPSIS

```
iha* at pci? dev ? function ?
scsibus* at iha?
```

DESCRIPTION

The **iha** driver supports PCI SCSI host adapters based on the Initio INIC-940 and INIC-950 chips.

The INI-9090U, INI-9100U/UW, the Iwill 2935UW and the IOI Technology IOI-4203U have been tested.

SEE ALSO

cd(4), ch(4), intro(4), pci(4), scsi(4), sd(4), ss(4), st(4), uk(4), scsipi(9)

http://www.initio.com/

AUTHORS

The **iha** driver was originally written for OpenBSD by Kenneth R. Westerback, based on a FreeBSD driver by Winston Hung. Izumi Tsutsui ported the driver to NetBSD.

BUGS

iha driver does not currently use tagged command queuing.

NAME

ik — Ikonas frame buffer, graphics device interface

SYNOPSIS

```
ik0 at uba? csr 0172460 vector ikintr
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **ik** driver provides an interface to an Ikonas frame buffer graphics device. Each minor device is a different frame buffer interface board. When the device is opened, its interface registers are mapped, via virtual memory, into the user processes address space. This allows the user process very high bandwidth to the frame buffer with no system call overhead.

Bytes written or read from the device are DMA'ed from or to the interface. The frame buffer XY address, its addressing mode, etc. must be set up by the user process before calling write or read.

Other communication with the driver is via ioctls. The `IK_GETADDR` `ioctl`(2) returns the virtual address where the user process can find the interface registers. The `IK_WAITINT` `ioctl`(2) suspends the user process until the ikonas device has interrupted (for whatever reason — the user process has to set the interrupt enables).

FILES

`/dev/ik`

DIAGNOSTICS

None.

HISTORY

The **ik** driver appeared in 4.2BSD.

BUGS

An invalid access (e.g., longword) to a mapped interface register can cause the system to crash with a machine check. A user process could possibly cause infinite interrupts hence bringing things to a crawl.

NAME

ikphy — driver for Intel i82563 Kumeran 10/100/1000 Ethernet PHYs

SYNOPSIS

```
ikphy* at mii? phy ?
```

DESCRIPTION

The **ikphy** driver supports the Intel i82563 Kumeran 10/100/1000 Ethernet PHYs. These PHYs are found on a variety of high-performance Ethernet interfaces.

NOTES

While not required for basic card operation, the lack of this device will mean that the wm(4) device cannot detect link and link-speed.

SEE ALSO

ifmedia(4), mii(4), wm(4), ifconfig(8)

NAME

il — Interlan NI1010 10 Mb/s Ethernet interface

SYNOPSIS

```
il0 at uba0 csr 164000
```

DESCRIPTION

The **il** interface provides access to a 10 Mb/s Ethernet network through an Interlan 1010 or 1010A controller.

Each of the host's network addresses is specified at boot time with an SIOCSIFADDR ioctl(2). The **il** interface employs the address resolution protocol described in arp(4) to dynamically map between Internet and Ethernet addresses on the local network.

DIAGNOSTICS

il%d: input error. The hardware indicated an error in reading a packet off the cable or an illegally sized packet.

il%d: can't handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

il%d: setaddr didn't work. The interface was unable to reprogram its physical Ethernet address. This may happen with very early models of the interface. This facility is used only when the controller is not the first network interface configured for XNS. The oldest interface tested (2.7.1.0.1.45) has never failed in this way.

il%d: reset failed, csr=%b.
il%d: status failed, csr=%b.
il%d: hardware diag failed, csr=%b.
il%d: verifying setaddr, csr=%b.
il%d: stray xmit interrupt, csr=%b.
il%d: can't initialize. The above messages indicate a probable hardware error performing the indicated operation during autoconfiguration or initialization. The status field in the control and status register (the low-order four bits) should indicate the nature of the failure. See the hardware manual for details.

SEE ALSO

arp(4), inet(4), netintro(4)

HISTORY

The **il** interface appeared in 4.2BSD.

NAME

imc — Indy Memory Controller and system controller

SYNOPSIS

```
imc0 at mainbus0 addr 0x1fa00000
```

DESCRIPTION

The Indy Memory Controller is responsible for acting as an interface from the gio(4) bus to the main memory and CPU. The **imc** is found in the Indigo R4k, Indy, Challenge S, Challenge M, and Indigo2 machines.

SEE ALSO

gio(4)

HISTORY

The **imc** driver first appeared in NetBSD 1.6.

NAME

inet — Internet protocol family

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
```

DESCRIPTION

The Internet protocol family is a collection of protocols layered atop the *Internet Protocol* (IP) transport layer, and using the Internet address format. The Internet family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types; the SOCK_RAW interface provides access to the IP protocol.

ADDRESSING

Internet addresses are four byte quantities, stored in network standard format (on the VAX these are word and byte reversed). The include file ⟨netinet/in.h⟩ defines this address as a discriminated union.

Sockets bound to the Internet protocol family use the following addressing structure,

```
struct sockaddr_in {
        uint8_t         sin_len;
        sa_family_t     sin_family;
        in_port_t       sin_port;
        struct in_addr  sin_addr;
        int8_t          sin_zero[8];
};
```

Sockets may be created with the local address INADDR_ANY to effect "wildcard" matching on incoming messages. The address in a connect(2) or sendto(2) call may be given as INADDR_ANY to mean "this host". The distinguished address INADDR_BROADCAST is allowed as a shorthand for the broadcast address on the primary network if the first network configured supports broadcast.

PROTOCOLS

The Internet protocol family comprises the IP transport protocol, Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). TCP is used to support the SOCK_STREAM abstraction while UDP is used to support the SOCK_DGRAM abstraction. A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The ICMP message protocol is accessible from a raw socket.

The 32-bit Internet address contains both network and host parts. It is frequency-encoded; the most-significant bit is clear in Class A addresses, in which the high-order 8 bits are the network number. Class B addresses use the high-order 16 bits as the network field, and Class C addresses have a 24-bit network part. Sites with a cluster of local networks and a connection to the Internet may chose to use a single network number for the cluster; this is done by using subnet addressing. The local (host) portion of the address is further subdivided into subnet and host parts. Within a subnet, each subnet appears to be an individual network; externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry. Subnet addressing is enabled and examined by the following ioctl(2) commands on a datagram socket in the Internet domain; they have the same form as the SIOCIFADDR command (see netintro(4)).

SIOCSIFNETMASK Set interface network mask. The network mask defines the network part of the address; if it contains more of the address than the address type would indicate, then subnets are in use.

`SIOCGIFNETMASK` Get interface network mask.

SEE ALSO

`ioctl`(2), `socket`(2), `icmp`(4), `intro`(4), `ip`(4), `netintro`(4), `tcp`(4), `udp`(4)

Stuart Sechrest, *An Introductory 4.4BSD Interprocess Communication Tutorial*. (see `/usr/share/doc/psd/20.ipctut`)

Samuel J. Leffler, Robert S. Fabry, William N. Joy, Phil Lapsley, Steve Miller, and Chris Torek, *Advanced 4.4BSD IPC Tutorial*. (see `/usr/share/doc/psd/21.ipc`)

HISTORY

The **`inet`** protocol interface appeared in 4.2BSD.

BUGS

The Internet protocol support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

NAME

inet6 — Internet protocol version 6 family

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
```

DESCRIPTION

The **inet6** family is an updated version of inet(4) family. While inet(4) implements Internet Protocol version 4, **inet6** implements Internet Protocol version 6.

inet6 is a collection of protocols layered atop the *Internet Protocol version 6* (IPv6) transport layer, and using the IPv6 address format. The **inet6** family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types; the SOCK_RAW interface provides access to the IPv6 protocol.

ADDRESSING

IPv6 addresses are 16 byte quantities, stored in network standard byteorder. The include file ⟨netinet/in.h⟩ defines this address as a discriminated union.

Sockets bound to the **inet6** family use the following addressing structure:

```
struct sockaddr_in6 {
        uint8_t         sin6_len;
        sa_family_t     sin6_family;
        in_port_t       sin6_port;
        uint32_t        sin6_flowinfo;
        struct in6_addr         sin6_addr;
        uint32_t        sin6_scope_id;
};
```

Sockets may be created with the local address "::" (which is equal to IPv6 address 0:0:0:0:0:0:0:0) to effect "wildcard" matching on incoming messages.

The IPv6 specification defines scoped addresses, like link-local or site-local addresses. A scoped address is ambiguous to the kernel, if it is specified without a scope identifier. To manipulate scoped addresses properly from the userland, programs must use the advanced API defined in RFC 2292. A compact description of the advanced API is available in ip6(4). If a scoped address is specified without an explicit scope, the kernel may raise an error. Note that scoped addresses are not for daily use at this moment, both from a specification and an implementation point of view.

The KAME implementation supports an extended numeric IPv6 address notation for link-local addresses, like "fe80::1%de0" to specify "fe80::1 on de0 interface". This notation is supported by getaddrinfo(3) and getnameinfo(3). Some of normal userland programs, such as telnet(1) or ftp(1), are able to use this notation. With special programs like ping6(8), you can specify the outgoing interface by an extra command line option to disambiguate scoped addresses.

Scoped addresses are handled specially in the kernel. In kernel structures like routing tables or interface structures, a scoped address will have its interface index embedded into the address. Therefore, the address in some kernel structures is not the same as that on the wire. The embedded index will become visible through a PF_ROUTE socket, kernel memory accesses via kvm(3) and on some other occasions. HOWEVER, users should never use the embedded form. For details please consult http://www.kame.net/dev/cvsweb.cgi/kame/IMPLEMENTATION. Note that the above URL describes the situation with the latest KAME tree, not the NetBSD tree.

PROTOCOLS

The **inet6** family comprises the IPv6 network protocol, Internet Control Message Protocol version 6 (ICMPv6), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). TCP is used to support the SOCK_STREAM abstraction while UDP is used to support the SOCK_DGRAM abstraction. Note that TCP and UDP are common to inet(4) and **inet6**. A raw interface to IPv6 is available by creating an Internet socket of type SOCK_RAW. The ICMPv6 message protocol is accessible from a raw socket.

Interaction between IPv4/v6 sockets

By default, NetBSD does not route IPv4 traffic to AF_INET6 sockets. The default behavior intentionally violates RFC 2553 for security reasons. Listen to two sockets if you want to accept both IPv4 and IPv6 traffic. IPv4 traffic may be routed with certain per-socket/per-node configuration, however, it is not recommended to do so. Consult ip6(4) for details.

The behavior of AF_INET6 TCP/UDP socket is documented in RFC 2553. Basically, it says this:

- A specific bind on an AF_INET6 socket (bind(2) with an address specified) should accept IPv6 traffic to that address only.
- If you perform a wildcard bind on an AF_INET6 socket (bind(2) to IPv6 address ::), and there is no wildcard bind AF_INET socket on that TCP/UDP port, IPv6 traffic as well as IPv4 traffic should be routed to that AF_INET6 socket. IPv4 traffic should be seen as if it came from an IPv6 address like ::ffff:10.1.1.1. This is called an IPv4 mapped address.
- If there are both a wildcard bind AF_INET socket and a wildcard bind AF_INET6 socket on one TCP/UDP port, they should behave separately. IPv4 traffic should be routed to the AF_INET socket and IPv6 should be routed to the AF_INET6 socket.

However, RFC 2553 does not define the ordering constraint between calls to bind(2), nor how IPv4 TCP/UDP port numbers and IPv6 TCP/UDP port numbers relate to each other (should they be integrated or separated). Implemented behavior is very different from kernel to kernel. Therefore, it is unwise to rely too much upon the behavior of AF_INET6 wildcard bind sockets. It is recommended to listen to two sockets, one for AF_INET and another for AF_INET6, when you would like to accept both IPv4 and IPv6 traffic.

It should also be noted that malicious parties can take advantage of the complexity presented above, and are able to bypass access control, if the target node routes IPv4 traffic to AF_INET6 socket. Users are advised to take care handling connections from IPv4 mapped address to AF_INET6 sockets.

SEE ALSO

ioctl(2), socket(2), sysctl(3), icmp6(4), intro(4), ip6(4), tcp(4), udp(4)

STANDARDS

Tatsuya Jinmei and Atsushi Onoe, *An Extension of Format for IPv6 Scoped Addresses*, internet draft, draft-ietf-ipngwg-scopedaddr-format-02.txt, June 2000, work in progress material.

HISTORY

The **inet6** protocol interfaces are defined in RFC 2553 and RFC 2292. The implementation described herein appeared in the WIDE/KAME project.

BUGS

The IPv6 support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

Users are suggested to implement “version independent” code as much as possible, as you will need to support both inet(4) and **inet6**.

NAME

inphy — Driver for Intel i82555 10/100 Ethernet PHYs

SYNOPSIS

inphy* **at mii? phy ?**

DESCRIPTION

The **inphy** driver supports the Intel i82555 10/100 Ethernet PHYs. These PHYs are found on a variety of high-performance Ethernet interfaces.

NOTES

While not required for basic card operation, the lack of this device will mean that the fxp(4) device cannot detect link and link-speed.

SEE ALSO

fxp(4), ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

intersil7170 — Intersil ICM7170 time-of-day clock driver

SYNOPSIS

```
#include <dev/ic/intersil7170reg.h>
#include <dev/ic/intersil7170var.h>
define intersil7170
file   dev/ic/intersil7170.c    intersil7170
```

DESCRIPTION

The **intersil7170** driver provides access to the Intersil ICM7170 time-of-day clock chip. Access methods to retrieve and set date and time are provided through the *TODR* interface defined in todr(9).

To tie an instance of this device to the system, use the **intersil7170_attach**() function and the *intersil7170_softc* structure defined as follows:

void **intersil7170_attach**(*struct intersil7170_softc* *)

```
struct intersil7170_softc {
        struct device  sc_dev;
        bus_space_tag_t        sc_bst;
        bus_space_handle_t sc_bsh;
        struct todr_chip_handle sc_handle;
        u_int          sc_year0;
        u_int          sc_flag;
};
```

bus_tag

bus_handle Specify bus space access to the chip's non-volatile memory (including the clock registers).

sc_handle TODR handle passed to the **todr_attach**() function to register todr(9) interface.

sc_year0 The actual year represented by the clock's 'year' counter. This is generally dependent on the system configuration in which the clock device is mounted. For instance, on Sun Microsystems machines the convention is to have clock's two-digit year represent the year 1968.

sc_flag This flag is used to specify machine-dependent features.

Note that if the resulting date retrieved with the todr_gettime() method is earlier that January 1, 1970, the driver will assume that the chip's year counter actually represents a year in the 21st century. This behaviour can be overridden by setting the *INTERSIL7170_NO_CENT_ADJUST* flag in *sc_flag*.

SEE ALSO

intro(4), todr(9)

HISTORY

The **intersil7170** driver first appeared in NetBSD 1.5.

AUTHORS

The **intersil7170** driver was written by Paul Kranenburg ⟨pk@NetBSD.org⟩.

NAME

intio — hp300 internal I/O space driver

SYNOPSIS

```
intio0 at mainbus0
```

DESCRIPTION

intio is a virtual device corresponding to the hp300 internal I/O space.

Internal I/O space spans the hp300 physical address space, and is mapped permanently in the kernel virtual space at the very early time of the kernel startup procedure.

intio driver manages the internal I/O space of hp300.

- Address range management to avoid confliction of address space of which devices probe by touching hardware port is difficult.
- Interrupt vector management.
- bus_space(9) and bus_dma(9) implementation.
- Other utility functions.

intio is always required to run the NetBSD kernel.

SEE ALSO

bus_dma(9), bus_space(9)

NAME

intio — X68K internal I/O space driver

SYNOPSIS

```
intio0 at mainbus0
```

DESCRIPTION

intio is a virtual device corresponding to the x68k internal I/O space.

Internal I/O space spans from 0xc00000 to 0xffffff of the x68k physical address space, and is mapped permanently in the kernel virtual space at the very early time of the kernel startup procedure.

intio driver manages the internal I/O space of x68k.

- Address range management to avoid confliction of address space of which devices probe by touching hardware port is difficult.
- Interrupt vector management.
- bus_space(9) and bus_dma(9) implementation.
- Other utility functions.

intio is always required to run the NetBSD kernel.

SEE ALSO

intro(4), bus_dma(9), bus_space(9)

NAME

io — HP PA-RISC I/O subsystem reference

DESCRIPTION

The following table lists the PA-RISC I/O subsystems and connected devices found in the supported HP 9000/700 machines.

Model	MBA	SCSI	Network	Video	Misc
705	A	SE	DX	Timber	
710	A	SE	DX	Timber	
712	LW	SE	CA, TR	Artist	
715/33	A	SE	DX	Stinger	
715/50	A	SE	DX	Stinger	
715/64	LW	SE	CA	Artist	
715/75	A	SE	DX	Stinger	
715/80	LW	SE	CA	Artist	
715/100	LW	SE	CA	Artist	
720	A	SE	DX	[SGC]	
725/50	A	SE	DX	Stinger	
725/75	A	SE	DX	Stinger	
725/64	LW	SE	CA	Artist	
725/100	LW	SE	CA	Artist	
730	A	SE	DX	[SGC]	
735	A	SE, FWD	DX, FDDI	[SGC]	
742i	A	SE	CA	Stinger	VME
743i	DLW	SE	DX	Artist	VME
744	DLW	SE	DX	EG	VME
745	DLW	SE	DX	EG	VME
745i	A	SE	DX	Stinger	
747i	A	SE	DX	Stinger	
748	DLW	SE	DX	EG	VME
748i	DLW	SE	DX	Artist	VME
750	A	SE	DX	[SGC]	
755	A	SE, FWD	DX, FDDI	[SGC]	
J2x0/XC	LW	SE, FWD	CA	[GSC]	
A180	DL	SE	TLP	[GSC/PCI]	
B1xxL	DLW	SE, FWD	CA	EG	
B1xxL+	DLW	SE, UW	TLP	EG	
C1x0	LW	SE, FWD	CA	[GSC]	
C1xxL	DLW	SE, UW	TLP	EG	
RDI	DL	SE	CA	EG	CardBus
SAIC	LW	SE	CA	Artist	PCMCIA

The MBA column denotes the bus adapters in use:

A `asp`(4)
D `dino`(4)
L `lasi`(4)
W `wax`(4)

The trailing characters in the SCSI row denote the SCSI bus configuration:

SE `oosiop`(4) or `osiop`(4); Symbios/NCR 53C700/710 8-bit (fast) single-ended,
FWD `siop`(4); NCR53C720 16-bit fast differential (HVD),
UW `siop`(4); NCR53C875 16-bit ultra single-ended.

The trailing digits in the Network row denote the interface speed:

CA `iee`(4); i82596CA 10 Mb/s,
DX `iee`(4); i82596DX 10 Mb/s,
TLP `tlp`(4); DEC 21142/3 10/100 Mb/s,
FDDI Am78830 Formac+ FDDI,
TR TMS380C26 4/16 Mb/s TokenRing.

SEE ALSO

`asp`(4), `dino`(4), `gsc`(4), `iee`(4), `intro`(4), `lasi`(4), `oosiop`(4), `osiop`(4), `siop`(4), `sti`(4), `tlp`(4), `wax`(4), `http://www.openpa.net/`

HISTORY

The hp700 **`io`** reference first appeared with OpenBSD 3.3. It was ported to NetBSD 2.0 by Jochen Kunz.

NAME

io — I/O privilege file

DESCRIPTION

This device is obsolete and is provided for compatibility purposes only; use `i386_iopl`(2) **instead.**

After opening `/dev/io` for writing the process is granted full I/O privileges; closing the returned file descriptor does *not* result in revocation of these privileges. The new I/O privileges can be useful in order to write userland programs that handle some hardware directly.

The entire access control is handled by the file access permissions of `/dev/io`, so care should be taken in granting rights for this device.

I/O privilege on access to `/dev/io` is only granted if the kernel was built with the `COMPAT_10` option.

FILES

`/dev/io`

SEE ALSO

`mem`(4)

HISTORY

The **io** file appeared in NetBSD 1.0 after it was in the kernel for some time.

NAME

ioasic — baseboard IO control ASIC for DEC TURBOchannel systems

SYNOPSIS

```
ioasic0 at tc? slot ? offset ?
```

DESCRIPTION

The **ioasic** driver provides support for the DEC proprietary IOCTL ASIC found on all DEC TURBOchannel machines with MIPS (DECstation 5000 series, excluding the 5000/200) and Alpha (3000-series) processors. On these machines (including the 5000/200), all baseboard devices should be configured as children of the **ioasic** device.

The **ioasic** provides hardware DMA channels and interrupt support for several baseboard devices, including one **asc** SCSI device with a scatter/gather DMA channel, an mc146818-compatible **mcclock**, an Am7930 audio device **bba**, one or two **scc** two-port serial devices, and a AMD 7990 LANCE **le** Ethernet interface.

The **ioasic** is also used for the floppy-disc drive and audio/ISDN hardware on the Personal DECstation and audio-equipped TURBOchannel Alphas, where the **ioasic** hardware provides a scatter-gather DMA channel between the 16-bit device and the 32-bit **tc** DMA address space.

Support for scatter-gather DMA eliminates the need for additional copying. A baseboard **asc** SCSI adaptor attached to an **ioasic** will give slightly better performance than its **tc** counterpart.

SEE ALSO

asc(4), bba(4), intro(4), le(4), mcclock(4), scc(4), tc(4)

HISTORY

The **ioasic** driver first appeared in NetBSD 1.1, derived from DECstation boot-time configuration code in 4.4BSD.

BUGS

The DECstation 5000/200 does not actually have an IOASIC chip, but for consistency it must be configured as if it did.

NAME

ioat — multiplexing serial communications interface

SYNOPSIS

For 6-port BOCA IOAT66 board:

```
ioat0   at isa? port 0x220 irq 5
com2    at ioat? slave ?
com3    at ioat? slave ?
com4    at ioat? slave ?
com5    at ioat? slave ?
com6    at ioat? slave ?
com7    at ioat? slave ?
```

DESCRIPTION

The **ioat** driver provides support for BOCA Research IOAT66 boards that multiplex together up to six EIA RS-232C (CCITT V.28) communications interfaces.

Each **ioat** device is the master device for up to six **com** devices. The kernel configuration specifies these **com** devices as slave devices of the **ioat** device, as shown in the synopsis. The slave ID given for each **com** device determines which bit in the interrupt multiplexing register is tested to find interrupts for that device. The port specification for the **ioat** device is used to compute the base addresses for the **com** subdevices. The port for the interrupt multiplexing register is not programmable.

FILES

`/dev/tty??`

SEE ALSO

com(4)

HISTORY

The **ioat** driver was adapted from the boca driver by Michael Richardson.

NAME

iobus — Device driver for the I/O bus

SYNOPSIS

```
iobus0 at root
```

DESCRIPTION

Systems supported by NetBSD/acorn26 have a 16-bit I/O bus connected to the CPU bus by bi-directional buffers managed by the IOC. The **iobus** driver provides support for this bus. All NetBSD/acorn26 kernels need this driver.

On Archimedes systems, the only device attached at **iobus** is the IOC. While MEMC podules are physically **iobus** devices, they're logically treated as being attached to the IOC. On machines with an IOEB, the Universal Peripheral Controller chip is also attached at **iobus**.

SEE ALSO

podulebus(4)

NAME

iomdkbc — ARM IOMD keyboard and mouse ports

SYNOPSIS

```
iomdkbc* at iomd?
pckbd* at iomdkbc?
pms* at iomdkbc?
```

DESCRIPTION

The **iomdkbc** driver interfaces machine-independent drivers for PC keyboards and PS/2 mice to the particular implementation of the keyboard and mouse ports provided by the ARM IOMD20, ARM7500, and ARM7500FE. The quadrature mouse interface on the IOMD20 is handled by the qms(4) driver.

This driver replaces the kbd(4) and rpckbd(4) drivers, which will be withdrawn in a future NetBSD release. If a kernel has kbd(4) or rpckbd(4) configured, **iomdkbc** will only be used for the mouse interface (if present).

SEE ALSO

kbd(4), pckbd(4), pms(4), qms(4), rpckbd(4)

HISTORY

The **iomdkbc** driver was introduced in NetBSD 2.0.

NAME

iop — I2O adapter driver

SYNOPSIS

```
iop* at pci? dev ? function ?
iopsp* at iop? tid ?
ld* at iop? tid ?
dpti* at iop? tid 0
```

DESCRIPTION

The **iop** driver provides support for PCI I/O processors conforming to the I2O specification, revision 1.5 and above.

I2O is a specification that defines a software interface for communicating with a number of device types. In its basic form, I2O provides the following:

- A vendor-neutral interface for communicating with an I/O processor (IOP) and a number of types of peripherals. In order to achieve this, hardware-specific device drivers run on the IOP, and hardware-neutral device drivers run on the host.
- Reduced I/O overhead for the host. All communication between the host and the IOP is performed using a high level protocol. The specification also provides for batching of requests and replies between the host and IOP.
- An optional vendor-neutral configuration interface. Data from HTTP GET and POST operations can be channeled to individual devices, and HTML pages returned.

Five types of devices are well defined by the specification. These are:

- Random block storage devices (disks).
- Sequential storage devices (tapes).
- LAN interfaces, including Ethernet, FDDI, and Token Ring.
- Bus ports (SCSI).
- SCSI peripherals.

The **iop** driver's role is to initialize and monitor the IOP, provide a conduit for messages and replies to and from devices, and provide other common services for peripheral drivers, such as DMA mapping.

IOCTL INTERFACE

The following structures and constants are defined in `dev/i2o/iopio.h`. Note that the headers `sys/types.h`, `sys/device.h` and `dev/i2o/i2o.h` are prerequisites and must therefore be included beforehand.

`IOPIOCPT (struct ioppt)`

Submit a message to the IOP and return the reply. Note that the return value of this ioctl is not affected by completion status as indicated by the reply.

```
struct ioppt {
        void    *pt_msg;        /* pointer to message buffer */
        size_t  pt_msglen;      /* message buffer size in bytes */
        void    *pt_reply;      /* pointer to reply buffer */
        size_t  pt_replylen;    /* reply buffer size in bytes */
        int     pt_timo;        /* completion timeout in ms */
        int     pt_nbufs;       /* number of transfers */
        struct  ioppt_buf pt_bufs[IOP_MAX_MSG_XFERS]; /* transfers */
};
```

```
struct ioppt_buf {
        void    *ptb_data;      /* pointer to buffer */
        size_t  ptb_datalen;    /* buffer size in bytes */
        int     ptb_out;        /* non-zero if transfer is to IOP */
};
```

The minimum timeout value that may be specified is 1000ms. All other values must not exceed the **iop** driver's operational limits.

The initiator context and transaction context fields in the message frame will be filled by the **iop** driver. As such, this ioctl may not be used to send messages without a transaction context payload.

`IOPIOCGSTATUS (struct iovec)`
Request the latest available status record from the IOP. This special-case ioctl is provided as the I2O_EXEC_STATUS_GET message does not post replies, and can therefore not be safely issued using the IOPIOCPT ioctl.

The following ioctls may block while attempting to acquire the **iop** driver's configuration lock, and may fail if the acquisition times out.

`IOPIOCGLCT (struct iovec)`
Retrieve the **iop** driver's copy of the logical configuration table. This copy of the LCT matches the current device configuration, but is not necessarily the latest available version of the LCT.

`IOPIOCRECONFIG`
Request that the **iop** driver scan all bus ports, retrieve the latest version of the LCT, and attach or detach devices as necessary. Note that higher-level reconfiguration tasks (such as logically re-scanning SCSI busses) will not be performed by this ioctl.

`IOPIOCGTIDMAP (struct iovec)`
Retrieve the TID to device map. This map indicates which targets are configured, and what the corresponding device name for each is. Although at any given point it contains the same number of entries as the LCT, the number of entries should be determined using the iov_len field from the returned iovec.

```
struct iop_tidmap {
        u_short it_tid;
        u_short it_flags;
        char    it_dvname[sizeof(((struct device *)NULL)->dv_xname)];
};
#define IT_CONFIGURED  0x02    /* target configured */
```

FILES

`/dev/iop`*u* control device for IOP unit *u*

SEE ALSO

`dpti`(4), `intro`(4), `iopsp`(4), `ld`(4), `iopctl`(8)

`http://www.intelligent-io.com/`

HISTORY

The **iop** driver first appeared in NetBSD 1.5.3.

NAME

iopaau — Intel I/O Processor Application Accelerator Unit

SYNOPSIS

```
iopxs* at mainbus?
iopaau* at iopxs?
```

DESCRIPTION

The Application Accelerator Unit, or AAU, provides hardware-assisted support for performing block fills on a region of memory, XOR of multiple regions of memory (parity computation), and parity verification.

The **iopaau** driver supports the Application Accelerator Units on the following Intel I/O Processors:

- Intel i80321 I/O Processor

The **iopaau** driver provides a back-end to the dmover(9) interface, and supports the following dmover(9) functions:

zero	Zero a region of memory
fill8	Fill a region of memory with an 8-bit value
copy	Copy a region of memory
xor2	Perform an XOR of 2 input streams
xor3	Perform an XOR of 3 input streams
xor4	Perform an XOR of 4 input streams
xor5	Perform an XOR of 5 input streams
xor6	Perform an XOR of 6 input streams
xor7	Perform an XOR of 7 input streams
xor8	Perform an XOR of 8 input streams

SEE ALSO

dmover(9)

HISTORY

The **iopaau** device first appeared in NetBSD 2.0.

AUTHORS

The **iopaau** driver was written by Jason R. Thorpe ⟨thorpej@wasabisystems.com⟩ and contributed by Wasabi Systems, Inc.

BUGS

Due to limitations in how scatter-gather is done by the AAU hardware, a given DMA segment must be the same length for the output stream and each input stream. The easiest way to achieve this is to ensure that all streams used in an AAU operation begin at the same offset into a page.

NAME

iophy — Driver for Intel i82553 10/100 Ethernet PHYs

SYNOPSIS

iophy* **at mii? phy ?**

DESCRIPTION

The **iophy** driver supports the Intel i82553 10/100 Ethernet PHYs found on some Intel i82557-based Ethernet cards.

SEE ALSO

fxp(4), ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

iopsp — I2O SCSI port driver

SYNOPSIS

```
iopsp* at iop? tid ?
scsibus* at iopsp?
```

DESCRIPTION

The **iopsp** driver provides support for I2O SCSI bus adapter ports and child peripherals.

IOPs present each child peripheral attached to a bus adapter port as an individual device. In order to present the appearance of a bus, the **iopsp** driver groups child peripherals by controlling port.

On IOPs containing a SCSI port and block or tape driver modules, some SCSI devices may not be directly accessible. For each inaccessible device, a message will be displayed during configuration. For example:

```
iopsp0: target 0,0 (tid 70): in use by tid 47
```

Such devices will usually be indirectly accessible as block devices, either individually or as part of an array.

SEE ALSO

intro(4), iop(4), ld(4), scsi(4)

HISTORY

The **iopsp** driver first appeared in NetBSD 1.5.3.

NAME

iopwdog — Intel I/O Processor Watchdog Timer

SYNOPSIS

```
iopxs* at mainbus?
iopwdog* at iopxs?
```

DESCRIPTION

The **iopwdog** driver supports the watchdog timer built-in to the following Intel I/O Processors:

- Intel i80321 I/O Processor

The watchdog timer counts down at a fixed rate based on the clock speed of the XScale CPU core. The period of the watchdog timer cannot be changed, nor can the watchdog time be disabled once is has been enabled.

SEE ALSO

wdogctl(8)

HISTORY

The **iopwdog** device first appeared in NetBSD 2.0.

AUTHORS

The **iopwdog** driver was written by Jason R. Thorpe ⟨thorpej@wasabisystems.com⟩ and contributed by Wasabi Systems, Inc.

NAME

ip — Internet Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

int
socket(AF_INET, SOCK_RAW, proto);
```

DESCRIPTION

IP is the network layer protocol used by the Internet protocol family. Options may be set at the IP level when using higher-level protocols that are based on IP (such as TCP and UDP). It may also be accessed through a "raw socket" when developing new protocols, or special-purpose applications.

There are several IP-level `setsockopt`(2)/`getsockopt`(2) options. `IP_OPTIONS` may be used to provide IP options to be transmitted in the IP header of each outgoing packet or to examine the header options on incoming packets. IP options may be used with any socket type in the Internet family. The format of IP options to be sent is that specified by the IP protocol specification (RFC 791), with one exception: the list of addresses for Source Route options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. To disable previously specified options, use a zero-length buffer:

```
setsockopt(s, IPPROTO_IP, IP_OPTIONS, NULL, 0);
```

`IP_TOS` and `IP_TTL` may be used to set the type-of-service and time-to-live fields in the IP header for `SOCK_STREAM` and `SOCK_DGRAM` sockets. For example,

```
int tos = IPTOS_LOWDELAY;       /* see <netinet/in.h> */
setsockopt(s, IPPROTO_IP, IP_TOS, &tos, sizeof(tos));

int ttl = 60;                   /* max = 255 */
setsockopt(s, IPPROTO_IP, IP_TTL, &ttl, sizeof(ttl));
```

`IP_IPSEC_POLICY` controls IPSec policy for sockets. For example,

```
const char *policy = "in ipsec ah/transport//require";
char *buf = ipsec_set_policy(policy, strlen(policy));
setsockopt(s, IPPROTO_IP, IP_IPSEC_POLICY, buf, ipsec_get_policylen(buf));
```

`IP_PORTRANGE` controls how ephemeral ports are allocated for `SOCK_STREAM` and `SOCK_DGRAM` sockets. For example,

```
int range = IP_PORTRANGE_LOW;       /* see <netinet/in.h> */
setsockopt(s, IPPROTO_IP, IP_PORTRANGE, &range, sizeof(range));
```

If the `IP_RECVDSTADDR` option is enabled on a `SOCK_DGRAM` or `SOCK_RAW` socket, the `recvmsg`(2) call will return the destination IP address for a UDP datagram. The msg_control field in the msghdr structure points to a buffer that contains a cmsghdr structure followed by the IP address. The cmsghdr fields have the following values:

```
cmsg_len = sizeof(struct in_addr)
cmsg_level = IPPROTO_IP
cmsg_type = IP_RECVDSTADDR
```

If the `IP_RECVIF` option is enabled on a `SOCK_DGRAM` or `SOCK_RAW` socket, the `recvmsg`(2) call will return a struct sockaddr_dl corresponding to the interface on which the packet was received. the msg_control field in the msghdr structure points to a buffer that contains a cmsghdr structure followed by the struct sock-

addr_dl. The cmsghdr fields have the following values:

```
cmsg_len = sizeof(struct sockaddr_dl)
cmsg_level = IPPROTO_IP
cmsg_type = IP_RECVIF
```

MULTICAST OPTIONS

IP multicasting is supported only on AF_INET sockets of type SOCK_DGRAM and SOCK_RAW, and only on networks where the interface driver supports multicasting.

The IP_MULTICAST_TTL option changes the time-to-live (TTL) for outgoing multicast datagrams in order to control the scope of the multicasts:

```
u_char ttl;    /* range: 0 to 255, default = 1 */
setsockopt(s, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));
```

Datagrams with a TTL of 1 are not forwarded beyond the local network. Multicast datagrams with a TTL of 0 will not be transmitted on any network, but may be delivered locally if the sending host belongs to the destination group and if multicast loopback has not been disabled on the sending socket (see below). Multicast datagrams with TTL greater than 1 may be forwarded to other networks if a multicast router is attached to the local network.

For hosts with multiple interfaces, each multicast transmission is sent from the primary network interface. The IP_MULTICAST_IF option overrides the default for subsequent transmissions from a given socket:

```
struct in_addr addr;
setsockopt(s, IPPROTO_IP, IP_MULTICAST_IF, &addr, sizeof(addr));
```

where "addr" is the local IP address of the desired interface or INADDR_ANY to specify the default interface. An interface's local IP address and multicast capability can be obtained via the SIOCGIFCONF and SIOCGIFFLAGS ioctls. An application may also specify an alternative to the default network interface by index:

```
struct uint32_t idx = htonl(ifindex);
setsockopt(s, IPPROTO_IP, IP_MULTICAST_IF, &idx, sizeof(idx));
```

where "ifindex" is an interface index as returned by if_nametoindex(3).

Normal applications should not need to use IP_MULTICAST_IF.

If a multicast datagram is sent to a group to which the sending host itself belongs (on the outgoing interface), a copy of the datagram is, by default, looped back by the IP layer for local delivery. The IP_MULTICAST_LOOP option gives the sender explicit control over whether or not subsequent datagrams are looped back:

```
u_char loop;   /* 0 = disable, 1 = enable (default) */
setsockopt(s, IPPROTO_IP, IP_MULTICAST_LOOP, &loop, sizeof(loop));
```

This option improves performance for applications that may have no more than one instance on a single host (such as a router demon), by eliminating the overhead of receiving their own transmissions. It should generally not be used by applications for which there may be more than one instance on a single host (such as a conferencing program) or for which the sender does not belong to the destination group (such as a time querying program).

A multicast datagram sent with an initial TTL greater than 1 may be delivered to the sending host on a different interface from that on which it was sent, if the host belongs to the destination group on that other interface. The loopback control option has no effect on such delivery.

A host must become a member of a multicast group before it can receive datagrams sent to the group. To join a multicast group, use the IP_ADD_MEMBERSHIP option:

```
struct ip_mreq mreq;
setsockopt(s, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));
```

where *mreq* is the following structure:

```
struct ip_mreq {
    struct in_addr imr_multiaddr; /* multicast group to join */
    struct in_addr imr_interface; /* interface to join on */
}
```

imr_interface should be INADDR_ANY to choose the default multicast interface, or the IP address of a particular multicast-capable interface if the host is multihomed. Membership is associated with a single interface; programs running on multihomed hosts may need to join the same group on more than one interface. Up to IP_MAX_MEMBERSHIPS (currently 20) memberships may be added on a single socket.

To drop a membership, use:

```
struct ip_mreq mreq;
setsockopt(s, IPPROTO_IP, IP_DROP_MEMBERSHIP, &mreq, sizeof(mreq));
```

where *mreq* contains the same values as used to add the membership. Memberships are dropped when the socket is closed or the process exits.

RAW IP SOCKETS

Raw IP sockets are connectionless, and are normally used with the sendto(2) and recvfrom(2) calls, though the connect(2) call may also be used to fix the destination for future packets (in which case the read(2) or recv(2) and write(2) or send(2) system calls may be used).

If *proto* is 0, the default protocol IPPROTO_RAW is used for outgoing packets, and only incoming packets destined for that protocol are received. If *proto* is non-zero, that protocol number will be used on outgoing packets and to filter incoming packets.

Outgoing packets automatically have an IP header prepended to them (based on the destination address and the protocol number the socket is created with), unless the IP_HDRINCL option has been set. Incoming packets are received with IP header and options intact.

IP_HDRINCL indicates the complete IP header is included with the data and may be used only with the SOCK_RAW type.

```
#include <netinet/ip.h>

int hincl = 1;                  /* 1 = on, 0 = off */
setsockopt(s, IPPROTO_IP, IP_HDRINCL, &hincl, sizeof(hincl));
```

Unlike previous BSD releases, the program must set all the fields of the IP header, including the following:

```
ip->ip_v = IPVERSION;
ip->ip_hl = hlen >> 2;
ip->ip_id = 0;  /* 0 means kernel set appropriate value */
ip->ip_off = offset;
```

If the header source address is set to INADDR_ANY, the kernel will choose an appropriate address.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
[ENOTCONN]	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
[ENOBUFS]	when the system runs out of memory for an internal data structure;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.
[EACCES]	when an attempt is made to create a raw IP socket by a non-privileged process.

The following errors specific to IP may occur when setting or getting IP options:

[EINVAL]	An unknown socket option name was given.
[EINVAL]	The IP option field was improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.

SEE ALSO

getsockopt(2), recv(2), send(2), ipsec_set_policy(3), icmp(4), inet(4), intro(4)

Internet Protocol, RFC, 791, September 1981.

Host Extensions for IP Multicasting, RFC, 1112, August 1989.

Requirements for Internet Hosts -- Communication Layers, RFC, 1122, October 1989.

HISTORY

The **ip** protocol appeared in 4.2BSD.

NAME

ip6 — Internet Protocol version 6 (IPv6) network layer

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

int
socket(AF_INET6, SOCK_RAW, proto);
```

DESCRIPTION

The IPv6 network layer is used by the IPv6 protocol family for transporting data. IPv6 packets contain an IPv6 header that is not provided as part of the payload contents when passed to an application. IPv6 header options affect the behavior of this protocol and may be used by high-level protocols (such as the tcp(4) and udp(4) protocols) as well as directly by "raw sockets", which process IPv6 messages at a lower-level and may be useful for developing new protocols and special-purpose applications.

Header

All IPv6 packets begin with an IPv6 header. When data received by the kernel are passed to the application, this header is not included in buffer, even when raw sockets are being used. Likewise, when data are sent to the kernel for transmit from the application, the buffer is not examined for an IPv6 header: the kernel always constructs the header. To directly access IPv6 headers from received packets and specify them as part of the buffer passed to the kernel, link-level access (bpf(4), for example) must be used instead.

The header has the following definition:

```
struct ip6_hdr {
     union {
          struct ip6_hdrctl {
               uint32_t ip6_un1_flow; /* 20 bits of flow ID */
               uint16_t ip6_un1_plen; /* payload length */
               uint8_t  ip6_un1_nxt;  /* next header */
               uint8_t  ip6_un1_hlim; /* hop limit */
          } ip6_un1;
          uint8_t ip6_un2_vfc;   /* version and class */
     } ip6_ctlun;
     struct in6_addr ip6_src; /* source address */
     struct in6_addr ip6_dst; /* destination address */
} __packed;

#define ip6_vfc              ip6_ctlun.ip6_un2_vfc
#define ip6_flow      ip6_ctlun.ip6_un1.ip6_un1_flow
#define ip6_plen      ip6_ctlun.ip6_un1.ip6_un1_plen
#define ip6_nxt              ip6_ctlun.ip6_un1.ip6_un1_nxt
#define ip6_hlim      ip6_ctlun.ip6_un1.ip6_un1_hlim
#define ip6_hops      ip6_ctlun.ip6_un1.ip6_un1_hlim
```

All fields are in network-byte order. Any options specified (see **Options** below) must also be specified in network-byte order.

ip6_flow specifies the flow ID. *ip6_plen* specifies the payload length. *ip6_nxt* specifies the type of the next header. *ip6_hlim* specifies the hop limit.

The top 4 bits of *ip6_vfc* specify the class and the bottom 4 bits specify the version.

ip6_src and *ip6_dst* specify the source and destination addresses.

The IPv6 header may be followed by any number of extension headers that start with the following generic definition:

```
struct ip6_ext {
     uint8_t ip6e_nxt;
     uint8_t ip6e_len;
} __packed;
```

Options

IPv6 allows header options on packets to manipulate the behavior of the protocol. These options and other control requests are accessed with the `getsockopt`(2) and `setsockopt`(2) system calls at level `IPPROTO_IPV6` and by using ancillary data in `recvmsg`(2) and `sendmsg`(2). They can be used to access most of the fields in the IPv6 header and extension headers.

The following socket options are supported:

`IPV6_UNICAST_HOPS` *int* ∗
: Get or set the default hop limit header field for outgoing unicast datagrams sent on this socket. A value of −1 resets to the default value.

`IPV6_MULTICAST_IF` *u_int* ∗
: Get or set the interface from which multicast packets will be sent. For hosts with multiple interfaces, each multicast transmission is sent from the primary network interface. The interface is specified as its index as provided by `if_nametoindex`(3). A value of zero specifies the default interface.

`IPV6_MULTICAST_HOPS` *int* ∗
: Get or set the default hop limit header field for outgoing multicast datagrams sent on this socket. This option controls the scope of multicast datagram transmissions.

  Datagrams with a hop limit of 1 are not forwarded beyond the local network. Multicast datagrams with a hop limit of zero will not be transmitted on any network but may be delivered locally if the sending host belongs to the destination group and if multicast loopback (see below) has not been disabled on the sending socket. Multicast datagrams with a hop limit greater than 1 may be forwarded to the other networks if a multicast router (such as `mrouted`(8)) is attached to the local network.

`IPV6_MULTICAST_LOOP` *u_int* ∗
: Get or set the status of whether multicast datagrams will be looped back for local delivery when a multicast datagram is sent to a group to which the sending host belongs.

  This option improves performance for applications that may have no more than one instance on a single host (such as a router daemon) by eliminating the overhead of receiving their own transmissions. It should generally not be used by applications for which there may be more than one instance on a single host (such as a conferencing program) or for which the sender does not belong to the destination group (such as a time-querying program).

  A multicast datagram sent with an initial hop limit greater than 1 may be delivered to the sending host on a different interface from that on which it was sent if the host belongs to the destination group on that other interface. The multicast loopback control option has no effect on such delivery.

`IPV6_JOIN_GROUP` *struct ipv6_mreq* ∗
: Join a multicast group. A host must become a member of a multicast group before it can receive datagrams sent to the group.

```
struct ipv6_mreq {
        struct in6_addr        ipv6mr_multiaddr;
        unsigned int   ipv6mr_interface;
};
```

ipv6mr_interface may be set to zeroes to choose the default multicast interface or to the index of a particular multicast-capable interface if the host is multihomed. Membership is associated with a single interface; programs running on multihomed hosts may need to join the same group on more than one interface.

If the multicast address is unspecified (i.e., all zeroes), messages from all multicast addresses will be accepted by this group. Note that setting to this value requires superuser privileges.

`IPV6_LEAVE_GROUP` *`struct ipv6_mreq *`*

Drop membership from the associated multicast group. Memberships are automatically dropped when the socket is closed or when the process exits.

`IPV6_IPSEC_POLICY` *`struct sadb_x_policy *`*

Get or set IPSec policy for sockets. For example,

```
const char *policy = "in ipsec ah/transport//require";
char *buf = ipsec_set_policy(policy, strlen(policy));
setsockopt(s, IPPROTO_IPV6, IPV6_IPSEC_POLICY, buf, ipsec_get_policylen(buf));
```

`IPV6_PORTRANGE` *`int *`*

Get or set the allocation policy of ephemeral ports for when the kernel automatically binds a local address to this socket. The following values are available:

`IPV6_PORTRANGE_DEFAULT`	Use the regular range of non-reserved ports (varies, see `sysctl`(8)).
`IPV6_PORTRANGE_HIGH`	Use a high range (varies, see `sysctl`(8)).
`IPV6_PORTRANGE_LOW`	Use a low, reserved range (600−1023).

`IPV6_PKTINFO` *`int *`*

Get or set whether additional information about subsequent packets will be provided as ancillary data along with the payload in subsequent `recvmsg`(2) calls. The information is stored in the following structure in the ancillary data returned:

```
struct in6_pktinfo {
        struct in6_addr ipi6_addr;    /* src/dst IPv6 address */
        unsigned int    ipi6_ifindex; /* send/recv if index */
};
```

`IPV6_HOPLIMIT` *`int *`*

Get or set whether the hop limit header field from subsequent packets will be provided as ancillary data along with the payload in subsequent `recvmsg`(2) calls. The value is stored as an *`int`* in the ancillary data returned.

`IPV6_HOPOPTS` *`int *`*

Get or set whether the hop-by-hop options from subsequent packets will be provided as ancillary data along with the payload in subsequent `recvmsg`(2) calls. The option is stored in the following structure in the ancillary data returned:

```
struct ip6_hbh {
        uint8_t ip6h_nxt;      /* next header */
        uint8_t ip6h_len;      /* length in units of 8 octets */
/* followed by options */
} __packed;
```

The **inet6_option_space**() routine and family of routines may be used to manipulate this data.

This option requires superuser privileges.

IPV6_DSTOPTS *int ∗*

Get or set whether the destination options from subsequent packets will be provided as ancillary data along with the payload in subsequent recvmsg(2) calls. The option is stored in the following structure in the ancillary data returned:

```
struct ip6_dest {
        uint8_t ip6d_nxt;      /∗ next header ∗/
        uint8_t ip6d_len;      /∗ length in units of 8 octets ∗/
/∗ followed by options ∗/
} __packed;
```

The **inet6_option_space**() routine and family of routines may be used to manipulate this data.

This option requires superuser privileges.

IPV6_RTHDR *int ∗*

Get or set whether the routing header from subsequent packets will be provided as ancillary data along with the payload in subsequent recvmsg(2) calls. The header is stored in the following structure in the ancillary data returned:

```
struct ip6_rthdr {
        uint8_t ip6r_nxt;      /∗ next header ∗/
        uint8_t ip6r_len;      /∗ length in units of 8 octets ∗/
        uint8_t ip6r_type;     /∗ routing type ∗/
        uint8_t ip6r_segleft;  /∗ segments left ∗/
/∗ followed by routing-type-specific data ∗/
} __packed;
```

The **inet6_option_space**() routine and family of routines may be used to manipulate this data.

This option requires superuser privileges.

IPV6_PKTOPTIONS *struct cmsghdr ∗*

Get or set all header options and extension headers at one time on the last packet sent or received on the socket. All options must fit within the size of an mbuf (see mbuf(9)). Options are specified as a series of *cmsghdr* structures followed by corresponding values. *cmsg_level* is set to IPPROTO_IPV6, *cmsg_type* to one of the other values in this list, and trailing data to the option value. When setting options, if the length *optlen* to setsockopt(2) is zero, all header options will be reset to their default values. Otherwise, the length should specify the size the series of control messages consumes.

Instead of using sendmsg(2) to specify option values, the ancillary data used in these calls that correspond to the desired header options may be directly specified as the control message in the series of control messages provided as the argument to setsockopt(2).

IPV6_CHECKSUM *int ∗*

Get or set the byte offset into a packet where the 16-bit checksum is located. When set, this byte offset is where incoming packets will be expected to have checksums of their data stored and where outgoing packets will have checksums of their data computed and stored by the kernel. A value of −1 specifies that no checksums will be checked on incoming packets and that no checksums will be computed or stored on outgoing packets. The offset of the checksum for ICMPv6 sockets cannot be relocated or turned off.

`IPV6_V6ONLY` *int* ∗
Get or set whether only IPv6 connections can be made to this socket. For wildcard sockets, this can restrict connections to IPv6 only.

`IPV6_FAITH` *int* ∗
Get or set the status of whether `faith`(4) connections can be made to this socket.

`IPV6_USE_MIN_MTU` *int* ∗
Get or set whether the minimal IPv6 maximum transmission unit (MTU) size will be used to avoid fragmentation from occurring for subsequent outgoing datagrams.

`IPV6_AUTH_LEVEL` *int* ∗
Get or set the `ipsec`(4) authentication level.

`IPV6_ESP_TRANS_LEVEL` *int* ∗
Get or set the ESP transport level.

`IPV6_ESP_NETWORK_LEVEL` *int* ∗
Get or set the ESP encapsulation level.

`IPV6_IPCOMP_LEVEL` *int* ∗
Get or set the `ipcomp`(4) level.

The `IPV6_PKTINFO`, `IPV6_HOPLIMIT`, `IPV6_HOPOPTS`, `IPV6_DSTOPTS`, and `IPV6_RTHDR` options will return ancillary data along with payload contents in subsequent `recvmsg`(2) calls with *cmsg_level* set to `IPPROTO_IPV6` and *cmsg_type* set to respective option name value (e.g., `IPV6_HOPTLIMIT`). These options may also be used directly as ancillary *cmsg_type* values in `sendmsg`(2) to set options on the packet being transmitted by the call. The *cmsg_level* value must be `IPPROTO_IPV6`. For these options, the ancillary data object value format is the same as the value returned as explained for each when received with `recvmsg`(2).

Note that using `sendmsg`(2) to specify options on particular packets works only on UDP and raw sockets. To manipulate header options for packets on TCP sockets, only the socket options may be used.

In some cases, there are multiple APIs defined for manipulating an IPv6 header field. A good example is the outgoing interface for multicast datagrams, which can be set by the `IPV6_MULTICAST_IF` socket option, through the `IPV6_PKTINFO` option, and through the *sin6_scope_id* field of the socket address passed to the `sendto`(2) system call.

Resolving these conflicts is implementation dependent. This implementation determines the value in the following way: options specified by using ancillary data (i.e., `sendmsg`(2)) are considered first, options specified by using `IPV6_PKTOPTIONS` to set “sticky” options are considered second, options specified by using the individual, basic, and direct socket options (e.g., `IPV6_UNICAST_HOPS`) are considered third, and options specified in the socket address supplied to `sendto`(2) are the last choice.

Multicasting
IPv6 multicasting is supported only on `AF_INET6` sockets of type `SOCK_DGRAM` and `SOCK_RAW`, and only on networks where the interface driver supports multicasting. Socket options (see above) that manipulate membership of multicast groups and other multicast options include `IPV6_MULTICAST_IF`, `IPV6_MULTICAST_HOPS`, `IPV6_MULTICAST_LOOP`, `IPV6_LEAVE_GROUP`, and `IPV6_JOIN_GROUP`.

Raw Sockets
Raw IPv6 sockets are connectionless and are normally used with the `sendto`(2) and `recvfrom`(2) calls, although the `connect`(2) call may be used to fix the destination address for future outgoing packets so that `send`(2) may instead be used and the `bind`(2) call may be used to fix the source address for future outgoing packets instead of having the kernel choose a source address.

By using connect(2) or bind(2), raw socket input is constrained to only packets with their source address matching the socket destination address if connect(2) was used and to packets with their destination address matching the socket source address if bind(2) was used.

If the *proto* argument to socket(2) is zero, the default protocol (IPPROTO_RAW) is used for outgoing packets. For incoming packets, protocols recognized by kernel are **not** passed to the application socket (e.g., tcp(4) and udp(4)) except for some ICMPv6 messages. The ICMPv6 messages not passed to raw sockets include echo, timestamp, and address mask requests. If *proto* is non-zero, only packets with this protocol will be passed to the socket.

IPv6 fragments are also not passed to application sockets until they have been reassembled. If reception of all packets is desired, link-level access (such as bpf(4)) must be used instead.

Outgoing packets automatically have an IPv6 header prepended to them (based on the destination address and the protocol number the socket was created with). Incoming packets are received by an application without the IPv6 header or any extension headers.

Outgoing packets will be fragmented automatically by the kernel if they are too large. Incoming packets will be reassembled before being sent to the raw socket, so packet fragments or fragment headers will never be seen on a raw socket.

EXAMPLES

The following determines the hop limit on the next packet received:

```
struct iovec iov[2];
u_char buf[BUFSIZ];
struct cmsghdr *cm;
struct msghdr m;
int found, optval;
u_char data[2048];

/* Create socket. */

(void)memset(&m, 0, sizeof(m));
(void)memset(&iov, 0, sizeof(iov));

iov[0].iov_base = data;              /* buffer for packet payload */
iov[0].iov_len = sizeof(data);       /* expected packet length */

m.msg_name = &from;           /* sockaddr_in6 of peer */
m.msg_namelen = sizeof(from);
m.msg_iov = iov;
m.msg_iovlen = 1;
m.msg_control = buf;  /* buffer for control messages */
m.msg_controllen = sizeof(buf);

/*
 * Enable the hop limit value from received packets to be
 * returned along with the payload.
 */
optval = 1;
if (setsockopt(s, IPPROTO_IPV6, IPV6_HOPLIMIT, &optval,
    sizeof(optval)) == -1)
        err(1, "setsockopt");
```

```
found = 0;
while (!found) {
       if (recvmsg(s, &m, 0) == -1)
              err(1, "recvmsg");
       for (cm = CMSG_FIRSTHDR(&m); cm != NULL;
            cm = CMSG_NXTHDR(&m, cm)) {
              if (cm->cmsg_level == IPPROTO_IPV6 &&
                  cm->cmsg_type == IPV6_HOPLIMIT &&
                  cm->cmsg_len == CMSG_LEN(sizeof(int))) {
                      found = 1;
                      (void)printf("hop limit: %d\n",
                          *(int *)CMSG_DATA(cm));
                      break;
              }
       }
}
```

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN] when trying to establish a connection on a socket which already has one or when trying to send a datagram with the destination address specified and the socket is already connected.

[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected.

[ENOBUFS] when the system runs out of memory for an internal data structure.

[EADDRNOTAVAIL] when an attempt is made to create a socket with a network address for which no network interface exists.

[EACCES] when an attempt is made to create a raw IPv6 socket by a non-privileged process.

The following errors specific to IPv6 may occur when setting or getting header options:

[EINVAL] An unknown socket option name was given.

[EINVAL] An ancillary data object was improperly formed.

SEE ALSO

getsockopt(2), recv(2), send(2), setsockopt(2), socket(2), if_nametoindex(3), bpf(4), icmp6(4), inet6(4), netintro(4), tcp(4), udp(4)

W. Stevens and M. Thomas, *Advanced Sockets API for IPv6*, RFC 2292, February 1998.

S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 2460, December 1998.

R. Gilligan, S. Thomson, J. Bound, and W. Stevens, *Basic Socket Interface Extensions for IPv6*, RFC 2553, March 1999.

W. Stevens, B. Fenner, and A. Rudoff, *UNIX Network Programming, third edition*.

STANDARDS

Most of the socket options are defined in RFC 2292 or RFC 2553. The IPV6_V6ONLY socket option is defined in RFC 3542. The IPV6_PORTRANGE socket option and the conflict resolution rule are not defined in the RFCs and should be considered implementation dependent.

NAME

ipf − packet filtering kernel interface

SYNOPSIS

#include <netinet/ip_compat.h>
#include <netinet/ip_fil.h>

IOCTLS

To add and delete rules to the filter list, three 'basic' ioctls are provided for use. The ioctl's are called as:

ioctl(fd, SIOCADDFR, struct frentry **)
ioctl(fd, SIOCDELFR, struct frentry **)
ioctl(fd, SIOCIPFFL, int *)

However, the full complement is as follows:

ioctl(fd, SIOCADAFR, struct frentry **) (same as SIOCADDFR)
ioctl(fd, SIOCRMAFR, struct frentry **) (same as SIOCDELFR)
ioctl(fd, SIOCADIFR, struct frentry **)
ioctl(fd, SIOCRMIFR, struct frentry **)
ioctl(fd, SIOCINAFR, struct frentry **)
ioctl(fd, SIOCINIFR, struct frentry **)
ioctl(fd, SIOCSETFF, u_int *)
ioctl(fd, SIOGGETFF, u_int *)
ioctl(fd, SIOCGETFS, struct friostat **)
ioctl(fd, SIOCIPFFL, int *)
ioctl(fd, SIOCIPFFB, int *)
ioctl(fd, SIOCSWAPA, u_int *)
ioctl(fd, SIOCFRENB, u_int *)
ioctl(fd, SIOCFRSYN, u_int *)
ioctl(fd, SIOCFRZST, struct friostat **)
ioctl(fd, SIOCZRLST, struct frentry **)
ioctl(fd, SIOCAUTHW, struct fr_info **)
ioctl(fd, SIOCAUTHR, struct fr_info **)
ioctl(fd, SIOCATHST, struct fr_authstat **)

The variations, SIOCADAFR vs. SIOCADIFR, allow operation on the two lists, active and inactive, respectively. All of these ioctl's are implemented as being routing ioctls and thus the same rules for the various routing ioctls and the file descriptor are employed, mainly being that the fd must be that of the device associated with the module (i.e., /dev/ipl).

The three groups of ioctls above perform adding rules to the end of the list (SIOCAD*), deletion of rules from any place in the list (SIOCRM*) and insertion of a rule into the list (SIOCIN*). The rule place into which it is inserted is stored in the "fr_hits" field, below.

```
typedef struct  frentry {
      struct  frentry *fr_next;
      u_short fr_group;       /* group to which this rule belongs */
      u_short fr_grhead;      /* group # which this rule starts */
      struct  frentry *fr_grp;
      int     fr_ref;         /* reference count - for grouping */
      void    *fr_ifa;
#if BSD >= 199306
      void    *fr_oifa;
#endif
      /*
       * These are only incremented when a packet  matches this rule and
       * it is the last match
       */
```

```
        U_QUAD_T        fr_hits;
        U_QUAD_T        fr_bytes;
        /*
         * Fields after this may not change whilst in the kernel.
         */
        struct  fr_ip   fr_ip;
        struct  fr_ip   fr_mip; /* mask structure */

        u_char  fr_tcpfm;       /* tcp flags mask */
        u_char  fr_tcpf;        /* tcp flags */

        u_short fr_icmpm;       /* data for ICMP packets (mask) */
        u_short fr_icmp;

        u_char  fr_scmp;        /* data for port comparisons */
        u_char  fr_dcmp;
        u_short fr_dport;
        u_short fr_sport;
        u_short fr_stop;        /* top port for <> and >< */
        u_short fr_dtop;        /* top port for <> and >< */
        u_32_t  fr_flags;       /* per-rule flags && options (see below) */
        u_short fr_skip;        /* # of rules to skip */
        u_short fr_loglevel;    /* syslog log facility + priority */
        int     (*fr_func) __P((int, ip_t *, fr_info_t *));
        char    fr_icode;       /* return ICMP code */
        char    fr_ifname[IFNAMSIZ];
#if BSD > 199306
        char    fr_oifname[IFNAMSIZ];
#endif
        struct  frdest  fr_tif; /* "to" interface */
        struct  frdest  fr_dif; /* duplicate packet interfaces */
} frentry_t;
```

When adding a new rule, all unused fields (in the filter rule) should be initialised to be zero. To insert a rule, at a particular position in the filter list, the number of the rule which it is to be inserted before must be put in the "fr_hits" field (the first rule is number 0).

Flags which are recognised in fr_flags:

```
    FR_BLOCK        0x000001    /* do not allow packet to pass */
    FR_PASS         0x000002    /* allow packet to pass */
    FR_OUTQUE       0x000004    /* outgoing packets */
    FR_INQUE        0x000008    /* ingoing packets */
    FR_LOG          0x000010    /* Log */
    FR_LOGB         0x000011    /* Log-fail */
    FR_LOGP         0x000012    /* Log-pass */
    FR_LOGBODY      0x000020    /* log the body of packets too */
    FR_LOGFIRST     0x000040    /* log only the first packet to match */
    FR_RETRST       0x000080    /* return a TCP RST packet if blocked */
    FR_RETICMP      0x000100    /* return an ICMP packet if blocked */
    FR_FAKEICMP     0x00180     /* Return ICMP unreachable with fake source */
    FR_NOMATCH      0x000200    /* no match occured */
    FR_ACCOUNT      0x000400    /* count packet bytes */
    FR_KEEPFRAG     0x000800    /* keep fragment information */
    FR_KEEPSTATE    0x001000    /* keep ‘connection’ state information */
```

```
FR_INACTIVE     0x002000
FR_QUICK        0x004000   /* match & stop processing list */
FR_FASTROUTE    0x008000   /* bypass normal routing */
FR_CALLNOW      0x010000   /* call another function (fr_func) if matches */
FR_DUP          0x020000   /* duplicate the packet */
FR_LOGORBLOCK   0x040000   /* block the packet if it can't be logged */
FR_NOTSRCIP     0x080000   /* not the src IP# */
FR_NOTDSTIP     0x100000   /* not the dst IP# */
FR_AUTH         0x200000   /* use authentication */
FR_PREAUTH      0x400000   /* require preauthentication */
```

Values for fr_scomp and fr_dcomp (source and destination port value comparisons) :

```
FR_NONE         0
FR_EQUAL        1
FR_NEQUAL       2
FR_LESST        3
FR_GREATERT     4
FR_LESSTE       5
FR_GREATERTE    6
FR_OUTRANGE     7
FR_INRANGE      8
```

The third ioctl, SIOCIPFFL, flushes either the input filter list, the output filter list or both and it returns the number of filters removed from the list(s). The values which it will take and recognise are FR_INQUE and FR_OUTQUE (see above). This ioctl is also implemented for **/dev/ipstate** and will flush all state tables entries if passed 0 or just all those which are not established if passed 1.

General Logging Flags

There are two flags which can be set to log packets independently of the rules used. These allow for packets which are either passed or blocked to be logged. To set (and clear)/get these flags, two ioctls are provided:

SIOCSETFF Takes an unsigned integer as the parameter. The flags are then set to those provided (clearing/setting all in one).

```
FF_LOGPASS      0x10000000
FF_LOGBLOCK     0x20000000
FF_LOGNOMATCH   0x40000000
FF_BLOCKNONIP   0x80000000    /* Solaris 2.x only */
```

SIOCGETFF Takes a pointer to an unsigned integer as the parameter. A copy of the flags currently in used is copied to user space.

Filter statistics

Statistics on the various operations performed by this package on packets is kept inside the kernel. These statistics apply to packets traversing through the kernel. To retrieve this structure, use this ioctl:

```
ioctl(fd, SIOCGETFS, struct friostat *)
```

```
struct  friostat    {
    struct  filterstats  f_st[2];
    struct  frentry      *f_fin[2];
    struct  frentry      *f_fout[2];
    struct  frentry      *f_acctin[2];
    struct  frentry      *f_acctout[2];
    struct  frentry      *f_auth;
```

```
        u_long  f_froute[2];
        int     f_active;       /* 1 or 0 - active rule set */
        int     f_defpass;      /* default pass - from fr_pass */
        int     f_running;      /* 1 if running, else 0 */
        int     f_logging;      /* 1 if enabled, else 0 */
        char    f_version[32];  /* version string */
};

struct     filterstats {
        u_long  fr_pass;        /* packets allowed */
        u_long  fr_block;       /* packets denied */
        u_long  fr_nom;         /* packets which don't match any rule */
        u_long  fr_ppkl;        /* packets allowed and logged */
        u_long  fr_bpkl;        /* packets denied and logged */
        u_long  fr_npkl;        /* packets unmatched and logged */
        u_long  fr_pkl;         /* packets logged */
        u_long  fr_skip;        /* packets to be logged but buffer full */
        u_long  fr_ret;         /* packets for which a return is sent */
        u_long  fr_acct;        /* packets for which counting was performed */
        u_long  fr_bnfr;        /* bad attempts to allocate fragment state */
        u_long  fr_nfr;         /* new fragment state kept */
        u_long  fr_cfr;         /* add new fragment state but complete pkt */
        u_long  fr_bads;        /* bad attempts to allocate packet state */
        u_long  fr_ads;         /* new packet state kept */
        u_long  fr_chit;        /* cached hit */
        u_long  fr_pull[2];     /* good and bad pullup attempts */
#if SOLARIS
        u_long  fr_notdata;     /* PROTO/PCPROTO that have no data */
        u_long  fr_nodata;      /* mblks that have no data */
        u_long  fr_bad;         /* bad IP packets to the filter */
        u_long  fr_notip;       /* packets passed through no on ip queue */
        u_long  fr_drop;        /* packets dropped - no info for them! */
#endif
};
```

If we wanted to retrieve all the statistics and reset the counters back to 0, then the ioctl() call would be made to SIOCFRZST rather than SIOCGETFS. In addition to the statistics above, each rule keeps a hit count, counting both number of packets and bytes. To reset these counters for a rule, load the various rule information into a frentry structure and call SIOCZRLST.

Swapping Active lists

IP Filter supports two lists of rules for filtering and accounting: an active list and an inactive list. This allows for large scale rule base changes to be put in place atomically with otherwise minimal interruption. Which of the two is active can be changed using the SIOCSWAPA ioctl. It is important to note that no passed argument is recognised and that the value returned is that of the list which is now inactive.

FILES

/dev/ipauth
/dev/ipl
/dev/ipnat
/dev/ipstate

SEE ALSO

ipl(4), ipnat(4), ipf(5), ipf(8), ipfstat(8)

BUGS

When a packet encapsulated by ipsec(4) tunnel comes in, ipf(4) looks at wire-format packet on inbound and outbound. ipf(4) will not look at decapsulated packets on inbound, nor packets prior to encapsulation on

outbound.

When tunneled packets arrive at the node and are handled by a tunneling pseudo interface like gif(4), the packets may go through ipf(4) twice, before and after decapsulation. In some cases it may be necessary to check, in the ipf(4) rules, if the inbound interface is a tunneling pseudo interface or not.

NAME

ipfilter − Introduction to IP packet filtering

DESCRIPTION

IP Filter is a TCP/IP packet filter, suitable for use in a firewall environment. To use, it can either be used as a loadable kernel module or incorporated into your UNIX kernel; use as a loadable kernel module where possible is highly recommended. Scripts are provided to install and patch system files, as required.

FEATURES

The IP packet filter can:

- explicitly deny/permit any packet from passing through
- distinguish between various interfaces
- filter by IP networks or hosts
- selectively filter any IP protocol
- selectively filter fragmented IP packets
- selectively filter packets with IP options
- send back an ICMP error/TCP reset for blocked packets
- keep packet state information for TCP, UDP and ICMP packet flows
- keep fragment state information for any IP packet, applying the same rule to all fragments.
- act as a Network Address Translator (NAT)
- use redirection to setup true transparent proxy connections
- provide packet header details to a user program for authentication
- in addition, supports temporary storage of pre-authenticated rules for passing packets through

Special provision is made for the three most common Internet protocols, TCP, UDP and ICMP. The IP Packet filter allows filtering of:

- Inverted host/net matchingTCP/UDP packets by port number or a port number range
- ICMP packets by type/code
- "established" TCP packets
- On any arbitrary combination of TCP flags
- "short" (fragmented) IP packets with incomplete headers can be filtered
- any of the 19 IP options or 8 registered IP security classes TOS (Type of Service) field in packets

To keep track of the performance of the IP packet filter, a logging device is used which supports logging of:

- the TCP/UDP/ICMP and IP packet headers
- the first 128 bytes of the packet (including headers)

A packet can be logged when:

- it is successfully passed through
- it is blocked from passing through
- it matches a rule setup to look for suspicious packets

IP Filter keeps its own set of statistics on:

- packets blocked
- packets (and bytes!) used for accounting
- packets passed packets logged

attempts to log which failed (buffer full)

and much more, for packets going both in and out.

Tools

The current implementation provides a small set of tools, which can easily be used and integrated with regular unix shells and tools. A brief description of the tools provided:

ipf(8) reads in a set of rules, from either stdin or a file, and adds them to the kernels current list (appending them). It can also be used to flush the current filter set or delete individual filter rules. The file format is described in ipf(5).

ipfs(8) is a utility to temporarily lock the IP Filter kernel tables (state tables and NAT mappings) and write them to disk. After that the system can be rebooted, and ipfs can be used to read these tables from disk and restore them into the kernel. This way the system can be rebooted without the connections being terminated.

ipfstat(8) interrogates the kernel for statistics on packet filtering, so far, and retrieves the list of filters in operation for inbound and outbound packets.

ipftest(1) reads in a filter rule file and then applies sample IP packets to the rule file. This allows for testing of filter list and examination of how a packet is passed along through it.

ipmon(8) reads buffered data from the logging device (default is /dev/ipl) for output to either:

screen (standard output)

file

syslog

ipsend(1) generates arbitary IP packets for ethernet connected machines.

ipresend(1) reads in a data file of saved IP packets (ie snoop/tcpdump/etherfind output) and sends it back across the network.

iptest(1) contains a set of test "programs" which send out a series of IP packets, aimed at testing the strength of the TCP/IP stack at which it is aimed at. WARNING: this may crash machine(s) targeted!

ipnat(8) reads in a set of rules, from either stdin or a file and adds them to the kernels current list of active NAT rules. NAT rules can also be deleted using ipnat. The format of the configuration file to be used with ipnat is described in ipnat(5).

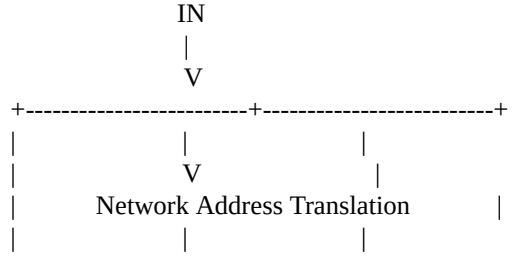
For use in your own programs (e.g. for writing of transparent application proxies), the programming interface and the associated ioctl's are documented in ipf(4).

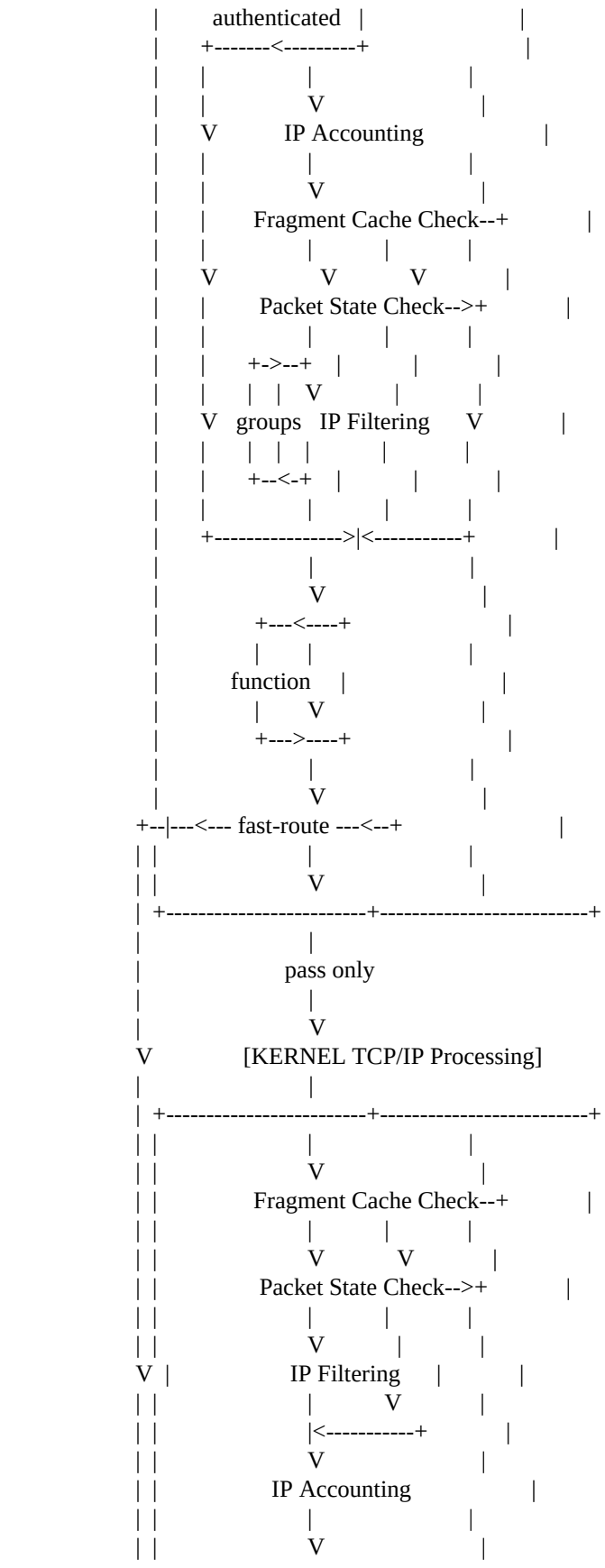
Documentation on ioctl's and the format of data saved to the logging character device is provided in ipl(4) so that you may develop your own applications to work with or in place of any of the above.

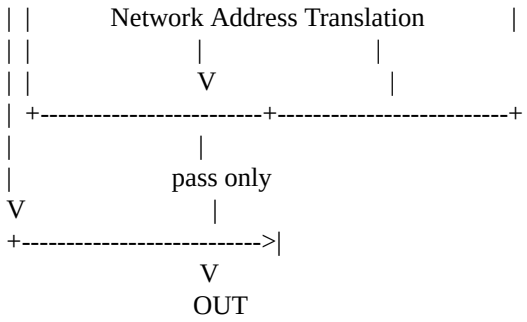
Similar, the interface to the NAT code is documented in ipnat(4).

PACKET PROCESSING FLOW

The following diagram illustrates the flow of TCP/IP packets through the various stages introduced by IP Filter.




authenticated
IP Accounting
Fragment Cache Check
Packet State Check
groups
IP Filtering
function
fast-route
pass only
[KERNEL TCP/IP Processing]
Fragment Cache Check
Packet State Check
IP Filtering
IP Accounting



MORE INFORMATION

More information (including pointers to the FAQ and the mailing list) can be obtained from the sofware's official homepage: www.ipfilter.org

SEE ALSO

ipf(4), ipf(5), ipf(8), ipfilter(5), ipfs(8), ipfstat(8), ipftest(1), ipl(4), ipmon(8), ipnat(8), ipnat(4),

NAME

ipkdb — IP-based kernel debugger

SYNOPSIS

```
options IPKDB
options IPKDBSECURE
options IPKDBKEY="\"IPKDB key for remote debugging\""
options IPKDB_NE_PCISLOT=x
```

DESCRIPTION

ipkdb is a kernel debugger that uses UDP and IP to communicate with a remote debugger (normally gdb(1)).

Since the debugger uses its own driver to talk to the networking hardware, the number of supported network interfaces is a lot less than what is supported by the kernel. At the time of this writing, there is only support for a NE2000 compatible card in a PCI slot. In order for **ipkdb** to find your card, you need to specify the slot the card is in via the `options IPKDB_NE_PCISLOT`.

To use **ipkdb**, you have to set up a DHCP server, from which **ipkdb** can get the IP address for the interface that is used for debugging.

To enter **ipkdb**, the remote debugger has to send the protocol start packet. E.g., gdb will do this on the command

```
target ipkdb debuggee IPKDB key for remote debugging
```

where `debuggee` is the name of the machine to debug (which must resolve to the IP address of the interface), and the rest of the line corresponds to the definition of `IPKDBKEY` (see below). For **ipkdb** to actually see this packet, the interface which is used for debugging has to be set up to actually receive packets, i.e. it has to be up and running.

To prevent messing around with a secured system, **ipkdb** normally also checks the security level at which the kernel is running. **ipkdb** will only work with security levels less than 1, unless the kernel is configured with `options IPKDBSECURE`.

In addition, the debugger is forcedly entered on a panic, as well as on initial startup, if you boot the kernel with the **-d** option (note that this however is machine dependent). On such a forced enter to **ipkdb** there is no need for the interface in question to already be up and running, since **ipkdb** only needs to send/receive packets via its own driver.

As some form of security against the occasional hacker, **ipkdb** uses the definition of `options IPKDBKEY` to compute a checksum on the data in every packet. The remote debugger has to send this checksum, based on the data it sends and the key, or **ipkdb** ignores the packet. This is also used in order to check against data errors on the connection.

SEE ALSO

gdb(1), ddb(4), ip(4), udp(4), init(8)

HISTORY

ipkdb appeared in NetBSD 1.3 for the first time. Its configuration and setup changed quite a bit in NetBSD 1.5.

BUGS

Since the kernel includes the definition of IPKDBKEY, anyone who can read the kernel can extract it and use it to enter **ipkdb**.

There is no support for `ip6`(4).

NAME

ipl − IP packet log device

DESCRIPTION

The **ipl** pseudo device's purpose is to provide an easy way to gather packet headers of packets you wish to log. If a packet header is to be logged, the entire header is logged (including any IP options − TCP/UDP options are not included when it calculates header size) or not at all. The packet contents are also logged after the header. If the log reader is busy or otherwise unable to read log records, up to IPLLOGSIZE (8192 is the default) bytes of data are stored.

Prepending every packet header logged is a structure containing information relevant to the packet following and why it was logged. The structure's format is as follows:

```
/*
 * Log structure.  Each packet header logged is prepended by one of these.
 * Following this in the log records read from the device will be an ipflog
 * structure which is then followed by any packet data.
 */
typedef struct iplog    {
      u_long  ipl_sec;
      u_long  ipl_usec;
      u_int   ipl_len;
      u_int   ipl_count;
      size_t  ipl_dsize;
      struct  iplog   *ipl_next;
} iplog_t;


typedef struct  ipflog  {
#if (defined(NetBSD) && (NetBSD <= 1991011) && (NetBSD >= 199603))
      u_char  fl_ifname[IFNAMSIZ];
#else
      u_int   fl_unit;
      u_char  fl_ifname[4];
#endif
      u_char  fl_plen;       /* extra data after hlen */
      u_char  fl_hlen;       /* length of IP headers saved */
      u_short fl_rule;       /* assume never more than 64k rules, total */
      u_32_t  fl_flags;
} ipflog_t;
```

When reading from the **ipl** device, it is necessary to call read(2) with a buffer big enough to hold at least 1 complete log record - reading of partial log records is not supported.

If the packet contents are more than 128 bytes when **log body** is used, then only 128 bytes of the packet contents are logged.

Although it is only possible to read from the **ipl** device, opening it for writing is required when using an ioctl which changes any kernel data.

The ioctls which are loaded with this device can be found under **ipf(4)**. The ioctls which are for use with logging and don't affect the filter are:

```
ioctl(fd, SIOCIPFFB, int *)
ioctl(fd, FIONREAD, int *)
```

The SIOCIPFFB ioctl flushes the log buffer and returns the number of bytes flushed. FIONREAD returns the number of bytes currently used for storing log data. If IPFILTER_LOG is not defined when compiling, SIOCIPFFB is not available and FIONREAD will return but not do anything.

There is currently no support for non-blocking IO with this device, meaning all read operations should be considered blocking in nature (if there is no data to read, it will sleep until some is made available).

SEE ALSO

ipf(4)

BUGS

Packet headers are dropped when the internal buffer (static size) fills.

FILES

/dev/ipl

NAME

ipmi — Intelligent Platform Management Interface driver

SYNOPSIS

```
ipmi0 at mainbus?
```

DESCRIPTION

The **ipmi** device driver supports motherboards implementing the Intelligent Platform Management Interface version 1.5 or 2.0, and exports sensors and the watchdog through the envsys(4) interface.

EVENTS

The **ipmi** driver is able to send events to powerd(8) when sensor's state has changed. Intrusion sensors will send a *critical* event when state is not ok. Power Supply sensors will send a *critical* event when the Power Supply unit is not installed and *warning-over* when the Power Supply unit is installed but not powered on. Fan, temperature and voltage sensors will send *critical-over* when the value is very critical, or *warning-over* if it's in a warning alert.

SEE ALSO

envsys(4), envstat(8), powerd(8), wdogctl(8)

HISTORY

The **ipmi** driver first appeared in OpenBSD 3.9 and was then ported to NetBSD 4.0.

AUTHORS

The **ipmi** driver was originally written by Jordan Hargrave and was ported to NetBSD by Manuel Bouyer ⟨bouyer@NetBSD.org⟩.

NAME

ipnat − Network Address Translation kernel interface

SYNOPSIS

```
#include <netinet/ip_compat.h>
#include <netinet/ip_fil.h>
#include <netinet/ip_proxy.h>
#include <netinet/ip_nat.h>
```

IOCTLS

To add and delete rules to the NAT list, two 'basic' ioctls are provided for use. The ioctl's are called as:

```
ioctl(fd, SIOCADNAT, struct ipnat **)
ioctl(fd, SIOCRMNAT, struct ipnat **)
ioctl(fd, SIOCGNATS, struct natstat **)
ioctl(fd, SIOCGNATL, struct natlookup **)
```

Unlike **ipf(4)**, there is only a single list supported by the kernel NAT interface. An inactive list which can be swapped to is not currently supported.

These ioctl's are implemented as being routing ioctls and thus the same rules for the various routing ioctls and the file descriptor are employed, mainly being that the fd must be that of the device associated with the module (i.e., /dev/ipl).

The structure used with the NAT interface is described below:

```
typedef struct  ipnat   {
      struct  ipnat   *in_next;
      void    *in_ifp;
      u_short in_flags;
      u_short in_pnext;
      u_short in_port[2];
      struct  in_addr in_in[2];
      struct  in_addr in_out[2];
      struct  in_addr in_nextip;
      int     in_space;
      int     in_redir; /* 0 if it's a mapping, 1 if it's a hard redir */
      char    in_ifname[IFNAMSIZ];
} ipnat_t;

#define in_pmin         in_port[0]      /* Also holds static redir port */
#define in_pmax         in_port[1]
#define in_nip          in_nextip.s_addr
#define in_inip         in_in[0].s_addr
#define in_inmsk        in_in[1].s_addr
#define in_outip        in_out[0].s_addr
#define in_outmsk       in_out[1].s_addr
```

Recognised values for in_redir:

```
#define NAT_MAP         0
#define NAT_REDIRECT    1
```

NAT statistics Statistics on the number of packets mapped, going in and out are kept, the number of times a new entry is added and deleted (through expiration) to the NAT table and the current usage level of the NAT table.

Pointers to the NAT table inside the kernel, as well as to the top of the internal NAT lists constructed with the **SIOCADNAT** ioctls. The table itself is a hash table of size NAT_SIZE (default size is 367).

To retrieve the statistics, the **SIOCGNATS** ioctl must be used, with the appropriate structure passed by reference, as follows:

```
ioctl(fd, SIOCGNATS, struct natstat *)
```

```
typedef struct  natstat {
      u_long  ns_mapped[2];
      u_long  ns_added;
      u_long  ns_expire;
      u_long  ns_inuse;
      nat_t   ***ns_table;
      ipnat_t *ns_list;
} natstat_t;
```

BUGS

It would be nice if there were more flexibility when adding and deleting filter rules.

FILES

/dev/ipnat

SEE ALSO

ipf(4), ipnat(5), ipf(8), ipnat(8), ipfstat(8)

NAME

ippp — ISDN synchronous PPP network driver

SYNOPSIS

pseudo-device ippp *count*

DESCRIPTION

The **ippp** driver interfaces the IP subsystem of the operating system with the ISDN layer so that a transport of IP packets over an ISDN link is possible.

For configuration of the **ippp** driver, either the `ipppctl`(8) utility is used or it is configured via `isdnd`(8) and its associated `isdnd.rc`(5) file.

In case an IP packet for a remote side arrives in the driver and no connection is established yet, the driver communicates with the `isdnd`(8) daemon to establish a connection.

The driver has support for interfacing to the `bpf`(4) subsystem for using `tcpdump`(8) with the **ippp** interfaces.

The `ipppctl`(8) utility is used to configure all aspects of PPP required to connect to a remote site.

LINK0 and LINK1

The *link0* and *link1* flags given as parameters to `ifconfig`(8) have the following meaning for the **ippp** devices:

`link0` Wait passively for connection. The administrative *Open* event to the Link Control Protocol (LCP) layer will be delayed until after the lower layers signal an *Up* event (rise of "carrier"). This can be used by lower layers to support a dial-in connection where the physical layer isn't available immediately at startup, but only after some external event arrives. Receipt of a *Down* event from the lower layer will not take the interface completely down in this case.

`link1` Dial-on-demand mode. The administrative *Open* event to the LCP layer will be delayed until either an outbound network packet arrives, or until the lower layer signals an *Up* event, indicating an inbound connection. As with passive mode, receipt of a *Down* event (loss of carrier) will not automatically take the interface down, thus it remains available for further connections.

The *link0* flag is set to *off* by default, the *link1* flag to *on*.

SEE ALSO

`bpf`(4), `isdnd.rc`(5), `ipppctl`(8), `isdnd`(8), `tcpdump`(8)

AUTHORS

The **ippp** device driver was written by Joerg Wunsch ⟨joerg@freebsd.org⟩ and then added to ISDN4BSD by Gary Jennejohn ⟨gary@freebsd.org⟩.

This man page was written by Hellmuth Michaelis ⟨hm@kts.org⟩.

NAME

ipsec — IP security protocol

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <netinet6/ipsec.h>

options IPSEC
options IPSEC_ESP
options IPSEC_NAT_T
options IPSEC_DEBUG
```

DESCRIPTION

ipsec is a security protocol in Internet Protocol layer. **ipsec** is defined for both IPv4 and IPv6 (inet(4) and inet6(4)). **ipsec** consists of two sub-protocols, namely ESP (encapsulated security payload) and AH (authentication header). ESP protects IP payload from wire-tapping by encrypting it by secret key cryptography algorithms. AH guarantees integrity of IP packet and protects it from intermediate alteration or impersonation, by attaching cryptographic checksum computed by one-way hash functions. **ipsec** has two operation modes: transport mode and tunnel mode. Transport mode is for protecting peer-to-peer communication between end nodes. Tunnel mode includes IP-in-IP encapsulation operation and is designed for security gateways, like VPN configurations.

The following kernel options are available:

options IPSEC
Includes support for the IPsec protocol. *IPSEC* will enable secret key management part, policy management part, AH and IPComp. Kernel binary will not be subject to export control in most of countries, even if compiled with *IPSEC*. For example, it should be okay to export it from within the United States to the outside. *INET6* and *IPSEC* are orthogonal so you can get IPv4-only kernel with IPsec support, IPv4/v6 dual support kernel without IPsec, and so forth. This option requires *INET* at this moment, but it should not.

options IPSEC_DEBUG
Enables debugging code in IPsec stack. This option assumes *IPSEC*.

options IPSEC_ESP
Includes support for IPsec ESP protocol. *IPSEC_ESP* will enable source code that is subject to export control in some countries (including the United States), and compiled kernel binary will be subject to certain restriction. This option assumes *IPSEC*.

options IPSEC_NAT_T
Includes support for IPsec Network Address Translator traversal (NAT-T), as described in RFCs 3947 and 3948. This feature might be patent-encumbered in some countries. This option assumes *IPSEC* and *IPSEC_ESP*.

Kernel interface

ipsec is controlled by key management engine and policy engine, in the operating system kernel.

Key management engine can be accessed from the userland by using PF_KEY sockets. The PF_KEY socket API is defined in RFC2367.

Policy engine can be controlled by extended part of PF_KEY API, setsockopt(2) operations, and sysctl(3) interface. The kernel implements extended version of PF_KEY interface, and allows you to define IPsec policy like per-packet filters. setsockopt(2) interface is used to define per-socket behavior, and sysctl(3) interface is used to define host-wide default behavior.

The kernel code does not implement dynamic encryption key exchange protocol like IKE (Internet Key Exchange). That should be implemented as userland programs (usually as daemons), by using the above described APIs.

Policy management

The kernel implements experimental policy management code. You can manage the IPsec policy in two ways. One is to configure per-socket policy using `setsockopt`(2). The other is to configure kernel packet filter-based policy using `PF_KEY` interface, via `setkey`(8). In both cases, IPsec policy must be specified with syntax described in `ipsec_set_policy`(3).

With `setsockopt`(2), you can define IPsec policy in per-socket basis. You can enforce particular IPsec policy onto packets that go through particular socket.

With `setkey`(8) you can define IPsec policy against packets, using sort of packet filtering rule. Refer to `setkey`(8) on how to use it.

In the latter case, "`default`" policy is allowed for use with `setkey`(8). By configuring policy to `default`, you can refer system-wide `sysctl`(8) variable for default settings. The following variables are available. `1` means "`use`", and `2` means "`require`" in the syntax.

Name	Type	Changeable
net.inet.ipsec.esp_trans_deflev	integer	yes
net.inet.ipsec.esp_net_deflev	integer	yes
net.inet.ipsec.ah_trans_deflev	integer	yes
net.inet.ipsec.ah_net_deflev	integer	yes
net.inet6.ipsec6.esp_trans_deflev	integer	yes
net.inet6.ipsec6.esp_net_deflev	integer	yes
net.inet6.ipsec6.ah_trans_deflev	integer	yes
net.inet6.ipsec6.ah_net_deflev	integer	yes

If kernel finds no matching policy system wide default value is applied. System wide default is specified by the following `sysctl`(8) variables. `0` means "`discard`" which asks the kernel to drop the packet. `1` means "`none`".

Name	Type	Changeable
net.inet.ipsec.def_policy	integer	yes
net.inet6.ipsec6.def_policy	integer	yes

Miscellaneous sysctl variables

The following variables are accessible via `sysctl`(8), for tweaking kernel IPsec behavior:

Name	Type	Changeable
net.inet.ipsec.ah_cleartos	integer	yes
net.inet.ipsec.ah_offsetmask	integer	yes
net.inet.ipsec.dfbit	integer	yes
net.inet.ipsec.ecn	integer	yes
net.inet.ipsec.debug	integer	yes
net.inet6.ipsec6.ecn	integer	yes
net.inet6.ipsec6.debug	integer	yes

The variables are interpreted as follows:

`ipsec.ah_cleartos`

If set to non-zero, the kernel clears type-of-service field in the IPv4 header during AH authentication data computation. The variable is for tweaking AH behavior to interoperate with devices that implement RFC1826 AH. It should be set to non-zero (clear the type-of-service field) for

RFC2402 conformance.

`ipsec.ah_offsetmask`
: During AH authentication data computation, the kernel will include 16bit fragment offset field (including flag bits) in IPv4 header, after computing logical AND with the variable. The variable is for tweaking AH behavior to interoperate with devices that implement RFC1826 AH. It should be set to zero (clear the fragment offset field during computation) for RFC2402 conformance.

`ipsec.dfbit`
: The variable configures the kernel behavior on IPv4 IPsec tunnel encapsulation. If set to 0, DF bit on the outer IPv4 header will be cleared. 1 means that the outer DF bit is set regardless from the inner DF bit. 2 means that the DF bit is copied from the inner header to the outer. The variable is supplied to conform to RFC2401 chapter 6.1.

`ipsec.ecn`
: If set to non-zero, IPv4 IPsec tunnel encapsulation/decapsulation behavior will be friendly to ECN (explicit congestion notification), as documented in `draft-ietf-ipsec-ecn-02.txt`. `gif`(4) talks more about the behavior.

`ipsec.debug`
: If set to non-zero, debug messages will be generated via `syslog`(3).

Variables under `net.inet6.ipsec6` tree has similar meaning as the `net.inet.ipsec` counterpart.

PROTOCOLS

The **`ipsec`** protocol works like plug-in to `inet`(4) and `inet6`(4) protocols. Therefore, **`ipsec`** supports most of the protocols defined upon those IP-layer protocols. Some of the protocols, like `icmp`(4) or `icmp6`(4), may behave differently with **`ipsec`**. This is because **`ipsec`** can prevent `icmp`(4) or `icmp6`(4) routines from looking into IP payload.

SEE ALSO

`ioctl`(2), `socket`(2), `ipsec_set_policy`(3), `fast_ipsec`(4), `icmp6`(4), `intro`(4), `ip6`(4), `racoon`(8), `setkey`(8), `sysctl`(8)

STANDARDS

Daniel L. McDonald, Craig Metz, and Bao G. Phan, *PF_KEY Key Management API, Version 2*, RFC, 2367.

HISTORY

The implementation described herein appeared in WIDE/KAME IPv6/IPsec stack.

BUGS

The IPsec support is subject to change as the IPsec protocols develop.

There is no single standard for policy engine API, so the policy engine API described herein is just for KAME implementation.

AH and tunnel mode encapsulation may not work as you might expect. If you configure inbound "require" policy against AH tunnel or any IPsec encapsulating policy with AH (like "`esp/tunnel/A-B/use ah/transport/A-B/require`"), tunneled packets will be rejected. This is because we enforce policy check on inner packet on reception, and AH authenticates encapsulating (outer) packet, not the encapsulated (inner) packet (so for the receiving kernel there's no sign of authenticity). The issue will be solved when we revamp our policy engine to keep all the packet decapsulation history.

Under certain condition, truncated result may be raised from the kernel against `SADB_DUMP` and `SADB_SPDDUMP` operation on `PF_KEY` socket. This occurs if there are too many database entries in the kernel and socket buffer for the `PF_KEY` socket is insufficient. If you manipulate many IPsec key/policy

database entries, increase the size of socket buffer or use `sysctl`(8) interface.

NAME

ipw — Intel PRO/Wireless 2100 IEEE 802.11 driver

SYNOPSIS

ipw* **at pci? dev ? function ?**

DESCRIPTION

The **ipw** driver provides support for the Intel PRO/Wireless 2100 MiniPCI network adapter.

By default, the **ipw** driver configures the adapter for BSS operation (aka infrastructure mode). This mode requires the use of an access point.

For more information on configuring this device, see `ifconfig`(8).

EXAMPLES

Join an existing BSS network (i.e.: connect to an access point):

```
ifconfig ipw0 inet 192.168.0.20 netmask 0xffffff00
```

Join a specific BSS network with network name "`my_net`":

```
ifconfig ipw0 inet 192.168.0.20 netmask 0xffffff00 nwid my_net
```

Join a specific BSS network with 64 bits WEP encryption:

```
ifconfig ipw0 inet 192.168.0.20 netmask 0xffffff00 nwid my_net \
        nwkey 0x1234567890
```

Join a specific BSS network with 128bits WEP encryption:

```
ifconfig ipw0 inet 192.168.0.20 netmask 0xffffff00 nwid my_net \
        nwkey 0x01020304050607080910111213
```

DIAGNOSTICS

ipw%d: device timeout The driver will reset the hardware. This should not happen.

SEE ALSO

`an`(4), `awi`(4), `pci`(4), `wi`(4), `ifconfig`(8), `ipwctl`(8), `pkgsrc/sysutils/ipw-firmware`

The IPW Web Page, http://damien.bergamini.free.fr/ipw/.

AUTHORS

The **ipw** driver and this man page were written by Damien Bergamini ⟨damien.bergamini@free.fr⟩.

NAME

irframe — IrDA frame level driver

SYNOPSIS

```
irframe* at oboe?
irframe* at uirda?
irframe* at ustir?
pseudo-device irframetty

#include <dev/irdaio.h>
```

DESCRIPTION

The **irframe** driver provides support for IrDA frame level transmission. It does not contain the IrDA protocol stack per se, but the stack can be built on top of the **irframe** driver.

Access to frames is via the `read`(2) and `write`(2) system calls. Each write constitutes one frame, and each read yields one frame. The `poll`(2) system call can be used to check for availability of frames. There are also a number of `ioctl`(2) calls to manipulate the device:

`IRDA_RESET_PARAMS`
: Reset the parameters set by `IRDA_SET_PARAMS`.

`IRDA_SET_PARAMS` *`(struct irda_params)`*
: Set the speed, extra beginning of frame bytes, and maximum frame size.

`IRDA_GET_SPEEDMASK` *`(int)`*
: Get the set of allowable speeds.

`IRDA_GET_TURNAROUNDMASK` *`(int)`*
: Get the set of allowable turn around times.

SEE ALSO

`cir`(4), `irframetty`(4), `oboe`(4), `uirda`(4), `ustir`(4)
comms/birda package

HISTORY

The **irframe** driver appeared in NetBSD 1.6.

NAME

irframetty — IrDA frame over serial line driver

SYNOPSIS

```
pseudo-device irframetty

#include <dev/irdaio.h>
```

DESCRIPTION

The **irframetty** driver provides a tty(4) line discipline to send and receive IrDA frames over a serial line via an IrDA dongle.

Access to the frames is via the irframe(4) driver.

Different dongles require different handling. The connected dongle type can be set with ioctl(2) calls:

IRFRAMETTY_SET_DONGLE *(int)*
: Set the dongle type. See the include file for possible dongles.

IRFRAMETTY_GET_DONGLE *(int)*
: Get the dongle type.

IRFRAMETTY_GET_DEVICE *(int)*
: Get the number of the irframe(4) device that must be used to access the frames.

SEE ALSO

irframe(4), irdaattach(8)

HISTORY

The **irframetty** driver appeared in NetBSD 1.6.

NAME

irip — Raw IP over ISDN network driver

SYNOPSIS

pseudo-device irip *count*

DESCRIPTION

The **irip** driver interfaces the IP subsystem of the operating system with the ISDN layer so that transport of IP packets over an ISDN link is possible.

The driver just packs IP packets without anything appended or prepended into raw HDLC packets on the B channel and transfers them to a remote site. IP packets received from the remote site are queued into the local IP protocol stack.

The format of the resulting packet on the B channel is:

```
(HDLC opening flag) (IP-packet) (CRC) (HDLC closing flag)
```

In the case where an IP packet for a remote site arrives in the driver and no connection has been established yet, the driver communicates with the `isdnd`(8) daemon to establish a connection.

The driver has support for interfacing to the `bpf`(4) subsystem for using `tcpdump`(8) with the **irip** interfaces.

The driver optionally (when compiled with the IRIP_VJ option) provides Van Jacobson header compression, under control of the link0 and link1 options to `ifconfig`(8):

link0	Apply VJ compression to outgoing packets on this interface, and assume that incoming packets require decompression.
link1	Check incoming packets for Van Jacobson compression; if they appear to be compressed, automatically set link0.

The default values are *on* for *link1* and *off* for *link0*.

SEE ALSO

`bpf`(4), `isdnd.rc`(5), `isdnd`(8), `tcpdump`(8)

AUTHORS

The **irip** device driver and this man page were written by Hellmuth Michaelis ⟨hm@kts.org⟩.

NAME

irongate — API UP1000 AMD751 Core Logic + AGP Chipset

SYNOPSIS

```
irongate* at mainbus?
pci* at irongate?
```

DESCRIPTION

The **irongate** driver provides support for the AMD751 Core Logic + AGP Chipset found on Alpha Processor, Inc.'s UP1000 systems.

SEE ALSO

intro(4), mainbus(4), pci(4)

NAME

isa — introduction to machine-independent ISA bus support and drivers

SYNOPSIS

Attachments are machine-dependent and depend on the bus topology and ISA bus interface of your system. See intro(4) for your system for details.

DESCRIPTION

NetBSD includes a machine-independent ISA bus subsystem and several machine-independent ISA device drivers.

Your system may support additional ISA devices. Drivers for ISA devices not listed here are machine-dependent. Consult your system's intro(4) for additional information.

SUPPORTED DEVICES

NetBSD includes machine-independent ISA drivers, sorted by device type and driver name:

SCSI interfaces

adv	Advansys SCSI interfaces.
aha	Adaptec AHA-154x family (154xA, 154xB, 154xC, and 154xCF) and the BusLogic BT54x SCSI interfaces.
ahc	Adaptec 29xx, 39xx, and other AIC-7xxx-based SCSI interfaces.
aic	Adaptec AIC-6260 and Adaptec AIC-6360 based SCSI interfaces, including the Adaptec 152x, SoundBlaster SCSI interfaces, and a variety of compatibles.
bha	BusLogic BT-445 SCSI interfaces.
dpt	DPT SmartCache/SmartRAID III and IV SCSI interfaces.
esp	NCR 53C9x, Emulex ESP406, and Qlogic FAS408 SCSI interfaces.
nca	NCR-5380/NCR-53C400
sea	Seagate/Future Domain SCSI cards. ST01/02, Future Domain TMC-885, and Future Domain TMC-950.
uha	Ultrastor 14f SCSI interfaces.
wds	WD-7000 family of bus-mastering SCSI interfaces.

Disk and tape controllers

mcd	Mitsumi CD-ROM drives.
wdc	Standard Western Digital type hard drive controllers: MFM, RLL, ESDI, and IDE/ATAPI.
wt	Wangtek and compatible QIC-02 and QIC-36 tape drives.

Serial and parallel interfaces

ast	Multi-port serial communications card first made by AST.
boca	Boca BB100[48] and BB2016 multiplexing serial communications cards.
com	NS8250-, NS16450-, and NS16550-based serial ports.

cy	Cyclades Cyclom-4Y, -8Y, and -16Y asynchronous serial communications cards.
ioat	BOCA Research IOAT66 serial interfaces.
lpt	Standard ISA parallel port interface.
rtfps	IBM RT four-port serial interfaces.
tcom	Byte Runner Technologies TC-400 and TC-800 series serial interfaces.

Network interfaces

ai	AT&T StarLan Ethernet interfaces.
ate	Allied Telesis AT1700 series and RE2000 series Ethernet interfaces.
cs	Cirrus Logic Crystal CS8900 Ethernet interfaces.
ec	3Com EtherLink II (3c503) Ethernet interfaces.
ef	3Com EtherLink II (3c507) Ethernet interfaces.
eg	3Com EtherLink Plus (3c505) Ethernet interfaces.
el	3Com EtherLink (3c501) Ethernet interfaces.
ep	3Com EtherLink III (3c509) Ethernet interfaces.
fmv	Fujitsu FMV-181 and FMV-182 interfaces.
ix	Intel EtherExpress/16 Ethernet interfaces.
iy	Intel i82595-based Ethernet interfaces, including the EtherExpress Pro/10.
lc	DEC EtherWORKS III Ethernet interfaces (DE203, DE204, and DE205).
le	Ethernet interfaces based on the AMD LANCE chip, including BICC Isolan, Novell NE2100, Digital DEPCA, and PCnet-ISA.
ne	Novel NE2000 and compatible Ethernet interfaces.
ntwoc	SDL Communications Riscom/N2 synchronous serial interfaces.
sm	SMC91C9x-based Ethernet interfaces.
tr	TROPIC based token ring interfaces.
we	Western Digital/SMC 80x3, SMC Elite Ultra, and SMC EtherEZ Ethernet interfaces.

Sound cards and MIDI interfaces

aria	Sierra’s Aria based sound cards.
cms	Creative Music System.
ess	ESS Technology AudioDrive 1788-, 1888-, 1887-, and 888-based sound cards.
gus	Gravis Ultrasound sound cards.
mpu	Roland MPU401 (and compatible) MIDI UARTs.
opl	Yamaha OPL2 and OPL3 FM MIDI synthesizers.
pas	ProAudio Spectrum sound cards.
sb	SoundBlaster, SoundBlaster 16, and SoundBlaster Pro sound cards.

wss	Windows Sound System-compatible sound cards based on the AD1848 and compatible chips.

Miscellaneous devices

az	Aztech/PackardBell radio card.
ega	EGA graphics boards.
lm	National Semiconductor LM78, LM79 and compatible hardware monitors.
pcdisplay	PC display adapters.
pcic	PCI PCMCIA controllers, including the Cirrus Logic GD6729.
pckbc	PC keyboard controllers.
pcppi	PC control and timer ports.
pms	PS/2 auxiliary port mice (including wheel mice).
rt	AIMS Lab Radiotrack FM radio.
rtii	AIMS Lab Radiotrack II FM radio.
sf2r	SoundForte RadioLink SF16-FMR2 FM radio.
tcic	Databook DB86082, DB86084, DB86184, and DB86072 PCMCIA controllers.
vga	VGA graphics boards.

Note that some ISA devices also have newer ISA Plug-and-Play variants. These are listed in `isapnp`(4). Some i386 platforms use `pnpbios`(4) to attach ISA devices.

DIAGNOSTICS

Stray interrupt on IRQ 7 It means the interrupt controller reported an unmasked interrupt on IRQ 7, but no driver attached to that IRQ ‘claimed’ it.

There are two reasons this can happen:

- In anything other than i386, it would almost always mean that there is a driver attached to the IRQ, but it is the wrong driver.
- On i386, there is the more obscure issue of ‘default IRQ7’s. That is, when a device asserts an IRQ, but the IRQ is deasserted after the PIC latches the interrupt and before the CPU acknowledges it, the PIC just flat out lies about which IRQ it was. It is usually due to a suboptimally coded driver.

SEE ALSO

`adv`(4), `aha`(4), `ahc`(4), `ai`(4), `aic`(4), `aria`(4), `ast`(4), `ate`(4), `az`(4), `bha`(4), `boca`(4), `cms`(4), `com`(4), `cs`(4), `cy`(4), `dpt`(4), `ec`(4), `ef`(4), `eg`(4), `el`(4), `ep`(4), `esp`(4), `ess`(4), `fmv`(4), `gus`(4), `intro`(4), `ioat`(4), `isapnp`(4), `ix`(4), `iy`(4), `joy`(4), `lc`(4), `le`(4), `lm`(4), `lpt`(4), `mcd`(4), `mpu`(4), `nca`(4), `ne`(4), `ntwoc`(4), `opl`(4), `pas`(4), `pcdisplay`(4), `pcic`(4), `pckbc`(4), `pcppi`(4), `pms`(4), `pnpbios`(4), `rt`(4), `rtfps`(4), `rtii`(4), `sb`(4), `sea`(4), `sf2r`(4), `sm`(4), `tcic`(4), `tcom`(4), `tr`(4), `uha`(4), `vga`(4), `wd`(4), `wdc`(4), `wds`(4), `we`(4), `wss`(4), `wt`(4)

HISTORY

The machine-independent ISA subsystem appeared in NetBSD 1.2.

NAME

isapnp — introduction to ISA Plug-and-Play support

SYNOPSIS

```
isapnp0 at isa?
```

An
isapnp bus can be configured for each supported ISA bus.

DESCRIPTION

NetBSD provides machine-independent bus support and drivers for ISA Plug-and-Play (isapnp) autoconfiguration of PnP-compatible devices on an ISA bus.

SUPPORTED DEVICES

NetBSD includes machine-independent ISAPNP drivers, sorted by function and driver name:

SCSI interfaces

aha	Adaptec AHA-154[02] SCSI interfaces.
aic	Adaptec AHA-1520B SCSI interfaces.

Disk controllers

wdc	Standard IDE and ATAPI drive controller.

Serial and parallel interfaces

com	8250/16450/16550-compatible ISA PnP serial cards and internal modems.

Network interfaces

an	Aironet 4500/4800 and Cisco 340 series 802.11 interfaces.
ep	3Com 3c509B EtherLink III Ethernet interface.
le	PCnet-PnP Ethernet interfaces based on the successor to the AMD LANCE chip.
ne	NE2000-compatible Ethernet interfaces.
tr	TROPIC based token ring interfaces.

Sound cards

ess	ESS Technology derived PnP sound cards and devices.
guspnp	Gravis Ultrasound PnP sound cards.
sb	SoundBlaster, SoundBlaster 16, and SoundBlaster Pro sound cards.
wss	Windows Sound System compatible cards, e.g., most of the cards with Crystal Semiconductor chips.
ym	Yamaha OPL3-SAx sound cards.

Miscellaneous devices

pcic	PCI PCMCIA controllers, including the Cirrus Logic GD6729.

ISA Plug-and-Play devices also have alternate ISA drivers with static ISA IO address configuration. These are listed in isa(4). The **isapnp** bus ignores devices that have already been found and configured as isa(4) devices. The **isapnp** bus is only effective on machines which lack a PnP BIOS, or on which the PnP BIOS has been disabled. The manual pages for each individual **isapnp** driver also list the supported

front-ends for other buses.

SEE ALSO

aha(4), aic(4), an(4), com(4), ep(4), ess(4), guspnp(4), intro(4), isa(4), le(4), ne(4), pcic(4), sb(4), tr(4), wdc(4), wss(4), ym(4)

HISTORY

The **isapnp** driver appeared in NetBSD 1.3.

NAME

isdn — ISDN kernel to userland master device

SYNOPSIS

```
pseudo-device isdn
```

DESCRIPTION

The **isdn** device driver is used by the isdnd(8) daemon to exchange messages with the ISDN kernel part for the purpose of call establishment, control and disconnection and to access various control and status informations.

The messages and message parameters are documented in the include file /usr/include/netisdn/i4b_ioctl.h.

The available ioctl's are:

I4B_CDID_REQ
: Request a unique Call Description IDentifier (cdid) which identifies uniquely a single interaction of the local D channel with the exchange.

I4B_CONNECT_REQ
: Actively request a call setup to a remote ISDN subscriber.

I4B_CONNECT_RESP
: Respond to an incoming call, either accept, reject or ignore it.

I4B_DISCONNECT_REQ
: Actively terminate a connection.

I4B_CTRL_INFO_REQ
: Request information about an installed ISDN controller card.

I4B_DIALOUT_RESP
: Give information about call setup to driver who requested dialing out.

I4B_TIMEOUT_UPD
: Update the kernels timeout value(s) in case of dynamically calculated shorthold mode timing changes.

I4B_UPDOWN_IND
: Inform the kernel userland drivers about interface soft up/down status changes.

I4B_CTRL_DOWNLOAD
: Download firmware to active card(s).

I4B_ACTIVE_DIAGNOSTIC
: Return diagnostic information from active cards.

Status and event messages available from the kernel are:

MSG_CONNECT_IND
: An incoming call from a remote ISDN user is indicated.

MSG_CONNECT_ACTIVE_IND
: After an incoming call has been accepted locally or an outgoing call has been accepted by a remote, the exchange signaled an active connection and the corresponding B-channel is switched through.

MSG_DISCONNECT_IND
: A call was terminated.

MSG_DIALOUT_IND
: A userland interface driver requests the daemon to dial out (typically a network interface when a packet arrives in its send queue).

MSG_IDLE_TIMEOUT_IND
: A call was terminated by the isdn4bsd kernel driver because a B-channel idle timeout occurred.

MSG_ACCT_IND
: Accounting information from a network driver.

MSG_CHARGING_IND
: Charging information from the kernel.

SEE ALSO

`isdnd`(8)

AUTHORS

The **`isdn`** device driver and this man page were written by Hellmuth Michaelis ⟨hm@kts.org⟩.

NAME

isdnbchan — ISDN Raw B-Channel access driver

SYNOPSIS

pseudo-device isdnbchan *count*

DESCRIPTION

The **isdnbchan** driver provides an interface to the raw untranslated B-channel.

SEE ALSO

isdnd.rc(5), isdnd(8)

AUTHORS

The **isdnbchan** device driver and this man page were written by Hellmuth Michaelis ⟨hm@kts.org⟩.

NAME

isdncapi — CAPI driver for isdn4bsd

DESCRIPTION

isdncapi is a CAPI driver for the *isdn4bsd* package. It sits between layer 4 of isdn4bsd and a driver for an active ISDN card; currently only the iavc(4) driver for the AVM B1 and T1 family of active cards is supported.

SEE ALSO

iavc(4)

STANDARDS

CAPI 2.0 (http://www.capi.org/)

AUTHORS

The **isdncapi** device driver was written by Juha-Matti Liukkonen ⟨jml@cubical.fi⟩ (Cubical Solutions Ltd, Finland) for FreeBSD and ported to NetBSD by Antti Kantee ⟨pooka@cubical.fi⟩. This manpage was written by Hellmuth Michaelis ⟨hm@FreeBSD.org⟩.

NAME

isdnctl — control device for the ISDN kernel part

SYNOPSIS

```
pseudo-device isdnctl
```

DESCRIPTION

isdnctl is used by the isdndebug(8) utility to get and set the current debugging level and other information of the kernel ISDN handling layers.

SEE ALSO

isdndebug(8)

AUTHORS

The **isdnctl** device driver and this man page were written by Hellmuth Michaelis ⟨hm@kts.org⟩.

NAME

isdntel — ISDN B-channel telephony interface driver

SYNOPSIS

pseudo-device isdntel *count*

DESCRIPTION

The **isdntel** driver provides an interface to the B-channel for telephony applications and is currently used by the isdnd(8) for answering machine support. The driver is part of the isdn4bsd package.

The lower six bits of the driver's minor number are used to specify a unit number, whereas the upper two bits specify a functionality.

Functionality zero is the usual telephony data stream i/o driver.

Functionality one is used to enable commands to dial out and hang up and receive responses about the state of the dial out progress and status. This commands may change in the future, for details see the file /usr/include/netisdn/i4b_tel_ioctl.h and the isdntel(8) utility.

The telephony data stream comes out of the line in a bit-reversed format, so the **isdntel** driver does the bit-reversion process in any case.

Additionally, the user can specify to do A-law to mu-law, mu-law to A-law or no conversion at all in the **isdntel** driver by using the isdntelctl(8) utility.

The driver is able to process several ioctl's:

I4B_TEL_GETAUDIOFMT
: get currently used audio format conversion.

I4B_TEL_SETAUDIOFMT
: set currently used audio format conversion.

I4B_TEL_EMPTYINPUTQUEUE
: clear the input queue.

For the I4B_TEL_GETAUDIOFMT and I4B_TEL_SETAUDIOFMT, the following parameters are available:

CVT_NONE
: do no A-law/mu-law audio format conversion. The conversion path looks like this:

  USER <--> bitreversing <--> ISDN-line

CVT_ALAW2ULAW
: set audio format conversion to do an audio conversion from A-law (on the ISDN line) to mu-law (in the userland). The read(2) conversion path looks like this:

  USER <-- mu-law/A-law <-- bitreversing <-- ISDN-line

  and the write(2) conversion path looks like this:

  USER --> mu-law/A-law --> bitreversing --> ISDN-line

CVT_ULAW2ALAW
: set audio format conversion to do an audio conversion from mu-law (on the ISDN line) to A-law (in the userland). The read(2) conversion path looks like this:

  USER <-- A-law/mu-law <-- bitreversing <-- ISDN-line

  and the write(2) conversion path looks like this:

USER --> A-law/mu-law --> bitreversing --> ISDN-line

SEE ALSO

`isdnd.rc`(5), `isdnd`(8), `isdntel`(8), `isdntelctl`(8)

STANDARDS

A-law and mu-law are specified in ITU Recommendation G.711.

AUTHORS

The **`isdntel`** device driver and this man page were written by Hellmuth Michaelis ⟨hm@kts.org⟩.

NAME

isdntrc — ISDN interface driver for D and B channel tracing

SYNOPSIS

pseudo-device isdntrc *count*

DESCRIPTION

The **isdntrc** driver is used to add a header to the data got from the D and/or B channel and queues it to be read and further processed by the isdntrace(8) utility. Currently, *count* should be the number of B channels (twice the number of cards).

SEE ALSO

isdnd(8), isdntrace(8)

AUTHORS

The **isdntrc** device driver and this man page were written by Hellmuth Michaelis ⟨hm@kts.org⟩.

NAME

isic — isdn4bsd Siemens ISDN Chipset device driver

SYNOPSIS

On the ISA bus:

Teles S0/8 or Niccy 1008 card:

```
options ISICISA_TEL_S0_8
isic0 at isa? iomem 0xd0000 irq 5
```

Teles S0/16 or Creatix ISDN-S0 or Niccy 1016 card:

```
options ISICISA_TEL_S0_16
isic0 at isa? port 0xd80 iomem 0xd0000 irq 5
```

Teles S0/16.3 card:

```
options ISICISA_TEL_S0_16_3
isic0 at isa? port 0xd80 irq 5
```

AVM A1 or AVM Fritz card:

```
options ISICISA_AVM_A1
isic0 at isa? port 0x340 irq 5
```

USRobotics Sportster ISDN TA internal or Stollmann Tina-pp card:

```
options ISICISA_USR_STI
isic0 at isa? port 0x268 irq 5
```

ITK ix1 micro card:

```
options ISICISA_ITKIX1
isic0 at isa? port 0x398 irq 10
```

On the ISAPNP bus:

Teles S0/16.3 PnP card

```
options ISICPNP_TEL_S0_16_3_P
isic* at isapnp?
```

Creatix ISDN-S0 P&P card

```
options ISICPNP_CRTX_S0_P
isic* at isapnp?
```

Dr. Neuhaus Niccy GO@

```
options ISICPNP_DRN_NGO
isic* at isapnp?
```

ELSA QuickStep 1000pro (ISA version):

```
options ISICPNP_ELSA_QS1ISA
isic* at isapnp?
```

Sedlbauer WinSpeed:

```
options ISICPNP_SEDLBAUER
isic* at isapnp?
```

Dynalink IS64PH:

```
options ISICPNP_DYNALINK
isic* at isapnp?
```

Cards on the PCI bus:

ELSA QuickStep 1000pro (PCI version)
isic* at pci?

Cards on the PCMCIA or PCCARD bus:

AVM Fritz!Card PCMCIA
options ISICPCMCIA_AVM_A1
isic* at pcmcia? function ?

ELSA MicroLink ISDN/MC
options ISICPCMCIA_ELSA_ISDNMC
isic* at pcmcia? function ?

ELSA MicroLink MC/all
options ICISPCMCIA_ELSA_MCALL
isic* at pcmcia? function ?

Cards on the Amiga Zorro bus:

BSC/ITH ISDN Master or MasterII, ITH ISDN MasterII, Individual Computers ISDN Surfer, VMC ISDN Blaster, or Zeus ISDN Link
aster* at zbus?
isic* at aster? port ?

DESCRIPTION

The **isic** driver provides D-channel layer 1 supports as specified in ITU Recommendation I.430 and layer 1 support for the B-channel.

The driver supports several 8- and 16-bit passive ISDN cards from various manufacturers which are all based upon the popular Siemens ISDN chipset consisting of the ISDN Subscriber Access Controller ISAC (such as the PEB2085 or PSB 2186) and the High-Level Serial Communications Controller Extended HSCX (such as the SAB82525 or PSB21525). The newer IPAC chip (which integrates an ISAC and a HSCX in one chip, with the added benefit of larger FIFO buffers) is also supported.

SUPPORTED CARDS

Teles S0/8, Dr. Neuhaus Niccy 1008, Creatix ISDN-S0/8

Notice that this cards must not have a *port* value in the config line.

Valid interrupts are 2, 3, 4, 5, 6 and 7.

The i/o ports are memory mapped and the memory start address may be in the range 0xA0000 to 0xDF000 and uses 4kB of memory.

The optional *flag* value is 1.

Teles S0/16, Creatix ISDN-S0, Dr. Neuhaus Niccy 1016

These boards have a jumper which specifies an i/o base address of either 0xd80, 0xe80 or 0xf80. The remaining necessary configuration values are then programmed at run time by accessing this i/o port.

Valid interrupts are 2, 3, 4, 5, 10, 11, 12 or 15.

Valid memory start addresses are 0xC0000, 0xC2000, 0xC4000, 0xC6000, 0xC8000, 0xCA000, 0xCC000, 0xCE000, 0xD0000, 0xD2000, 0xD4000, 0xD6000, 0xD8000, 0xDA000, 0xDC000 and 0xDE000.

Notice: Although the Jumpers are labeled 0xd80, 0xe80 or 0xf80, they also require i/o space at addresses 0x180, 0x280 or 0x380.

The optional *flag* value is 2.

Teles S0/16.3

This card is completely i/o mapped and must not have an *iomem* statement in the config line.

Valid interrupts are 2, 5, 9, 10, 12 or 15.

Notice: Although the switch positions are labeled 0x180, 0x280 and 0x380, the card is to be configured at 0xd80, 0xe80 or 0xf80 respectively!

The optional *flag* value is 3.

AVM A1, AVM Fritz!Card

These boards have a jumper which specifies an i/o base address of either 0x200, 0x240, 0x300 or 0x340.

Valid interrupt configurations are 3, 4, 5, 6, 7, 10, 11, 12 or 15.

Older Versions of the AVM A1 also require setting of an IRQ jumper, newer versions of this and the Fritz!Card only have an i/o base jumper and the interrupt is set up at runtime by reprogramming a register.

This card is completely i/o mapped and must not have an *iomem* statement in the config line.

The optional *flag* value is 4.

Teles S0/16.3 PnP

Possible i/o port values are 0x580, 0x500 and 0x680. Possible interrupt configurations are 3, 5, 7, 10, 11 and 12.

The card is autoconfigured by the PnP kernel subsystem.

Creatix ISDN-S0 P&P

Valid i/o port values are 0x120, 0x180 and 0x100.

Valid interrupt configurations are 3, 5, 7, 10, 11 and 12.

The card is autoconfigured by the PnP kernel subsystem.

3Com USRobotics Sportster ISDN TA intern and Stollmann Tina pp

Valid i/o port values are 0x200, 0x208, 0x210, 0x218, 0x220, 0x228, 0x230, 0x238, 0x240, 0x248, 0x250, 0x258, 0x260, 0x268, 0x270 and 0x278.

Valid interrupt configurations are 5, 7, 10, 11, 12, 14, 15.

Notice: this card has a strange address decoding scheme resulting in 64 windows of some bytes length. Anyway, support for this card is good because the manufacturer gave out technical docs for this card!

The optional *flag* value is 7.

Dr. Neuhaus Niccy Go@

Valid i/o port values must be in the range 0x200 ... 0x3e0.

Valid interrupt configurations are 3, 4, 5, 9, 10, 11, 12, 15.

The card is autoconfigured by the PnP kernel subsystem.

Sedlbauer Win Speed

Valid i/o port values must be in the range 0x100 ... 0x3f0. (alignment 0x8, len 0x8)

Valid interrupt configurations are 3, 4, 5, 7, 10, 11, 12, 13, 15.

The card is autoconfigured by the PnP kernel subsystem.

ELSA QuickStep 1000pro (ISA)

I/O port in the range 0x160 ... 0x360 (occupies 8 bytes).

Valid interrupt configurations are 3, 4, 5, 7, 10, 11, 12, 15.

The card is autoconfigured by the PnP kernel subsystem.

ELSA QuickStep 1000pro-PCI

The card is autoconfigured by the PCI kernel subsystem.

ITK ix1 micro

Valid i/o port values must be in the range (<unknown>).

Valid interrupt configurations are (<unknown>).

The optional *flag* value is 18.

BSC ISDN Master (2092/64)
BSC ISDN MasterII (2092/65)
ITH ISDN MasterII (5000/1)
VMC ISDN Blaster (5001/1)
Zeus ISDN Link (2189/3)

The card addresses are autoconfigured by the Zorro bus kernel subsystem. The ISDN functions of the boards are at known (to the driver) relative addresses.

Note that currently, you have to jumper the card interrupt for *IPL 2* instead of *IPL 6* (which is used by most AmigaOS software).

Note that the ITH ISDN MasterII doesn't work in the DraCo Zorro bus. This is not a NetBSD-specific problem, but a general one.

Individual Computers ISDN Surfer (4626/5 serno 0)

The card addresses are autoconfigured by the Zorro bus kernel subsystem. The ISDN functions of the boards are at known (to the driver) relative addresses.

The card is operated by the driver at *IPL 2* instead of *IPL 6* (which is used by most AmigaOS software). Because of this, if an AmigaOS driver did lock the interrupt priority level of the card, your system might hang soon after boot. In this case, boot using the boot block, or without enabling the AmigaOS driver if you use loadbsd.

SEE ALSO

`isdnd`(8)

STANDARDS

CCITT Recommendation I.430

AUTHORS

The **isic** driver and this man page were written by Hellmuth Michaelis ⟨hm@kts.org⟩. It is based on earlier work of Arne Helme, Andrew Gordon and Gary Jennejohn.

The complete porting to and maintenance of NetBSD was done by Martin Husemann ⟨martin@NetBSD.org⟩.

The NetBSD/amiga ISDN Blaster/Master/MasterII driver was written by Ignatios Souvatzis ⟨is@NetBSD.org⟩.

BUGS

Note that all of the boards with I/O ports actually use several ranges of port addresses; Teles happen to refer to the 0xd80 range in their documentation (the board also uses 0x180 etc.), while AVM happen to refer to the 0x200 range in their documentation (the board also uses 0x600 etc.) The driver matches the manufacturers' description for the purposes of configuration, but of course makes use of all the ports in order to operate the card.

Since there is no hardware documentation available from several manufacturers for their boards, it is likely that there are many, many bugs left.

NAME

iso — ISO protocol family

SYNOPSIS

```
#include <sys/types.h>
#include <netiso/iso.h>
```

DESCRIPTION

The ISO protocol family is a collection of protocols that uses the ISO address format. The ISO family provides protocol support for the SOCK_SEQPACKET abstraction through the TP protocol (ISO 8073), for the SOCK_DGRAM abstraction through the connectionless transport protocol (ISO 8602), and for the SOCK_RAW abstraction by providing direct access (for debugging) to the CLNP (ISO 8473) network layer protocol.

ADDRESSING

ISO addresses are based upon ISO 8348/AD2, *Addendum to the Network Service Definition Covering Network Layer Addressing.*

Sockets bound to the OSI protocol family use the following address structure:

```
struct iso_addr {
     u_char    isoa_len;  /* length, not including this byte */
     char      isoa_genaddr[20];  /* general opaque address */
};

struct sockaddr_iso {
     u_char          siso_len;      /* size of this sockaddr */
     sa_family_t     siso_family;   /* addressing domain, AF_ISO */
     u_char          siso_plen;     /* presentation selector length */
     u_char          siso_slen;     /* session selector length */
     u_char          siso_tlen;     /* transport selector length */
     struct iso_addr siso_addr; /* network address */
     u_char          siso_pad[6];    /* space for gosip v2 SELs */
};
#define siso_nlen siso_addr.isoa_len
#define siso_data siso_addr.isoa_genaddr
```

The fields of this structure are:

siso_len:
: Length of the entire address structure, in bytes, which may grow to be longer than the 32 bytes shown above.

siso_family:
: Identifies the domain: AF_ISO.

siso_tlen:
: Length of the transport selector.

siso_slen:
: Length of the session selector. This is not currently supported by the kernel and is provided as a convenience for user level programs.

siso_plen:
: Length of the presentation selector. This is not currently supported by the kernel and is provided as a convenience for user level programs.

siso_addr:
The network part of the address, described below.

TRANSPORT ADDRESSING

An ISO transport address is similar to an Internet address in that it contains a network-address portion and a portion that the transport layer uses to multiplex its services among clients. In the Internet domain, this portion of the address is called a *port*. In the ISO domain, this is called a *transport selector* (also known at one time as a *transport suffix*). While ports are always 16 bits, transport selectors may be of (almost) arbitrary size.

Since the C language does not provide convenient variable length structures, we have separated the selector lengths from the data themselves. The network address and various selectors are stored contiguously, with the network address first, then the transport selector, and so on. Thus, if you had a network address of less than 20 bytes, the transport selector would encroach on space normally reserved for the network address.

NETWORK ADDRESSING

ISO network addresses are limited to 20 bytes in length. ISO network addresses can take any format.

PROTOCOLS

The ARGO 1.0 implementation of the ISO protocol family comprises the Connectionless-Mode Network Protocol (CLNP), and the Transport Protocol (TP), classes 4 and 0, and X.25. TP is used to support the `SOCK_SEQPACKET` abstraction. A raw interface to CLNP is available by creating an ISO socket of type `SOCK_RAW`. This is used for CLNP debugging only.

SEE ALSO

`clnp`(4), `cltp`(4), `tp`(4)

NAME

isp — Qlogic based SCSI and FibreChannel SCSI Host Adapters

SYNOPSIS

```
isp* at pci? dev? function? (PCI)
isp* at sbus? slot ? offset ? (SBus)
scsibus* at isp?
```

DESCRIPTION

This driver provides access to SCSI or FibreChannel devices.

SCSI features include support for Ultra SCSI and wide mode transactions for SCSI, and LVD (for the ISP1080 and ISP1280),

Fibre Channel support uses FCP SCSI profile for FibreChannel. and uses Class 3 connections only. Support is available for Public and Private loops. Command tagging is supported for all (in fact, FibreChannel requires tagging).

CONFIGURATION

An optional flags 0x80 appended to the above isp* declarations will disable the download of driver firmware, which means you use whatever firmware is running on the card. If no firmware is running on the card, the driver cannot operate the card.

An optional flags 0x40 appended to the above isp* declarations (can be OR'd in with the other config flags option) will keep the driver from looking at device or bus NVRAM settings (this is in case NVRAM is just wrong and you have the card in a platform where it is inconvenient to change NVRAM settings on the card).

HARDWARE

Supported cards include:

ISP1000 SBus Fast Wide, Ultra Fast Wide cards, Single Ended or Differential cards.

PTI SBS440
: Performance Technology ISP1000 variants.

ISP1020 Qlogic 1020 Fast Wide and Differential Fast Wide PCI cards.

ISP1040 Qlogic 1040 Ultra Wide and Differential Ultra Wide PCI cards.

PTI SBS450
: Performance Technology ISP1040 variants.

Qlogic 1240
: Qlogic 1240 Dual Bus Ultra Wide and Differential Ultra Wide PCI cards.

Qlogic 1080
: Qlogic 1280 LVD Ultra2 Wide PCI cards.

Qlogic 1280
: Qlogic 1280 Dual Bus LVD Ultra2 Wide PCI cards.

Qlogic 2100
: Qlogic 2100 and 2100A Copper and Optical Fibre Channel Arbitrated Loop

Qlogic 2102
: Qlogic Dual Loop 2100A Optical Fibre Channel Arbitrated Loop PCI cards.

Qlogic 2200
: Qlogic 2200 Copper and Optical Fibre Channel Arbitrated Loop PCI cards.

Qlogic 2202
: Qlogic 2200 Dual Bus Optical Fibre Channel Arbitrated Loop PCI cards.

Qlogic 2204
: Qlogic 2200 Quad Bus Optical Fibre Channel Arbitrated Loop PCI cards.

Qlogic 2300
: Qlogic 2300 2-Gigabit Optical Fibre Channel PCI cards.

Qlogic 2312
: Qlogic 2300 2-Gigabit Dual Channel Optical Fibre Channel PCI cards.

PTI SBS470
: Performance Technology ISP2100 variants.

Antares P-0033
: Antares Microsystems ISP2100 variants.

SEE ALSO

`cd`(4), `intro`(4), `scsi`(4), `sd`(4), `st`(4)

AUTHORS

The **`isp`** driver was written by Matthew Jacob for NASA/Ames Research Center.

BUGS

The driver currently ignores some NVRAM settings.

The driver currently doesn't do error recovery for timed out commands very gracefully.

Sometimes, when booting, the driver gets stuck waiting for the Fibre Channel firmware to tell it that the loop port database is ready. In this case you'll see an announcement that the loop state has a value of 0x1. To unwedge the system, unplug and replug the fibre channel connection, or otherwise cause a LIP (Loop Initialization Primitive sequence) - this will kick the firmware into getting unstuck.

NAME

isv — IDEC Supervision/16 image capture board

SYNOPSIS

```
isv0 at isa? port 0x2f0
isv0 at isa? port 0x2e0
isv0 at isa? port 0x3f0
isv0 at isa? port 0x3e0
```

DESCRIPTION

isv is a driver for the IDEC Supervision/16, an image capture board that plugs into a 16-bit ISA bus. The IDEC Supervision/16 digitizes an NTSC television signal, storing a 512 x 480-pixel, 8-bit grayscale image in its 256kB dynamic RAM array every 1/30th of a second. The host reads frames from the DRAM using 122881 16-bit I/O reads. Reading frames from the Supervision/16 is quite slow: after the host reads a 16-bit word from the DRAM, the Supervision/16 state machine takes approximately 0.5 microseconds to get ready for the next read. Theoretically, a frame rate of approximately 10 frames per second is possible. **isv** achieves a frame rate of approximately 6 frames per second.

SEE ALSO

isvctl(8)

Programming the Supervision/16 Image Capture Board, IDEC, circa 1991.

HISTORY

The **isv** device first appeared in NetBSD 5.0.

AUTHORS

The **isv** driver was written by David Young ⟨dyoung@NetBSD.org⟩.

BUGS

Synchronizing with the hardware and reading frames from it is very CPU-intensive.

isv will not detect the capture board if it is not attached to an active video source. To force NetBSD to detect the capture board at any time, re-scan the ISA bus using, e.g., **drvctl -r** *isa0*.

NAME

ite — Amiga Internal Terminal Emulator

SYNOPSIS

```
ite0 at grf0
ite1 at grf1
ite2 at grf2
ite3 at grf3
ite4 at grf4
ite5 at grf5
ite6 at grf6
ite7 at grf7
```

DESCRIPTION

TTY special files of the form ‘‘ttye?’’ are interfaces to the Amiga ITE for bit-mapped displays. An **ite** is the main system console on most Amiga workstations and is the mechanism through which a user communicates with the machine. If more than one of the supported displays exists on a system, any or all can be used as **ite** s with the limitation that only one will have a keyboard (since only one keyboard is supported) and only one of each type can be used.

ite devices use the HP-UX ‘300h’ termcap(5) entry. However, as currently implemented, the **ite** does not support the full range of HP-UX capabilities for this device. Missing are multiple colors, blinking, soft-keys, programmable tabs, scrolling memory and keyboard arrow keys. The keyboard will use the left and right *Amiga* keys as meta keys, in that it will set the eighth bit of the character code. **ite** devices also do a good job at emulating the ‘vt100’ termcap(5) entry.

Upon booting, the kernel will first look for an **ite** device to use as the system console (/dev/console). If a display exists at any hardware address, it will be the console. The kernel looks for them in decreasing order (that is, choosing the highest-numbered one).

On most systems, a display is used both as an **ite** (/dev/ttye? aka /dev/console) and as a graphics device (/dev/grf?). In this environment, there is some interaction between the two uses that should be noted. For example, opening /dev/grf0 will deactivate the **ite** that is, write over whatever may be on the **ite** display. When the graphics application is finished and /dev/grf0 closed, the **ite** will be reinitialized with the frame buffer cleared and the old colormap installed.

SEE ALSO

grf(4), kbd(4)

HISTORY

The Amiga **ite** first appeared in NetBSD 1.0

NAME

ite — HP Internal Terminal Emulator graphics driver

SYNOPSIS

ite* at grf?

DESCRIPTION

TTY special files of the form ‘‘ttye?’’ are interfaces to the HP ITE for bit-mapped displays as implemented under BSD. An ITE is the main system console on most HP300 workstations and is the mechanism through which a user communicates with the machine. If more than one display exists on a system, any or all can be used as ITEs with the limitation that only the first one opened will have a keyboard (since only one keyboard is supported).

ITE devices use the HP-UX ‘`300h`’ `termcap`(5) or `terminfo`(5) entries. However, as currently implemented, the ITE does not support the full range of HP-UX capabilities for this device. Missing are multiple colors, underlining, blinking, softkeys, programmable tabs, scrolling memory and keyboard arrow keys. The keyboard does not have any of the international character support of HP’s NLS system. It does use the left and right *extend char* keys as meta keys, in that it will set the eighth bit of the character code.

Upon booting, the kernel will first look for an ITE device to use as the system console (`/dev/console`). If a display exists at any hardware address, it will be the console. The kernel looks for, in order: a 98544, 98545, or 98547 Topcat display, a 98700 Gatorbox at a supported address (see `gbox`(4)), or a 98720 Renaissance at a supported address (see `rbox`(4)). Currently there is no ITE support for the 98548, 98549, 98550 and 98556 boards.

When activated as an ITE (special file opened), all displays go through a standard initialization sequence. The frame buffer is cleared, the ROM fonts are unpacked and loaded into off-screen storage and a cursor appears. The ITE initialization routine also sets the colormap entry used to white. Variable colors are not used, mainly for reasons of simplicity. The font pixels are all set to 0xff and the colormap entry corresponding to all planes is set to R=255, G=255 and B=255. The actual number of planes used to display the characters depends on the hardware installed. Finally, if the keyboard HIL device is not already assigned to another ITE device, it is placed in ‘‘cooked’’ mode and assigned to this ITE.

On most systems, a display is used both as an ITE (`/dev/ttye?` aka `/dev/console`) and as a graphics device (/dev/grf?). In this environment, there is some interaction between the two uses that should be noted. For example, opening `/dev/grf0` will deactivate the ITE, that is, write over whatever may be on the ITE display. When the graphics application is finished and `/dev/grf0` closed, the ITE will be reinitialized with the frame buffer cleared and the ITE colormap installed.

DIAGNOSTICS

None under BSD.

SEE ALSO

`grf`(4), `hil`(4), `tty`(4)

NAME

iteide — Integrated Technology Express IDE disk controllers driver

SYNOPSIS

iteide* **at pci? dev ? function ? flags 0x0000**

DESCRIPTION

The **iteide** driver supports the IT8211 and IT8212 IDE controllers which are found on some Gigabyte motherboards (as "GigaRAID") and some PCI cards. This driver provides the interface with the hardware for the ata(4) driver.

The optional RAID functionality of the IT8211 and IT8212 is not supported and is explicitly disabled by the **iteide** driver. The controller will act like a regular IDE controller with no RAID functionality.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

AUTHORS

The **iteide** driver was written by Alexander Yurchenko ⟨grange@openbsd.org⟩ and ported to NetBSD by Grant Beattie ⟨grant@NetBSD.org⟩.

NAME

itesio — ITE IT87xxF Super I/O driver

SYNOPSIS

```
itesio0 at isa? port 0x2e
itesio1 at isa? port 0x4e
```

DESCRIPTION

The **itesio** driver provides support for the Environment Controller and the Watchdog Timer on the IT87xxF Super I/Os, and may be used to monitor hardware sensors or setting up a watchdog timeout value for the system.

The **itesio** driver has 15 sensors:

Sensor	Units	Typical Use
Temp0	uK	CPU Temp
Temp1	uK	System Temp
Temp2	uK	Aux Temp
VIN0	uV DC	Vcore A
VIN1	uV DC	Vcore B
VIN2	uV DC	+3.3V
VIN3	uV DC	+5V
VIN4	uV DC	+12V
VIN5	uV DC	-12V
VIN6	uV DC	-5V
VIN7	uV DC	STANDBY
VBAT	uV DC	VBAT
Fan0	RPM	CPU Fan
Fan1	RPM	System Fan
Fan2	RPM	Aux Fan

The **itesio** Watchdog Timer is configurable via the wdogctl(8) utility and has a resolution between 1 and 65535 seconds. The Watchdog Timer is not supported on the IT8705 Super I/O.

SEE ALSO

envsys(4), envstat(8), wdogctl(8)

HISTORY

The **itesio** driver first appeared in OpenBSD 3.4 and then it was ported to NetBSD 4.0.

AUTHORS

The **itesio** driver was written by Julien Bordet ⟨zejames@greyhats.org⟩ and Juan Romero Pardines ⟨xtraeme@netbsd.org⟩.

BUGS

Interrupt support is unimplemented.

NAME

iwi — Intel PRO/Wireless 2200BG/2915ABG IEEE 802.11 driver

SYNOPSIS

iwi* **at pci? dev ? function ?**

DESCRIPTION

The **iwi** driver provides support for Intel(R) PRO/Wireless 2200BG and 2915ABG MiniPCI network adapters.

By default, the **iwi** driver configures the adapter for BSS operation (aka infrastructure mode). This mode requires the use of an access point.

For more information on configuring this device, see `ifconfig`(8).

The **iwi** driver requires the `pkgsrc/sysutils/iwi-firmware3` package to be installed and loaded for proper functionality.

EXAMPLES

Join an existing BSS network (i.e.: connect to an access point):

```
ifconfig iwi0 inet 192.168.0.20 netmask 0xffffff00
```

Join a specific BSS network with network name "`my_net`":

```
ifconfig iwi0 inet 192.168.0.20 netmask 0xffffff00 nwid my_net
```

Join a specific BSS network with 64 bits WEP encryption:

```
ifconfig iwi0 inet 192.168.0.20 netmask 0xffffff00 nwid my_net \
        nwkey 0x1234567890
```

Join a specific BSS network with 128bits WEP encryption:

```
ifconfig iwi0 inet 192.168.0.20 netmask 0xffffff00 nwid my_net \
        nwkey 0x01020304050607080910111213
```

DIAGNOSTICS

iwi%d: device timeout The driver will reset the hardware. This should not happen.

SEE ALSO

`an`(4), `awi`(4), `ipw`(4), `pci`(4), `wi`(4), `ifconfig`(8), `iwictl`(8), `firmload`(9), `pkgsrc/sysutils/iwi-firmware3`

The IWI Web Page, http://damien.bergamini.free.fr/ipw/.

AUTHORS

The **iwi** driver and this man page were written by Damien Bergamini ⟨damien.bergamini@free.fr⟩.

NAME

iwic — isdn4bsd Winbond ISDN Chip device driver

SYNOPSIS

iwic* **at pci?**

DESCRIPTION

The **iwic** driver provides D-channel layer 1 support as specified in ITU Recommendation I.430 and layer 1 support for the B-channel.

The driver supports passive PCI ISDN cards from various manufacturers based on the Winbond W6692 chip.

SUPPORTED CARDS

ASUSCOM P-IN100-ST-D

Dynalink IS64PPH

SEE ALSO

isdnd(8)

STANDARDS

CCITT Recommendation I.430

AUTHORS

The **iwic** driver was written by Dave Boyce ⟨dave@abyss.demon.co.uk⟩.

This manpage was written by Hellmuth Michaelis ⟨hm@freebsd.org⟩.

CAVEATS

The driver is still in a somewhat experimental state.

BUGS

Layer 1 persistent deactivation not yet implemented.

NAME

iwm, **fd** — floppy disk driver for IWM and non-DMA SWIM controllers

SYNOPSIS

```
iwm0 at obio?
fd* at iwm0 drive ?
```

DESCRIPTION

The **iwm** driver interfaces to the built-in and external floppy disk drives on the Macintosh. It supports double-density media, written in Apple's proprietary GCR format. Currently, there is no disklabel support for the floppy drives. Instead, the **iwm** driver sets up a fake in-core disklabel, using the minor device number to select from the supported disk formats.

The following formats are supported:

Partition	Size	sides	tracks	sectors/track	
a	800Kb	2	80	10	(default)
b	400Kb	1	80	10	
c	800Kb	2	80	10	

(The above table describes the logical mapping as implemented by the driver; the physical layout of GCR floppies has 8..12 sectors per track.)

FORMATTING

The **iwm** driver does currently not support floppy disk formatting.

SEE ALSO

Apple Computer, Inc.: "Inside Macintosh", Vol III-33f. (Addison-Wesley)

Apple Computer, Inc.: "New Technical Notes DV 17 - Sony Driver"

Neil Parker: "iwmstuff"

`eject`(1)

HISTORY

The **iwm** interface first appeared in NetBSD 1.4.

AUTHORS

Hauke Fath put together the beginnings of the **iwm** driver in 1996 from the sparse documentation in "Inside Macintosh", Neil Parker's "iwmstuff" documentation for the Apple IIgs and a long, hard look at the .Sony driver.

BUGS

The FFS code is incapable of dealing with a varying number of sectors per track. We have to fake a mapping and so lose FFS support for hardware parameters like transition times.

The driver only supports an obsolete format.

NAME

ix — Intel EtherExpress/16 Ethernet ISA bus NIC driver

SYNOPSIS

```
ix0 at isa? port 0x300 irq 10
```

DESCRIPTION

The **ix** device driver supports the EtherExpress/16 card, and might support other ISA bus cards using the same chip. The EtherExpress/16 Ethernet adapter is based on the Intel 82586 Ethernet chip.

SEE ALSO

ai(4), ef(4), elmc(4), ifmedia(4), intro(4), ifconfig(8)

NAME

ix — Interlan Np100 10 Mb/s Ethernet interface

SYNOPSIS

```
np0 at uba0 csr 166000 vector npintr
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **ix** interface provides access to a 10 Mb/s Ethernet network through an Interlan Np100 controller used as a link-layer interface.

This interface is unusual in that it requires loading firmware into the controller before it may be used as a network interface. This is accomplished by opening a character special device, and writing data to it. A program to load the image is provided in `/usr/src/new/np100`. The sequence of commands would be:

```
# ./npload np.image [/dev/np<board #> if other than np00]
# sleep 10
# ifconfig ix0 ...
```

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **ix** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

The interface normally tries to use a "trailer" encapsulation to minimize copying data on input and output. The use of trailers is negotiated with ARP. This negotiation may be disabled, on a per-interface basis, by setting the `IFF_NOTRAILERS` flag with an `SIOCSIFFLAGS` `ioctl`(2).

DIAGNOSTICS

ix%d: Req failed, cmd %x, stat %x, ust error %x,%x. The firmware in the controller refused to honor a request from UNIX in initializing packet level communications. The board may need to be reset and reloaded. Or, you may not have allowed enough time between loading the board and issuing the request to begin UNIX network operation.

ix%d: can't initialize. The interface was unable to obtain UNIBUS resources required for operation.

ix%d: failed to reinitialize DLA module. The interface got sick after attempting to reprogram its physical Ethernet address. Try reloading the firmware. The attempt is made only when this interfaces is not the first one configured for XNS.

ix%d: can't handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

ix%d: stray xmit interrupt, npreq=%x. This may happen if the board is reloaded while network processes are still running.

ixrint: cqe error %x, %x, %x. This will result if an `ifconfig`(8) request is made at an inopportune time, such as not allowing enough time after loading the firmware. After 100 such errors are logged, the UNIX network driver will shut itself down, saying:

ixrint: shutting down unix dla. The recourse is to reload the firmware and allow more time.

SEE ALSO

`arp`(4), `inet`(4), `netintro`(4), `np`(4)

HISTORY

The **ix** driver appeared in 4.3BSD.

NAME

ixpide — PCI IDE disk controllers driver

SYNOPSIS

ixpide* **at pci? dev ? function ? flags 0x0000**

DESCRIPTION

The **ixpide** driver supports the ATI Technologies IXP IDE controller, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **ixpide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

HARDWARE

The supported IDE controllers are

- ATI SB200
- ATI SB300
- ATI SB400, Parallel ATA
- ATI SB400, Serial ATA

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

NAME

iy — Intel EtherExpress PRO/10 Ethernet driver (Intel i82595)

SYNOPSIS

```
iy0 at isa? port {port} irq ?
```

DESCRIPTION

The **iy** device driver supports the EtherExpress PRO/10 card, and might support other ISA cards using the same chip.

MEDIA SELECTION

The different models of the supported boards come with some subset of RJ-45, BNC and AUI connectors. Supported media include:

AUI/DIX	Standard 15 pin connector
10Base2	BNC, also known as thin-net
10BaseT	UTP, also known as twisted pair

The default port to use is the port the card autodetects at **ifconfig up** time. To choose an alternative port, an explicit medium can be specified to ifconfig(8) or in your ifconfig.iy? file.

NOTES

The EtherExpress PRO card has no jumpers to set the address. Intel supplies software to set the address of the card in software. You have to hardwire this address in your kernel configuration file.

SEE ALSO

ed(4), eg(4), el(4), ep(4), intro(4), ix(4), le(4), ifconfig(8)

STANDARDS

are great. There's so many to choose from.

NAME

j6x0lcd — driver for Jornada 680 LCD screen

SYNOPSIS

```
j6x0lcd* at shb?
```

DESCRIPTION

The **j6x0lcd** driver provides support for controlling brightness, contrast, and power of the LCD screen in Jornada 680 series machines.

The default keymap binds ⟨Ctrl⟩ + ⟨Alt⟩ + ⟨*arrowkey*⟩ to control brightness with up and down arrow keys, and to control contrast with left and right arrow keys.

SEE ALSO

screenblank(1)

HISTORY

The **j6x0lcd** driver first appeared in NetBSD 2.0.

NAME

j6x0tp — driver for Jornada 680 touch-screen

SYNOPSIS

```
j6x0tp* at adc?
wsmouse* at j6x0tp? mux 0
wskbd* at j6x0tp? mux 1

options J6X0TP_WSMOUSE_EXCLUSIVE
options J6X0TP_SETTINGS_ICON_KEYSYM=keysym
options J6X0TP_PGUP_ICON_KEYSYM=keysym
options J6X0TP_PGDN_ICON_KEYSYM=keysym
options J6X0TP_SWITCH_ICON_KEYSYM=keysym
```

DESCRIPTION

The **j6x0tp** driver provides support for the Jornada 680/690 touch-screen.

Pen movements are passed to wsmouse(4) as mouse clicks and drags.

Taps on the on-screen “HP hard icons” are passed to wskbd(4) as key presses.

The **j6x0tp** driver has the following config options.

options J6X0TP_WSMOUSE_EXCLUSIVE
Stop feeding input to wskbd(4) when the attached wsmouse(4) device is opened.

options J6X0TP_SETTINGS_ICON_KEYSYM=keysym

options J6X0TP_PGUP_ICON_KEYSYM=keysym

options J6X0TP_PGDN_ICON_KEYSYM=keysym

options J6X0TP_SWITCH_ICON_KEYSYM=keysym
Customize the mapping of on-screen “HP hard icons” to keys. Default keymap assigns KS_Home, KS_Prior, KS_Next, and KS_End respectively. It is possible to change the mapping at run time with wsconsctl(8), but for a “keyboard” as small as this one it is convenient to be able to change the defaults at compile time.

SEE ALSO

adc(4), wskbd(4), wsmouse(4), tpctl(8)

HISTORY

The **j6x0tp** driver first appeared in NetBSD 2.0.

NAME

j720kbd — driver for Jornada 710/720/728 keyboard

SYNOPSIS

```
j720kbd* at j720ssp?
hpckbd* at j720kbd?
wskbd* at hpckbd? mux 1
```

DESCRIPTION

The **j720kbd** driver provides support for the keyboard in the Jornada 720 series machines.

Pressing the on/off button will suspend the machine. Pressing this key again will cause it to resume.

SEE ALSO

hpckbd(4), wskbd(4), wsconsctl(8)

NAME

j720lcd — driver for Jornada 710/720/728 LCD screen

SYNOPSIS

```
j720lcd* at j720ssp?
```

DESCRIPTION

The **j720lcd** driver provides support for controlling brightness, contrast, and power of the LCD screen in the Jornada 720 series machines.

The default keymap binds ⟨Ctrl⟩ + ⟨Alt⟩ + ⟨*arrowkey*⟩ to control brightness with up and down arrow keys, and to control contrast with left and right arrow keys.

SEE ALSO

screenblank(1), wsconsctl(8)

NAME

j720tp — driver for Jornada 710/720/728 touch-screen

SYNOPSIS

```
j720tp* at j720ssp?
wsmouse* at j720tp? mux 0
wskbd* at j720tp? mux 1
```

DESCRIPTION

The **j720tp** driver provides support for the touch-screen in the Jornada 720 series machines.

Pen movements are passed to wsmouse(4) as mouse clicks and drags.

When **wskbd* at j720tp?** is enabled, taps on the on-screen “HP hard icons” are passed to wskbd(4) as key presses.

The default icon to key mapping is:

Settings icon	KS_Home
Backup icon	KS_Prior
Dialup icon	KS_Next
Media Player icon	KS_End

SEE ALSO

wskbd(4), wsmouse(4), tpctl(8), wsconsctl(8)

NAME

jensenio — DEC 2000/300 (Jensen) I/O module

SYNOPSIS

jensenio* **at mainbus0**

DESCRIPTION

The **jensenio** driver provides support for the I/O module found on the DEC 2000/300. The module is comprised of two things:

- VLSI VL82C106 junk I/O chip; and
- Intel EISA bus interface.

The following devices are supported by the **jensenio** driver:

com	serial communications interface
eisa	machine-independent EISA bus
isa	machine-independent ISA bus
lpt	Parallel port driver
mcclock	DS1287 real-time clock
pckbc	PC keyboard controller driver

SEE ALSO

com(4), eisa(4), intro(4), isa(4), lpt(4), mcclock(4), pckbc(4)

NAME

jmide — JMicron Technology JMB36x PCIe to SATA II/PATA controller driver

SYNOPSIS

```
jmide* at pci? dev ? function ? flags 0x0000
ahcisata* at jmide?
```

DESCRIPTION

The **jmide** driver supports the JMicron Technology JMB36x IDE controllers.

These PCI-e controllers exist in different flavors (1 or 2 PATA Ultra/133 ports and/or 1 or 2 SATA-II ports), and are highly flexible. The SATA ports can be attached to the integrated AHCI controller or attached to the PCI IDE channels in PATA emulation, in either single-drive emulation on each PCI IDE channels, or in master/slave emulation on one of the PCI IDE channels.

When enabled, the AHCI controller can either be on the PCI function 0 or function 2. The PCI IDE controller can also be on either PCI function, and can share the PCI function with AHCI. The **jmide** driver supports both the AHCI controller and PCI IDE controller in various configurations.

SEE ALSO

ahcisata(4), ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

AUTHORS

The **jmide** driver was written by Manuel Bouyer.

NAME

joy — game adapter driver

SYNOPSIS

```
joy* at acpi?
joy* at eap?
joy* at eso?
joy0 at isa? port 0x201
joy* at isapnp?
joy* at ofisa?
joy* at pci?
joy* at pnpbios? index ?
```

DESCRIPTION

This driver provides access to the game adapter. The lower bit in the minor device number selects the joystick: 0 is the first joystick and 1 is the second.

The game control adapter allows up to two joysticks to be attached to the system. The adapter plus the driver convert the present resistive value to a relative joystick position. On receipt of an output signal, four timing circuits are started. By determining the time required for the circuit to time-out (a function of the resistance), the paddle position can be determined. The adapter could be used as a general purpose I/O card with four analog (resistive) inputs plus four digital input points.

Applications may call `ioctl`(2) on a game adapter driver file descriptor to set and get the offsets of the two potentiometers and the maximum time-out value for the circuit. The `ioctl`(2) commands are listed in ⟨`machine/joystick.h`⟩ and currently are:

`JOY_SETTIMEOUT`	Sets the maximum time-out for the adapter.
`JOY_GETTIMEOUT`	Returns the current maximum time-out.
`JOY_SET_X_OFFSET`	Sets an offset on X value.
`JOY_GET_X_OFFSET`	Returns the current X offset.
`JOY_SET_Y_OFFSET`	Sets an offset on Y value.
`JOY_GET_Y_OFFSET`	Returns the current Y offset.

All these commands take an integer parameter.

`read`(2) on the file descriptor returns a *`joystick`* structure:

```
struct joystick {
        int x;
        int y;
        int b1;
        int b2;
};
```

The fields have the following functions:

x current X coordinate of the joystick (or position of paddle 1)

y current Y coordinate of the joystick (or position of paddle 2)

b1 current state of button 1

b2 current state of button 2

The b1 and b2 fields in struct joystick are set to 1 if the corresponding button is down, 0 otherwise.

The x and y coordinates are supposed to be between 0 and 255 for a good joystick and a good adapter. Unfortunately, because of the hardware hack that is used to measure the position (by measuring the time needed to discharge an RC circuit made from the joystick's potentiometer and a capacitor on the adapter), calibration is needed to determine exactly what values are returned for a specific joystick/adapter combination. Incorrect hardware can yield negative or values greater than 255.

A typical calibration procedure uses the values returned at lower left, center and upper right positions of the joystick to compute the relative position.

This calibration is not part of the driver.

FILES

`/dev/joy0`	first joystick
`/dev/joy1`	second joystick

SEE ALSO

`acpi`(4), `eap`(4), `eso`(4), `isa`(4), `isapnp`(4), `ofisa`(4), `pci`(4), `pnpbios`(4)

AUTHORS

Jean-Marc Zucconi wrote the FreeBSD driver. Matthieu Herrb ported it to NetBSD and wrote this manual page.

NAME

kbd — Sun workstation keyboard

SYNOPSIS

```
pseudo-device kbd
```

DESCRIPTION

The **kbd** driver provides an interface to the workstation console keyboard. The Sun "type 2", "type 3", "type 4", and "type 5" keyboards are supported. The "type 5" keyboard is treated as if it were a "type 4". All types generate keycodes encoding the key identity and motion (up or down) as the keys are pressed and released. The **kbd** driver either passes the keycodes to an application as they come in (e.g. X(1)), or translates them into ASCII characters first according to a set of built-in tables.

If the **kbd** is configured as the device to be used for system console input (see openprom(4)), it will be internally connected to the /dev/console device special file, which can be used as a tty(4) device.

The device special file /dev/kbd is used to get direct access to the keyboard input stream.

The following ioctl's are supported (mostly just enough to keep the X(1) server going):

KIOCTRANS Set translation mode. The argument is of type *int* *, the only value supported is TR_UNTRANS_EVENT.

KIOCGTRANS Get translation mode. The argument is of type *int* *. TR_UNTRANS_EVENT is always returned.

KIOCGETKEY Fill in old-style key station translation. The argument is of type *struct okiockey* *.

KIOCCMD Send a command to the keyboard. The argument is of type *int* *, and can have one of the following values:

KBD_CMD_BELL Start the keyboard beeper.

KBD_CMD_NOBELL
Stop the keyboard beeper.

KBD_CMD_CLICK Instruct the keyboard to make extra noise when touching keys.

KBD_CMD_NOCLICK
Instruct the keyboard to stop making extra noise when touching keys.

KIOCTYPE Get keyboard type. The argument is of type *int* *, in which one of the values KB_SUN2, KB_SUN3 or KB_SUN4 will be returned.

KIOCSDIRECT
Route the keyboard input stream through the SunOS compatible event module. The argument is of type *int* *, a non-zero value will put the driver into "event" mode, while a value of zero will make it return to "ASCII translation" mode.

KIOCSKEY Set key station translation. The argument is of type *struct kiockey* * (see /usr/include/machine/kbio.h for more details).

KIOCGKEY Get key station translation. The argument is of type *struct kiockey* *.

KIOCLAYOUT Get keyboard layout ("type 4" only). The argument is of type *int* *, in which the uninterpreted result of the KBD_CMD_GLAYOUT keyboard command is returned (on KB_SUN4 type keyboards this will be the setting of a DIP switch bank).

KIOCSLED Set LED state ("type 4" only). The argument is of type *char *, and is the inclusive OR of the following flags:

LED_NUM_LOCK
LED_COMPOSE
LED_SCROLL_LOCK
LED_CAPS_LOCK

Each of these flags turn on the LED in the obvious key.

KIOCGLED Get LED state ("type 4" only). The argument is of type *char *, in which the current LED state is returned.

SEE ALSO

ms(4)

BUGS

kbd is hardwired to the built-in *zs1* serial port at 1200 bps.

NAME

kbd — Sun workstation keyboard

SYNOPSIS

```
kbd0 at zstty?
```

DESCRIPTION

The **kbd** driver provides an interface to the workstation console keyboard. "type 2", "type 3", "type 4", and "type 5" keyboards are supported. The "type 5" keyboard is treated as if it were a "type 4". All types generate keycodes encoding the key identity and motion (up or down) as the keys are pressed and released. The **kbd** driver either passes the keycodes to an application as they come in, or translates them into ASCII characters first according to a set of built-in tables.

If the keyboard is configured as the device to be used for system console input, it will be internally connected to the `/dev/console` device special file, which can be used as a `tty`(4) device.

The device special file `/dev/kbd` is used to get direct access to the keyboard input stream. The following ioctl's are supported (mostly just enough to keep the `X`(1) server going):

KIOCTRANS Set translation mode. The argument is of type *`int *`*, the only value supported is `TR_UNTRANS_EVENT`.

KIOCGTRANS Get translation mode. The argument is of type *`int *`*. `TR_UNTRANS_EVENT` is always returned.

KIOCGETKEY Fill in old-style key station translation. The argument is of type *`struct okiockey *`*.

KIOCCMD Send a command to the keyboard. The argument is of type *`int *`*, and can have one of the following values:

KBD_CMD_BELL Start the keyboard beeper.

KBD_CMD_NOBELL
Stop the keyboard beeper.

KBD_CMD_CLICK Instruct the keyboard to make extra noise when touching keys.

KBD_CMD_NOCLICK
Instruct the keyboard to stop making extra noise when touching keys.

KIOCTYPE Get keyboard type. The argument is of type *`int *`*, in which one of the values `KB_SUN2`, `KB_SUN3` or `KB_SUN4` will be returned.

KIOCSDIRECT
Route the keyboard input stream through the SunOS compatible event module. The argument is of type *`int *`*, a non-zero value will put the driver into "event" mode, while a value of zero will make it return to "ASCII translation" mode.

KIOCSKEY Set key station translation. The argument is of type *`struct kiockey *`* (see `/usr/include/machine/kbio.h` for more details).

KIOCGKEY Get key station translation. The argument is of type *`struct kiockey *`*.

KIOCLAYOUT Get keyboard layout ("type 4" only). The argument is of type *`int *`*, in which the uninterpreted result of the `KBD_CMD_GLAYOUT` keyboard command is returned (on `KBDUN4` type keyboards this will be the setting of a DIP switch bank).

KIOCSLED Set LED state ("type 4" only). The argument is of type *char *, and is the inclusive OR of the following flags:

LED_NUM_LOCK
LED_COMPOSE
LED_SCROLL_LOCK
LED_CAPS_LOCK

Each of these flags turn on the LED in the obvious key.

KIOCGLED Get LED state ("type 4" only). The argument is of type *char *, in which the current LED state is returned.

SEE ALSO

ms(4)

BUGS

kbd is hardwired to the built-in *zs1* serial port at 1200 bps.

NAME

kbd — Sun workstation keyboard

SYNOPSIS

```
pseudo-device kbd
```

DESCRIPTION

The **kbd** driver provides an interface to the workstation console keyboard. "type 2", "type 3", "type 4", and "type 5" keyboards are supported. The "type 5" keyboard is treated as if it were a "type 4". All types generate keycodes encoding the key identity and motion (up or down) as the keys are pressed and released. The **kbd** driver either passes the keycodes to an application as they come in, or translates them into ASCII characters first according to a set of built-in tables.

If the keyboard is configured as the device to be used for system console input (see `eeprom`(8)), it will be internally connected to the `/dev/console` device special file, which can be used as a `tty`(4) device.

The device special file `/dev/kbd` is used to get direct access to the keyboard input stream. The following ioctl's are supported (mostly just enough to keep the `X`(1) server going):

KIOCTRANS Set translation mode. The argument is of type *`int *`*, the only value supported is `TR_UNTRANS_EVENT`.

KIOCGTRANS Get translation mode. The argument is of type *`int *`*. `TR_UNTRANS_EVENT` is always returned.

KIOCGETKEY Fill in old-style key station translation. The argument is of type *`struct okiockey *`*.

KIOCCMD Send a command to the keyboard. The argument is of type *`int *`*, and can have one of the following values:

KBD_CMD_BELL Start the keyboard beeper.

KBD_CMD_NOBELL
Stop the keyboard beeper.

KBD_CMD_CLICK Instruct the keyboard to make extra noise when touching keys.

KBD_CMD_NOCLICK
Instruct the keyboard to stop making extra noise when touching keys.

KIOCTYPE Get keyboard type. The argument is of type *`int *`*, in which one of the values `KB_SUN2`, `KB_SUN3` or `KB_SUN4` will be returned.

KIOCSDIRECT
Route the keyboard input stream through the SunOS compatible event module. The argument is of type *`int *`*, a non-zero value will put the driver into "event" mode, while a value of zero will make it return to "ASCII translation" mode.

KIOCSKEY Set key station translation. The argument is of type *`struct kiockey *`* (see `/usr/include/machine/kbio.h` for more details).

KIOCGKEY Get key station translation. The argument is of type *`struct kiockey *`*.

KIOCLAYOUT Get keyboard layout ("type 4" only). The argument is of type *`int *`*, in which the uninterpreted result of the `KBD_CMD_GLAYOUT` keyboard command is returned (on `KBDUN4` type keyboards this will be the setting of a DIP switch bank).

KIOCSLED Set LED state ("type 4" only). The argument is of type *char *, and is the inclusive OR of the following flags:

LED_NUM_LOCK
LED_COMPOSE
LED_SCROLL_LOCK
LED_CAPS_LOCK

Each of these flags turn on the LED in the obvious key.

KIOCGLED Get LED state ("type 4" only). The argument is of type *char *, in which the current LED state is returned.

SEE ALSO

eeprom(4), ms(4), eeprom(8)

BUGS

kbd is hardwired to the built-in *zs1* serial port at 1200 bps.

NAME

kft — KFTIA and KFTHA Bus Adapter Node for I/O hoses

SYNOPSIS

kft* **at tlsb? node ? offset ?**

DESCRIPTION

The **kft** driver provides support for the KFTIA and KFTHA Bus Adapter Node for I/O hoses found on AlphaServer 8x00 systems.

The following devices are supported by the **kft** driver:

dwlpx ???

SEE ALSO

dwlpx(4), intro(4), tlsb(4)

NAME

kg — KL-11/DL-11W line clock

SYNOPSIS

```
kg0 at uba0 csr 0176500 vector kglock
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

A KL-11 or DL-11W can be used as an alternative real time clock source. When configured, certain system statistics and, optionally, system profiling work will be collected each time the clock interrupts. For optimum accuracy in profiling, the DL-11W should be configured to interrupt at the highest possible priority level. The **kg** device driver automatically calibrates itself to the line clock frequency.

SEE ALSO

config(1), kgmon(8)

HISTORY

The **kg** driver appeared in 4.2BSD.

NAME

kloader — in-kernel bootloader

SYNOPSIS

```
options KLOADER
options KLOADER_KERNEL_PATH="\"/netbsd\""
```

DESCRIPTION

The **kloader** is the in-kernel bootloader for platforms that do not have a proper firmware.

Some platforms supported by NetBSD do not have a firmware that can boot the NetBSD kernel. Examples are game consoles (dreamcast and playstation2 ports), and handhelds (hpcarm, hpcmips, and hpcsh ports). On such platforms the bootloader is usually a host program that runs under the native OS. This means that rebooting NetBSD is a lengthy process of booting into the native OS first, launching the bootloader program, and finally booting NetBSD again. This problem is addressed by **kloader**, which allows the currently running kernel to serve as a bootloader for the kernel being booted, thus avoiding the burden of booting into the native OS first.

When **kloader** is configured into the kernel, a call to reboot(2) causes the **kloader** to load the new kernel into memory, and arrange for control to be passed to the new kernel — just like a standalone bootloader does. The new kernel then boots in the ordinary manner.

SEE ALSO

reboot(2), boot(8), reboot(8)

HISTORY

kloader first appeared in NetBSD 1.6.

BUGS

kloader ignores *howto* and *bootstr* arguments passed to the reboot(2) system call, and reboots the system with the previous boot settings.

kloader doesn't support booting compressed kernels.

The hpcarm port doesn't support **kloader** yet.

NAME

kse — Micrel 8842/8841 PCI Ethernet controller driver

SYNOPSIS

kse* **at pci? dev ? function ?**

DESCRIPTION

The **kse** driver supports Ethernet interfaces based on the Micrel 8842/8841 PCI Ethernet chips. The 8842 has 2 Ethernet ports which behave as a managed switch to bridge each other. It works like a T-shape connector of Ethernet data flow in which an Ethernet controller sits at the leg of the T. Frames can flow between the two ports while traffic destined for the 8842 reaches the EMAC. The 8841 is a plain 10/100 Ethernet. The **kse** driver distinguishs and handles them according to the HW model.

SEE ALSO

arp(4), ifmedia(4), netintro(4), pci(4), ifconfig(8)

AUTHORS

The **kse** driver was written by Tohru Nishimura.

BUGS

__STRICT_ALIGNMENT case is not written. 8842 media selection keeps "auto" and indicates "up 100baseTX-FX flow" when either of two ports is found link-up. There is no functional provision to see and control the media selection of them this moment. Advanced features like flow volume bound, VLAN tag insertion/removal, QoS DiffServ are not implemented and remain uncontrollable by the **kse** driver. UDP4CSUM is not very useful since the HW has an implementation error for the case when a large UDP datagram is fragmented into MTU sized frames.

NAME

ksyms — kernel symbol table interface

SYNOPSIS

```
pseudo-device ksyms
```

DESCRIPTION

The /dev/ksyms character device provides a read-only interface to the current kernel symbol table. It can be accessed either as a sequential file, where it looks like an executable file but with zero-sized text and data segments, or via ioctl(2).

/dev/ksyms represents the symbol table at the time when the device is opened, and may not change until it is closed.

The in-kernel symbol manager is designed to be able to handle any type of symbol table. However, only elf(5) symbol tables are currently dealt with.

IOCTLS

The ioctl(2) command codes below are defined in ⟨sys/ksyms.h⟩.

The (third) argument to the ioctl(2) should be a pointer to the type indicated.

KIOCGSIZE (int)
: Returns the total size of the current symbol table. This should be used when allocating a buffer to read in the whole symbol table to memory.

KIOCGVALUE (struct ksyms_gsymbol)
: Returns the value for the given symbol name in a symtab-independent fashion.

```
struct ksyms_gsymbol {
        const char *kg_name;
        unsigned long *kg_value;
};
```

The struct element *kg_name* should be set to the name of the requested value, and the address that *kg_value* points to will receive the symbol value.

KIOCGSYMBOL (struct ksyms_gsymbol)
: Returns the complete symbol for the given symbol name.

```
struct ksyms_gsymbol {
        const char *kg_name;
         void *kg_sym;
};
```

The struct element *kg_name* should be set to the name of the requested symbol, and the found symbol will be written to the *kg_sym* address. It is the callers responsibility to ensure that enough space for the symbol is allocated.

FILES

/dev/ksyms

SEE ALSO

ioctl(2), nlist(3), elf(5)

HISTORY

A **ksyms** device exists in many different operating systems. This implementation is modelled in function after Solaris **ksyms**. This **ksyms** driver was written by Anders Magnusson for NetBSD.

The **ksyms** driver first appeared in NetBSD 2.0.

BUGS

No LKM modules can be loaded or unloaded while `/dev/ksyms` is open.

NAME

kttcp — kernel support for testing network throughput

SYNOPSIS

```
pseudo-device kttcp
```

DESCRIPTION

This driver provides kernel support for testing network throughput from the perspective of the kernel. It is similar in spirit to the classic ttcp network benchmark program, the main difference being that with kttcp, the kernel is the source and sink of the data.

Testing like this is useful for a few reasons:

1. This allows us to know what kind of performance we can expect from network applications that run in the kernel space, such as the NFS server or the NFS client. These applications don't have to move the data to/from userspace, and so benchmark programs which run in userspace don't give us an accurate model.
2. Since data received is just thrown away, the receiver is very fast. This can provide better exercise for the sender at the other end.
3. Since the NetBSD kernel currently uses a run-to-completion scheduling model, kttcp provides a benchmark model where preemption of the benchmark program is not an issue.

SEE ALSO

`pkgsrc/benchmarks/kttcp`, `pkgsrc/benchmarks/ttcp`

NAME

kue — Kawasaki LSI KL5KUSB101B USB Ethernet driver

SYNOPSIS

kue* **at uhub?**

HARDWARE

The **kue** driver supports the following adapters:

3Com 3c19250
3Com 3c460 HomeConnect Ethernet USB Adapter
Abocom URE450
ADS Technologies USB-10BT
Aox USB101
ATen UC10T
Corega EtherUSB
D-Link DSB-650
Entrega NET-USB-E45
I/O Data USB-ET/T
Kawasaki USB101
LinkSys USB10T
Netgear EA101
Peracom USB Ethernet Adapter (3 models)
SMC 2102USB
SMC 2104USB

DESCRIPTION

The **kue** driver provides support for USB Ethernet adapters based on the Kawasaki LSI KL5KLUSB101B chipset.

The KL5KLUSB101B supports a 128-entry multicast filter, single perfect filter entry for the station address and promiscuous mode. Packets are received and transmitted over separate USB bulk transfer endpoints.

The Kawasaki adapter supports only 10Mbps half-duplex mode, hence there are no `ifmedia`(4) modes to select.

For more information on configuring this device, see `ifconfig`(8).

DIAGNOSTICS

kue%d: watchdog timeout A packet was queued for transmission and a transmit command was issued, however the device failed to acknowledge the transmission before a timeout expired.

kue%d: no memory for rx list The driver failed to allocate an mbuf for the receiver ring.

SEE ALSO

`arp`(4), `netintro`(4), `usb`(4), `ifconfig`(8)

HISTORY

The **kue** device driver first appeared in FreeBSD 4.0, and in NetBSD 1.5.

AUTHORS

The **kue** driver was written by Bill Paul ⟨wpaul@ee.columbia.edu⟩.

BUGS

The **kue** driver does not accumulate Ethernet collisions statistics because the Kawasaki firmware does not appear to maintain any internal statistics.

NAME

lasi — 2nd generation I/O subsystem

SYNOPSIS

```
lasi*   at mainbus0
lasi*   at phantomas?
gsc*    at lasi?
```

DESCRIPTION

Core bus controller and I/O subsystem as present on newer HP 9000/700 workstations and single-board computers. The supported core bus controllers are those used in conjunction with PA7100LC, PA7300LC and early PA8000 / PA8200 CPUs and based upon LSI's macrochip that includes:

- Core bus controller
- System Clock
- Interrupt Controller
- Real Time Clock Interface
- Power System support
- RAM and Flash EEPROM controllers
- Two PS/2 ports
- RS-232 and Centronics I/O ports
- i82596CA LAN coprocessor
- NCR53c710 SCSI I/O processor
- Serial Interface for the CS4216 Audio Codec
- Interface for the WD37C65C floppy drive controller
- Optional Telephone interface (two fax/voice modem channels)

MACHINES

An incomplete list of machines that use the LASI bus controller:

- 712/*
- 715/{64/80/100}[XC]
- 725/{64/80/100}
- 743/*
- 744/*
- 748/*
- A180[C]
- B132L[+], B160L, B180L+
- C100, C110, C132L, C160[L], C180, C200, C240, C360
- J200, J210[XC], J280, J282, J2240
- all [EK]-Class machines
- R380, R390
- RDI PrecisionBook
- SAIC Galaxy 1100

SEE ALSO

com(4), gsc(4), gsckbc(4), harmony(4), iee(4), intro(4), io(4), lpt(4), osiop(4), phantomas(4),

Precision I/O Architecture Reference Specification, Hewlett-Packard.

712 I/O Subsystem ERS, Revision 1.1, Hewlett-Packard, 12 February 1993, Dwg No. A-A2263-66510-31.

Hewlett-Packard Journal, Number 2, Volume 46, April 1995.

HISTORY

The **lasi** driver appeared in OpenBSD 2.4. It was ported to NetBSD 1.6 by Matthew Fredette.

NAME

lc — DEC EtherWORKS III Ethernet interfaces device driver

SYNOPSIS

```
lc0 at isa? port ? iomem ? irq ?
```

DESCRIPTION

The **lc** device driver supports DEC EtherWORKS III Ethernet interfaces, which are based on the LEMAC Ethernet chip. This includes the DE203, DE204, and DE205.

MEDIA SELECTION

The EtherWORKS III series supports various combinations of media, depending on model. Some models also support media autoselect. Media is selected with `ifconfig`(8)'s **media** directive.

SEE ALSO

`ifmedia`(4), `intro`(4), `isa`(4), `ifconfig`(8)

BUGS

The **lc** does not currently support EISA LEMAC interfaces.

NAME

lca — DECchip 21066 Core Logic chipset

SYNOPSIS

```
lca* at mainbus?
pci* at lca?
```

DESCRIPTION

The **lca** driver provides support for the DECchip 21066 Core Logic chipset.

SEE ALSO

intro(4), mainbus(4), pci(4)

NAME

ld — logical disk driver

SYNOPSIS

```
ld* at aac? unit ?
ld* at amr? unit ?
ld* at cac? unit ?
ld* at icp? unit ?
ld* at iop? tid ?
ld* at mlx? unit ?
ld* at twa? unit ?
ld* at twe? unit ?
ld* at ataraid? vendtype ? unit ?
```

DESCRIPTION

The **ld** driver provides support for simple block devices, usually presented by a disk array controller.

FILES

/dev/ld*up*	block mode disk unit *u*, partition *p*
/dev/rld*up*	raw mode disk unit *u*, partition *p*

SEE ALSO

aac(4), amr(4), cac(4), icp(4), intro(4), iop(4), mlx(4), twa(4), twe(4)

HISTORY

The **ld** driver first appeared in NetBSD 1.5.3.

BUGS

The capacity and geometry of units as accessed through the **ld** driver may be different than when accessed through some other medium (e.g. SCSI).

NAME

le — AMD 7990, 79C90, 79C960, 79C970 LANCE Ethernet interface driver

SYNOPSIS

ISA boards

```
nele0 at isa? port 0x320 irq 9 drq 7    # NE2100
le* at nele?
bicc0 at isa? port 0x320 irq 10 drq 7   # BICC Isolan
le* at bicc?
depca0 at isa? port 0x300 iomem 0xc8000 iosiz 0x8000 irq 5 # DEC DEPCA
le* at depca?
le* at isapnp?                          # ISA Plug-and-Play adapters
```

EISA boards

```
depca* at eisa? slot ?                  # DEC DE422
le* at depca?
```

MCA boards

```
le* at mca? slot ?                      # SKNET Personal/MC2+
```

PCI boards and mainboard adapters

```
le* at pci? dev? function ?
```

TURBOchannel PMAD-A or onboard (alpha, pmax)

```
le* at tc? slot ? offset ?
```

alpha

```
le* at ioasic? offset ?
```

amiga

```
le* at zbus0
```

atari

```
le0 at vme0 irq 4  # BVME410
le0 at vme0 irq 5  # Riebl/PAM
```

hp300

```
le* at dio? scode ?
```

mvme68k

```
le0 at pcc? ipl 3  # MVME147
```

news68k

```
le0 at hb0 addr 0xe0f00000 ipl 4
```

newsmips

```
le0 at hb0 addr 0xbff80000 level 1
```

pmax

```
le* at ioasic? offset ?
le* at ibus0 addr ?
```

sparc and sparc64

```
le* at sbus? slot ? offset ?
le* at ledma0 slot ? offset ?
le* at lebuffer? slot ? offset ?
```

sun3

```
le0 at obio0 addr 0x120000 ipl 3
options LANCE_REVC_BUG
```

vax

```
le0 at vsbus0 csr 0x200e0000
```

DESCRIPTION

The **le** interface provides access to a Ethernet network via the AMD Am7990 and Am79C90 (CMOS, pin-compatible) LANCE (Local Area Network Controller - Ethernet) chip set.

The **le** driver also supports PCnet-PCI cards based on the AMD 79c970 chipset, which is a single-chip implementation of a LANCE chip and PCI bus interface.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **le** interface employs the Address Resolution Protocol (ARP) described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

Selective reception of multicast Ethernet frames is provided by a 64-bit mask; multicast destination addresses are hashed to a bit entry using the Ethernet CRC function.

The use of "trailer" encapsulation to minimize copying data on input and output is supported by the interface but offers no advantage on systems with large page sizes. The use of trailers is automatically negotiated with ARP. This negotiation may be disabled, on a per-interface basis, with `ifconfig`(8).

HARDWARE

amiga

The **le** interface supports the following Zorro II expansion cards:

A2065	Commodore's Ethernet card, manufacturer 514, product 112
AMERISTAR	Ameristar's Ethernet card, manufacturer 1053, product 1
ARIADNE	Village Tronic's Ethernet card, manufacturer 2167, product 201

The A2065 and Ameristar Ethernet cards support only manual media selection.

The Ariadne card supports a software media selection for its two different connectors:

10Base2/BNC	also known as thinwire-Ethernet
10BaseT/UTP	also known as twisted pair

The Ariadne card uses an autoselect between UTP and BNC, so it uses UTP when an active UTP line is connected or otherwise BNC. See `ifmedia`(4) for media selection options for `ifconfig`(8).

ISA

The ISA-bus Ethernet cards supported by the **le** interface are:

BICC Isolan
Novell NE2100

Digital DEPCA

EISA

The EISA-bus Ethernet cards supported by the **le** interface are:

DEC DE422

MCA

The MCA-bus Ethernet cards supported by the **le** interface are:

SKNET Personal MC2
SKNET MC2+

pmax

All LANCE interfaces on DECstations are supported, as are interfaces on Alpha AXP machines with a TURBOchannel bus.

No support is provided for switching between media ports. The DECstation 3100 provides both AUI and BNC (thinwire or 10BASE2) connectors. Port selection is via a manual switch and is not software configurable.

The DECstation model 5000/200 PMAD-AA baseboard device provides only a BNC connector.

The **ioasic** baseboard devices and the PMAD-AA TURBOchannel option card provide only an AUI port.

sparc

The Sbus Ethernet cards supported by the **le** interface include:

SBE/S
: SCSI and Buffered Ethernet (sun part 501-1860)

FSBE/S
: Fast SCSI and Buffered Ethernet (sun part 501-2015)

Interfaces attached to an **ledma0** on SPARC systems typically have two types of connectors:

AUI/DIX	Standard 15 pin connector
10BaseT	UTP, also known as twisted pair

The appropriate connector can be selected by supplying a **media** parameter to ifconfig(8). The supported arguments for **media** are:

10base5/AUI	to select the AUI connector, or
10baseT/UTP	to select the UTP connector.

If a **media** parameter is not specified, a default connector is selected for use by examining all media types for carrier. The first connector on which a carrier is detected will be selected. Additionally, if carrier is dropped on a port, the driver will switch between the possible ports until one with carrier is found.

DIAGNOSTICS

le%d: overflow More packets came in from the Ethernet than there was space in the receive buffers. Packets were missed.

le%d: receive buffer error Ran out of buffer space, packet dropped.

le%d: lost carrier The Ethernet carrier disappeared during an attempt to transmit. It will finish transmitting the current packet, but will not automatically retry transmission if there is a collision.

le%d: excessive collisions, tdr %d Ethernet extremely busy or jammed, outbound packets dropped after 16 attempts to retransmit.

TDR is "Time Domain Reflectometry". The LANCE TDR value is an internal counter of the interval between the start of a transmission and the occurrence of a collision. This value can be used to determine the distance from the Ethernet tap to the point on the Ethernet cable that is shorted or open (unterminated).

le%d: dropping chained buffer Packet didn't fit into a single receive buffer, packet dropped. Since the **le** driver allocates buffers large enough to receive the maximum size Ethernet packet, this means some other station on the LAN transmitted a packet larger than allowed by the Ethernet standard.

le%d: transmit buffer error LANCE ran out of buffer before finishing the transmission of a packet. If this error occurs, the driver software has a bug.

le%d: underflow LANCE ran out of buffer before finishing the transmission of a packet. If this error occurs, the driver software has a bug.

le%d: controller failed to initialize Driver failed to start the AM7990 LANCE. This is potentially a hardware failure.

le%d: memory error RAM failed to respond within the timeout when the LANCE wanted to read or write it. This is potentially a hardware failure.

le%d: receiver disabled The LANCE receiver was turned off due to an error.

le%d: transmitter disabled The LANCE transmitter was turned off due to an error.

SEE ALSO

arp(4), ifmedia(4), inet(4), intro(4), mca(4), ifconfig(8)

Am79C90 - CMOS Local Area Network Controller for Ethernet, 17881, May 1994, Advanced Micro Devices.

HISTORY

The pmax **le** driver is derived from a **le** driver that first appeared in 4.4BSD. Support for multiple bus attachments first appeared in NetBSD 1.2.

The Amiga **le** interface first appeared in NetBSD 1.0

The Ariadne Ethernet card first appeared with the Amiga ae interface in NetBSD 1.1 and was converted to the Amiga **le** interface in NetBSD 1.3

BUGS

The Am7990 Revision C chips have a bug which causes garbage to be inserted in front of the received packet occasionally. The work-around is to ignore packets with an invalid destination address (garbage will usually not match), by double-checking the destination address of every packet in the driver. This work-around is enabled with the LANCE_REVC_BUG kernel option.

When LANCE_REVC_BUG is enabled, the **le** driver executes one or two calls to an inline Ethernet address comparison function for every received packet. On the mc68000 it is exactly eight instructions of 16 bits each. There is one comparison for each unicast packet, and two comparisons for each broadcast packet.

In summary, the cost of the LANCE_REVC_BUG option is:

1. loss of multicast support, and
2. eight extra CPU instructions per received packet, sometimes sixteen, depending on both the processor, and the type of packet.

All sun3 systems are presumed to have this bad revision of the Am7990, until proven otherwise. Alas, the only way to prove what revision of the chip is in a particular system is inspection of the date code on the chip package, to compare against a list of what chip revisions were fabricated between which dates.

Alas, the Am7990 chip is so old that AMD has "de-archived" the production information about it; pending a search elsewhere, we don’t know how to identify the revision C chip from the date codes.

On all pmax front-ends, performance is impaired by hardware which forces a software copy of packets to and from DMA buffers. The **ioasic** machines and the DECstation 3100 must copy packets to and from non-contiguous DMA buffers. The DECstation 5000/200 and the PMAD-AA must copy to and from an onboard SRAM DMA buffer. The CPU overhead is noticeable, but all machines can sustain full 10 Mb/s media speed.

NAME

leds — sun2 diagnostic Light Emitting Diodes driver

SYNOPSIS

```
#include <machine/leds.h>
```

DESCRIPTION

All sun2 machines are equipped a diagnostic display of eight Light Emitting Diodes (LEDs), located on either the front (deskside chassis) or the back (desktop machine) of the system unit.

The kernel changes the display during periods of idle processor activity according to a stored sequential pattern list. The `/dev/leds` interface provides a way of manipulating the pattern list via simple file I/O.

The structure of the file is as follows:

```
struct led_patterns {
        u_char divisor;
        u_char patlen;
        u_char pat[256];
};
```

divisor The number of idle periods to wait before switching to the next pattern in the array.

patlen The number of patterns stored in the array.

pat The array of patterns to display.

When a clock interrupt occurs while the processor is idle, a pattern countdown timer is decremented. When the countdown timer reaches zero it is reset with the **divisor** value and the next pattern in the array is selected and displayed.

Each 8-bit pattern describes the state of the diagnostic LEDs. A set bit in a pattern indicates that its corresponding LED should be extinguished, while a reset bit indicates an LED to be illuminated.

FILES

`/dev/leds`

EXAMPLES

The following example uses `awk`(1) to display the repeating animation of a single lit LED scrolling from one end of the display to the other, using six clock ticks between each update.

```
# echo 5 8 254 253 251 247 239 223 191 127 |
  awk '{ for (i=1;i≤NF;i++) printf("%c",$i+0); }' > /dev/leds
```

ERRORS

An I/O transfer to `/dev/leds` will complete successfully unless:

`[EIO]` A read or write starting beyond the end of the file was attempted.

SEE ALSO

`ppt`(6)

HISTORY

`/dev/leds` first appeared in NetBSD 1.2.

NAME

leds — sun3 diagnostic Light Emitting Diodes driver

SYNOPSIS

```
#include <machine/leds.h>
```

DESCRIPTION

With the exception of the Sun 3/80, all sun3 machines are equipped a diagnostic display of eight Light Emitting Diodes (LEDs), located on the back of the system unit. The Sun 3/80 has a single LED, which is located on the front panel.

The kernel changes the display during periods of idle processor activity according to a stored sequential pattern list. The `/dev/leds` interface provides a way of manipulating the pattern list via simple file I/O.

The structure of the file is as follows:

```
struct led_patterns {
        u_char divisor;
        u_char patlen;
        u_char pat[256];
};
```

divisor The number of idle periods to wait before switching to the next pattern in the array.

patlen The number of patterns stored in the array.

pat The array of patterns to display.

When a clock interrupt occurs while the processor is idle, a pattern countdown timer is decremented. When the countdown timer reaches zero it is reset with the **divisor** value and the next pattern in the array is selected and displayed.

Each 8-bit pattern describes the state of the diagnostic LEDs. With the exception of the 3/80, a set bit in a pattern indicates that its corresponding LED should be extinguished, while a reset bit indicates an LED to be illuminated. On the 3/80 the polarity of the bits is reversed and only the lowest order bit is used.

FILES

`/dev/leds`

EXAMPLES

The following example uses `awk`(1) to display the repeating animation of a single lit LED scrolling from one end of the display to the other, using six clock ticks between each update.

```
# echo 5 8 254 253 251 247 239 223 191 127 |
  awk '{ for (i=1;i≤NF;i++) printf("%c",$i+0); }' > /dev/leds
```

ERRORS

An I/O transfer to `/dev/leds` will complete successfully unless:

`[EIO]` A read or write starting beyond the end of the file was attempted.

SEE ALSO

`ppt`(6)

HISTORY

`/dev/leds` first appeared in NetBSD 1.2.

NAME

light — SGI Light/Entry/Starter (LG1/LG2) graphics controller

SYNOPSIS

```
light* at gio? slot ?
wsdisplay* at light? console ?
```

DESCRIPTION

The **light** driver supports the SGI Light series of graphics controllers, also known as Entry or Starter graphics and designated LG1 or LG2. These controllers have a fixed 1024x768 resolution with 8-bit colour depth. Unlike most SGI offerings, both 13w3 and VGA video cables are supported. **light** is found in Indigo and Crimson machines.

SEE ALSO

gio(4), grtwo(4), newport(4), wscons(4)

HISTORY

The **light** driver first appeared in NetBSD 5.0.

AUTHORS

Stephen M. Rumble wrote this driver.

BUGS

Much is unknown about the REX chipset. Therefore, the driver relies on the PROM to properly initialise the graphics.

This driver will not run without modification on Crimson machines.

NAME

lii — Attansic/Atheros L2 Fast-Ethernet device driver

SYNOPSIS

lii* **at pci? dev ? function ?**

DESCRIPTION

The **lii** provides support for the Attansic/Atheros Fast-Ethernet card. This card is found in a variety of low-end Asus hardware, notably the Asus EeePC.

SEE ALSO

mii(4), ukphy(4)

HISTORY

The **lii** driver appeared in NetBSD 5.0.

AUTHORS

Quentin Garnier ⟨cube@NetBSD.org⟩

NAME

lkm — Loadable Kernel Modules interface

SYNOPSIS

```
options LKM
```

DESCRIPTION

Loadable kernel modules allow the system administrator to dynamically add and remove functionality from a running system. This ability also helps software developers to develop new parts of the kernel without constantly rebooting to test their changes.

Various types of modules can be loaded into the system. There are several defined module types, listed below, which can be added to the system in a predefined way. In addition, there is a generic type, for which the module itself handles loading and unloading.

The **lkm** interface is used by performing `ioctl`(2) calls on the `/dev/lkm` device. Normally all operations involving Loadable Kernel Modules are handled by the `modload`(8), `modunload`(8), and `modstat`(8) programs. Users should never have to interact with `/dev/lkm` directly.

MODULE TYPES

System Call modules

System calls may be replaced by loading new ones via the **lkm** interface. All system calls may be replaced, but special care should be taken with the `ioctl`(2) system call, as it is used to load and unload modules.

When a system call module is unloaded, the system call which was replaced by the loadable module is returned to its rightful place in the system call table by LKM code.

Virtual File System modules

Virtual file systems may be added via the **lkm** interface.

Device Driver modules

New block and character device drivers may be loaded into the system with `options LKM`. A problem with loading a device driver is that the driver's device nodes must exist for the devices to be accessed. They are usually created by instructing `modload`(8) to run an appropriate program when the driver has been successfully loaded.

Emulation modules

Emulation modules allow to load an emulation support for foreign operating systems.

Execution Interpreters

Execution Interpreters allow to load support for execution of new type of binaries, not normally supported by kernel. This also allows to load support for executing foreign system binaries. Execution Interpreters normally depend on Emulation modules, in that appropriate Emulation module has to be loaded before Execution Interpreter can be.

Miscellaneous modules

Miscellaneous modules are modules for which there are not currently well-defined or well-used interfaces for extension. They are provided for extension, and the user is expected to write their own loader to handle the kernel pointer/table manipulation to "wire in" their loaded module (and "unwire" it on unload). An example of a "miscellaneous module" might be a loader for card-specific VGA drivers or alternate terminal emulations in an appropriately layered console driver.

NOTES

Security considerations

Loaded kernel modules can do anything with kernel structures. There is no memory protection between modules and the rest of the kernel. Hence, a potential attacker controlling `/dev/lkm` can do anything they want with the system.

To avoid associated security risks, new LKMs cannot be loaded when `securelevel` is higher than zero.

Module might crash system

Loading and using a buggy module is likely to crash operating system - since the module becomes part of kernel, a code error is much more fatal than for userland programs. If you are doing kernel development, this would hopefully end up happening less frequently than changing, recompiling, installing, and rebooting would normally occur. This should speed development considerably for a lot of the in-kernel work that is currently taking place.

FILES

`/dev/lkm`	**lkm** interface device
`/usr/include/sys/lkm.h`	file containing definitions of module types
`lkm/*`	subdirectory `lkm` within kernel source tree contains many LKMs which are suitable as a base for new ones

SEE ALSO

`modload`(8), `modstat`(8), `modunload`(8)

HISTORY

The **lkm** facility was designed to be similar in functionality to the loadable kernel modules facility provided by SunOS 4.1.3.

AUTHORS

Terrence R. Lambert ⟨terry@cs.weber.edu⟩

NAME

lm — National Semiconductor LM78, LM79 and compatible hardware monitors

SYNOPSIS

```
lm0 at isa? port 0x280
lm1 at isa? port 0x290
lm2 at isa? port 0x310
lm3 at isa? port 0xa00
lm0 at pnpbios0 index ?
```

DESCRIPTION

The **lm** driver provides support for the National Semiconductor LM series hardware monitors and register compatible chips to be used with the envsys(4) API.

Most supported devices possess 11 sensors:

Sensor	Units	Typical Use
IN0	uV DC	Core voltage
IN1	uV DC	unknown
IN2	uV DC	+3.3V
IN3	uV DC	+5V
IN4	uV DC	+12V
IN5	uV DC	-12V
IN6	uV DC	-5V
Temp	uK	Motherboard Temperature
Fan0	RPM	Fan
Fan1	RPM	Chassis Fan
Fan2	RPM	Fan

for some devices (most Winbond devices) sensor names and numbers will be different.

Due to hardware limitations, fresh sensor data is only available every 2 seconds.

HARDWARE

Chips supported by the **lm** driver include:

National Semiconductor *LM78*, *LM78-J*, *LM79* and *LM81*.

Winbond *W83627HF*, *W83627THF*, *W83627EHF*, *W83627DHG*, *W83637HF*, *W83697HF*, *W83781D*, *W83782D*, *W83783S*, *W83791D*, *W83791SD* and *W83792D*.

ASUS *AS99127F*.

SEE ALSO

envsys(4), envstat(8)

HISTORY

The **lm** device appeared in NetBSD 1.5.

BUGS

Interrupt support is unimplemented.

There are currently no known pnpbios IDs assigned to LM chips.

NAME

lmc — device driver for LMC (and some SBE) wide-area network interface cards

SYNOPSIS

This driver is built into the GENERIC kernel so it should "just work". The driver can be loaded into a running kernel with `modload`(8).

```
modload if_lmc.o
```

The kernel must be built with support for Loadable Kernel Modules, `lkm`(8).

```
options     LKM
```

securelevel must be 0 to load modules; see `init`(8).

The driver can be built into a kernel by adding the following to `/sys/arch/ARCH/conf/YOURKERNEL`:

```
lmc*        at pci?
options     ALTQ
options     ALTQ_HFSC # for altq example
pseudo-device sppp
pseudo-device bpfilter
```

The driver can send and receive raw IP packets even if SPPP is not configured into the kernel.

DESCRIPTION

This is an open-source Unix device driver for PCI-bus wide-area network interface cards. It sends and receives packets in HDLC frames over synchronous circuits. A computer plus UNIX plus some LMC cards makes an *open* wide-area network router.

The **lmc** driver works with FreeBSD, NetBSD, OpenBSD, BSD/OS, and Linux OSs. It has been tested on i386 (SMP 32-bit little-end), PowerPC (32-bit big-end), Alpha (64-bit little-end), and Sparc (64-bit big-end) architectures.

The **lmc** driver works with the following cards:

LMC5200
HSSI—High Speed Serial Interface,
EIA612/613, 50-pin connector,
0 to 52 Mb/s, DTE only.

LMC5245
T3, 2xBNC conns, 75 ohm
C-Parity or M13 Framing,
DSX-3 up to 910 ft.

LMC1000
SSI—Synchronous Serial Interface,
V.35, X.21, EIA449, EIA530(A), EIA232,
0 to 10 Mb/s, DTE or DCE.

LMC1200
T1/E1, RJ45 conn, 100 or 120 ohms,
T1-B8ZS-ESF, T1-AMI-SF, E1-HDB3-many,
DSX-1 up to 1500 ft; CSU up to 6 Kft.

LMC cards contain a high-performance **PCI** interface, an **HDLC** function and either integrated **modems** (T1, T3) or **modem** interfaces (HSSI and SSI).

PCI The PCI interface is a DEC 21140A Tulip Fast Ethernet chip. This chip has an efficient PCI implementation with scatter/gather DMA, and can run at 100 Mb/s full duplex (twice as fast as needed here).

HDLC The HDLC functions (ISO-3309: flags, bit-stuffing, CRC) are implemented in a Field Programmable Gate Array (FPGA) which talks to the Ethernet chip through a Media Independent Interface (MII). The hardware in the FPGA translates between Ethernet packets and HDLC frames on-the-fly; think of it as a WAN PHY chip for Ethernet.

Modem
The modem chips are the main differences between cards. HSSI cards use ECL10K chips to implement the EIA-612/613 interface. T3 cards use a TranSwitch TXC-03401 framer chip. SSI cards use Linear Technology LTC1343 modem interface chips. T1 cards use a BrookTree/Conexant/Mindspeed Bt8370 framer and line interface chip.

Line protocol stacks exist above device drivers and below internet protocol stacks. They typically encapsulate packets in HDLC frames and deal with higher-level issues like protocol multiplexing and security. The driver is compatible with several line protocol stacks:

SPPP `sppp`(4) implements Synchronous-PPP and Cisco-HDLC in the kernel.

RawIP The null line protocol, built into the driver, sends and receives raw IPv4 and IPv6 packets in HDLC frames with no extra bytes of overhead and no state at the end points.

EXAMPLES

ifconfig and lmcconfig

The program `lmcconfig`(8) manipulates interface parameters beyond the scope of `ifconfig`(8). **`lmcconfig`** has many flags and options, but in normal operation only a few are needed.

```
lmcconfig lmc0
```
displays interface configuration and status.

```
lmcconfig lmc0 -X 1
```
selects the built-in RawIP line protocol stack.

```
lmcconfig lmc0 -X 2 -x 2
```
selects the SPPP stack and the PPP protocol.

Some configuration options are available through **`ifconfig`** as well as **`lmcconfig`**.

```
ifconfig -m lmc0
```
lists the available media options.

```
ifconfig lmc0 mediaopt loopback
```
loops the interface transmitter to the receiver for testing. This loopback uses a path present in every card type. **`lmcconfig`** can select card-specific loopbacks, such as outbound payload loopback.

```
ifconfig lmc0 debug
```
enables debugging output from the device driver and from the line protocol stack above it.

```
lmcconfig lmc0 -D
```
enables debugging output from the device driver.

Debugging messages that appear on the console are also written to file `/var/log/messages`. *Caution*: when things go very wrong, a torrent of debugging messages can swamp the console and bring a machine to its knees.

Operation

Configure a PPP link using SPPP with

```
lmcconfig lmc0 -X 2 -x 2
ifconfig lmc0 10.0.0.1 10.0.0.2
```

Configure a Cisco-HDLC link using SPPP with

```
lmcconfig lmc0 -X 2 -x 3
ifconfig lmc0 10.0.0.1 10.0.0.2
```

Configure a RAWIP link with

```
lmcconfig lmc0 -X 1
ifconfig lmc0 10.0.0.1 10.0.0.2
```

TESTING

Testing with Loopbacks

Testing with loopbacks requires only one card and can test everything on that card. Packets can be looped back at many points: in the PCI chip, in the modem chips, through a loopback plug, in the local external equipment, or at the far end of a circuit.

All cards can be looped through the PCI chip. Cards with internal modems can be looped through the modem framer and the modem line interface. Cards for external modems can be looped through the driver/receiver chips. See `lmcconfig`(8) for details.

Configure the card with

```
ifconfig lmc0 10.0.0.1 10.0.0.1
```

HSSI Loopback plugs can be ordered from SBE (and others). Transmit clock is normally supplied by the external modem. When an HSSI card is operated with a loopback plug, the PCI bus clock must be used as the transmit clock, typically 33 MHz. When testing an HSSI card with a loopback plug, configure it with

```
lmcconfig lmc0 -a 2
```

"**`-a 2`**" selects the PCI bus clock as the transmit clock.

T3 Connect the two BNC jacks with a short coax cable.

SSI Loopback plugs can be ordered from SBE (only). Transmit clock is normally supplied by the external modem. When an SSI card is operated with a loopback plug, the on-board clock synthesizer must be used. When testing an SSI card with a loopback plug, configure it with

```
lmcconfig lmc0 -E -f 10000000
```

"**`-E`**" puts the card in DCE mode to source a transmit clock. "**`-f 10000000`**" sets the internal clock source to 10 Mb/s.

T1/E1 A loopback plug is a modular plug with two wires connecting pin 1 to pin 4 and pin 2 to pin 5.

One can also test by connecting to a local modem (HSSI and SSI) or NI (T1 and T3) configured to loop back. Cards can generate signals to loopback remote equipment so that complete circuits can be tested; see `lmcconfig`(8) for details.

Testing with a Modem

Testing with a modem requires two cards of different types. The cards can be in the same machine or different machines.

Configure the two cards with

```
ifconfig lmc0 10.0.0.1 10.0.0.2
ifconfig lmc1 10.0.0.2 10.0.0.1
```

T3/HSSI If you have a T3 modem with an HSSI interface (made by Digital Link, Larscom, Kentrox etc.) then use an HSSI card and a T3 card. The coax cables between the card and the modem must "cross over" (see below).

T1/V.35 If you have a T1 (or E1) modem with a V.35, X.21 or EIA530 interface, then use an SSI card and a T1 card. Use a T1 null modem cable (see below) between the external modem and the T1 card.

Testing with a Null Modem Cable

Testing with a null modem cable requires two cards of the same type. The cards can be in the same machine or different machines.

Configure the two cards with

```
ifconfig lmc0 10.0.0.1 10.0.0.2
ifconfig lmc1 10.0.0.2 10.0.0.1
```

HSSI Three-meter HSSI null-modem cables can be ordered from SBE. In a pinch, a 50-pin SCSI-II cable up to a few meters will work as a straight HSSI cable (not a null modem cable). Longer cables should be purpose-built HSSI cables because the cable impedance is different. Transmit clock is normally supplied by the external modem. When an HSSI card is connected by a null modem cable, the PCI bus clock can be used as the transmit clock, typically 33 MHz. When testing an HSSI card with a null modem cable, configure it with

```
lmcconfig lmc0 -a 2
```

"`-a 2`" selects the PCI bus clock as the transmit clock.

T3 T3 null modem cables are just 75-ohm coax cables with BNC connectors. TX OUT on one card should be connected to RX IN on the other card. In a pinch, 50-ohm thin Ethernet cables *usually* work up to a few meters, but they will *not* work for longer runs—75-ohm coax is *required*.

SSI Three-meter SSI null modem cables can be ordered from SBE. An SSI null modem cable reports a cable type of V.36/EIA449. Transmit clock is normally supplied by the external modem. When an SSI card is connected by a null modem cable, an on-board clock synthesizer is used. When testing an SSI card with a null modem cable, configure it with

```
lmcconfig lmc0 -E -f 10000000
```

"`-E`" puts the card in DCE mode to source a transmit clock. "`-f 10000000`" sets the internal clock source to 10 Mb/s.

T1/E1 A T1 null modem cable has two twisted pairs that connect pins 1 and 2 on one plug to pins 4 and 5 on the other plug. Looking into the cable entry hole of a plug, with the locking tab oriented down, pin 1 is on the left. A twisted pair Ethernet cable makes an excellent straight T1 cable. Alas, Ethernet cross-over cables do not work as T1 null modem cables.

OPERATING NOTES

LEDs

HSSI and SSI cards should be operational if all three green LEDs are on (the upper-left one should be blinking) and the red LED is off.

RED	upper-right	No Transmit clock
GREEN	upper-left	Device driver is alive if blinking
GREEN	lower-right	Modem signals are good
GREEN	lower-left	Cable is plugged in (SSI only)

T1/E1 and T3 cards should be operational if the upper-left green LED is blinking and all other LEDs are off. For the T3 card, if other LEDs are on or blinking, try swapping the coax cables!

RED	upper-right	Received signal is wrong
GREEN	upper-left	Device driver is alive if blinking

BLUE	lower-right	Alarm Information Signal (AIS)
YELLOW	lower-left	Remote Alarm Indication (RAI)

RED	blinks if an outward loopback is active.
GREEN	blinks if the device driver is alive.
BLUE	blinks if sending AIS, on solid if receiving AIS.
YELLOW	blinks if sending RAI, on solid if receiving RAI.

Packet Lengths

Maximum transmit and receive packet length is unlimited.
Minimum transmit and receive packet length is one byte.

Cleaning up after one packet and setting up for the next packet involves making several DMA references. This can take longer than the duration of a short packet, causing the adapter to fall behind. For typical PCI bus traffic levels and memory system latencies, back-to-back packets longer than about 20 bytes will always work (53 byte cells work), but a burst of several hundred back-to-back packets shorter than 20 bytes will cause packets to be dropped. This usually is not a problem since an IPv4 packet header is at least 20 bytes long.

The device driver imposes no constraints on packet size. Most operating systems set the default Maximum Transmission Unit (MTU) to 1500 bytes; the legal range is usually (72..65535). This can be changed with

```
ifconfig lmc0 mtu 2000
```

SPPP enforces an MTU of 1500 bytes for PPP and Cisco-HDLC. RAWIP sets the default MTU to 4032 bytes, but allows it to be changed to anything.

ALTQ: Alternate Output Queue Disciplines

The driver has hooks for `altq`(9), the Alternate Queueing package. To see ALTQ in action, use your favorite traffic generation program to generate three flows sending down one T3 circuit. Without ALTQ, the speeds of the three connections will vary chaotically. Enable ALTQ and two of the connections will run at about 20 Mb/s and the third will run at about 2 Mb/s.

Enable `altqd`(8) and add the following lines to `/etc/altq.conf`:

```
interface lmc0 bandwidth 44M hfsc
class hfsc lmc0 a root pshare 48
filter lmc0 a 10.0.0.2 12345 10.0.0.1 0 6
filter lmc0 a 10.0.0.1 0 10.0.0.2 12345 6
class hfsc lmc0 b root pshare 48
filter lmc0 b 10.0.0.2 12346 10.0.0.1 0 6
filter lmc0 b 10.0.0.1 0 10.0.0.2 12346 6
class hfsc lmc0 c root pshare 4 default
filter lmc0 c 10.0.0.2 12347 10.0.0.1 0 6
filter lmc0 c 10.0.0.1 0 10.0.0.2 12347 6
```

The example above requires the `altq`(4) Hierarchical Fair Service Curve queue discipline to be configured in `conf/YOURKERNEL`:

```
options ALTQ
options ALTQ_HFSC
```

.

BPF: Berkeley Packet Filter

The driver has hooks for `bpf`(4), the Berkeley Packet Filter, a protocol-independent raw interface to data link layers.

To test the BPF kernel interface, bring up a link between two machines, then run `ping`(8) and `tcpdump`(1):

```
ping 10.0.0.1
```

and in a different window:

```
tcpdump -i lmc0
```

The output from tcpdump should look like this:

```
03:54:35.979965 10.0.0.2 > 10.0.0.1: icmp: echo request
03:54:35.981423 10.0.0.1 > 10.0.0.2: icmp: echo reply
```

Line protocol control packets may appear among the ping packets occasionally.

The kernel must be configured with

```
options bpfilter
```

SNMP: Simple Network Management Protocol

The driver is aware of what is required to be a Network Interface Object managed by an Agent of the Simple Network Management Protocol. The driver exports SNMP-formatted configuration and status information sufficient for an SNMP Agent to create MIBs for:

RFC-2233 *Interfaces group*
RFC-2496 *DS3 interfaces*
RFC-2495 *DS1/E1 interfaces*
RFC-1659 *RS232-like interfaces*

An SNMP Agent is a user program, not a kernel function. Agents can retrieve configuration and status information by using `ioctl`(2) system calls. User programs should poll *sc->cfg.ticks* which increments once per second after the SNMP state has been updated.

E1 Framing

Phone companies usually insist that customers put a *Frame Alignment Signal* (FAS) in time slot 0. A Cyclic Redundancy Checksum (CRC) can also ride in time slot 0. *Channel Associated Signalling* (CAS) uses Time Slot 16. In telco-speak *signalling* is on/off hook, ringing, busy, etc. Signalling is not needed here and consumes 64 Kb/s. Only use E1-CAS formats if the other end insists on it! Use E1-FAS+CRC framing format on a public circuit. Depending on the equipment installed in a private circuit, it may be possible to use all 32 time slots for data (E1-NONE).

T3 Framing

M13 is a technique for multiplexing 28 T1s into a T3. Muxes use the C-bits for speed-matching the tributaries. Muxing is not needed here and usurps the FEBE and FEAC bits. Only use T3-M13 format if the other end insists on it! Use T3-CParity framing format if possible. Loop Timing, Fractional T3, and HDLC packets in the Facility Data Link are *not* supported.

T1 & T3 Frame Overhead Functions

Performance Report Messages (PRMs) are enabled in T1-ESF.
Bit Oriented Protocol (BOP) messages are enabled in T1-ESF.
In-band loopback control (framed or not) is enabled in T1-SF.
Far End Alarm and Control (FEAC) msgs are enabled in T3-CPar.
Far End Block Error (FEBE) reports are enabled in T3-CPar.
Remote Alarm Indication (RAI) is enabled in T3-Any.
Loopbacks initiated remotely time out after 300 seconds.

T1/E1 'Fractional' 64 kb/s Time Slots

T1 uses time slots 24..1; E1 uses time slots 31..0. E1 uses TS0 for FAS overhead and TS16 for CAS overhead. E1-NONE has *no* overhead, so all 32 TSs are available for data. Enable/disable time slots by setting 32 1s/0s in a config param. Enabling an E1 overhead time slot, or enabling TS0 or TS25-TS31 for T1, is ignored by the driver, which knows better. The default TS param, 0xFFFFFFFF, enables the maximum number of time slots for whatever frame format is selected. 56 Kb/s time slots are *not* supported.

SEE ALSO

`tcpdump`(1), `ioctl`(2), `bpf`(4), `de`(4), `sppp`(4), `altq.conf`(5), `altqd`(8), `ifconfig`(8), `init`(8), `lkm`(8), `lmcconfig`(8), `modload`(8), `ping`(8), `tcpdump`(8), `altq`(9), `ifnet`(9)

`http://www.sbei.net/`

HISTORY

Ron Crane had the idea to use a Fast Ethernet chip as a PCI interface and add an Ethernet-to-HDLC gate array to make a WAN card. David Boggs designed the Ethernet-to-HDLC gate array and PC cards. We did this at our company, LAN Media Corporation (LMC). SBE Corporation aquired LMC and continues to make the cards.

Since the cards use Tulip Ethernet chips, we started with Matt Thomas' ubiquitous `de`(4) driver. Michael Graff stripped out the Ethernet stuff and added HSSI stuff. Basil Gunn ported it to Solaris (lost) and Rob Braun ported it to Linux. Andrew Stanley-Jones added support for three more cards. David Boggs rewrote everything and now feels responsible for it.

AUTHORS

David Boggs ⟨boggs@boggs.palo-alto.ca.us⟩

NAME

lms — Logitech-style bus mouse driver

SYNOPSIS

```
lms0 at isa? port 0x23c irq 5
lms1 at isa? port 0x238 irq 5
wsmouse* at lms?
```

DESCRIPTION

This driver provides an interface to a Logitech-style bus mouse. Mouse related data are accessed by wsmouse(4) devices.

SEE ALSO

mms(4), pms(4), wsmouse(4)

NAME

lmtemp — National Semiconductor LM75, LM77 and compatible hardware monitors

SYNOPSIS

```
lmtemp0 at iic? addr 0x48 flags 0x0000
```

DESCRIPTION

The **lmtemp** driver provides support for the National Semiconductor LM on iicbus series temperatures and register compatible chips to be used with the envsys(4) API. The **lmtemp** supports ranges of the temperature between +125 from -55.

Each device is specified by the value of the address and flags.

Temperature	addr	flags
LM75	0x48 - 0x4f	0x0000
DS75	0x48 - 0x4f	0x0001
LM77	0x48 - 0x4b	0x0002

HARDWARE

Chips supported by the **lmtemp** driver include:

National Semiconductor *LM75*

National Semiconductor *LM77*

Dallas Semiconductor *DS75*

SEE ALSO

envsys(4), envstat(8)

HISTORY

The **lmtemp** device appeared in NetBSD 4.0.

BUGS

Interrupt support is unimplemented.

NAME

lo — software loopback network interface

SYNOPSIS

```
pseudo-device loop
```

DESCRIPTION

The **loop** interface is a software loopback mechanism which may be used for performance analysis, software testing, and/or local communication. As with other network interfaces, the loopback interface must have network addresses assigned for each address family with which it is to be used. These addresses may be set or changed with the SIOCSIFADDR `ioctl`(2). The loopback interface should be the last interface configured, as protocols may use the order of configuration as an indication of priority. The loopback should *never* be configured first unless no hardware interfaces exist.

The loopback interface **lo0** is created at boottime, it always exists and cannot be destroyed with `ifconfig`(8). Additional loopback interfaces can be created by using the `ifconfig`(8) **create** command.

DIAGNOSTICS

lo%d: can't handle af%d . The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

SEE ALSO

`inet`(4), `intro`(4), `ifconfig`(8)

HISTORY

The **lo** device appeared in 4.2BSD.

BUGS

Previous versions of the system enabled the loopback interface automatically, using a nonstandard Internet address (127.1). Use of that address is now discouraged; a reserved host address for the local network should be used instead.

NAME

lp — parallel line printer

SYNOPSIS

```
lp0 at uba0 csr 0177514 vector lpintr
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **lp** device supports DEC and DEC compatible printers on the LP-11 parallel interface.

The unit number of the printer is specified by the minor device after removing the low 3 bits, which act as per-device parameters. Currently only the lowest of the low three bits is interpreted: if it is set, the device is assumed to have a 64-character set or half-ASCII mode, rather than a full 96-character set.

If the 64-character set is assumed, any lower case characters are mapped to upper case; left curly and right curly braces are mapped to left and right parentheses over laid with a hyphen; grave accents are mapped to acute accents with overlaid with a hyphen; the pipe bar character is mapped to an exclamation sign overlaid with a hyphen; and the tilde character is mapped to a carat overlaid with a hyphen.

The default page width is 132 columns; longer lines are truncated. This may be overridden by specifying, for example, `flags 256`.

FILES

`/dev/lp`

SEE ALSO

`lpr`(1)

HISTORY

A **lp** driver appeared in Version 6 AT&T UNIX.

NAME

lpt — generic printer device driver

SYNOPSIS

```
lpt* at ppbus?
options LPT_DEBUG
options LPT_VERBOSE
```

DESCRIPTION

The **lpt** driver is the port of the lpt driver to the ppbus(4) system.

The purpose of this driver is to allow parallel port sharing with other parallel devices. See ppbus(4) for more information about the parallel port bus system.

The parallel port bus is reserved by **lpt** when the printer device is opened and released when the device is closed.

Ports can be configured to use interrupts, DMA, IEEE negotiations, and IEEE compliant transfer modes by using the lptctl(8) command, with modes depending on the hardware available.

FILES

/dev/lpt?	parallel port device
/dev/lpt?ctl	parallel port control device

SEE ALSO

atppc(4), ppbus(4), lptctl(8), mknod(8)

HISTORY

This driver is derived from the FreeBSD ppbus and also from the historic NetBSD drivers.

The **lpt** driver used to assign special meaning for some bits of device minor. This was dropped in ppbus-based **lpt** in favour of configuration via lptctl(8).

AUTHORS

This manual page was based on the FreeBSD **lpt** manual page. The information has been updated for the NetBSD port by Gary Thorpe.

NAME

lpt — parallel port driver

SYNOPSIS

```
lpt* at pioc? offset 0x0278 irq 0
lpt* at pioc? offset 0x0378 irq -1
lpt* at pioc? offset 0x03bc irq -1
```

DESCRIPTION

This driver provides access to parallel ports. The highest bit in the minor number selects whether to do polling or wait for interrupts. If no IRQ is specified in the kernel configuration, only the polling device may be used.

FILES

`/dev/lpt0`	first interrupt-driven parallel port device
`/dev/lpa0`	first polled parallel port device

NAME

lpt — Parallel port driver

SYNOPSIS

```
lpt0 at isa? port "IO_LPT1" irq 7
lpt1 at isa? port "IO_LPT2"
lpt* at acpi?
lpt* at ofisa?
lpt* at pnpbios? index ?
lpt* at puc? port ?
```

DESCRIPTION

This driver provides access to parallel ports. The bits in the minor number select various features of the driver. If no IRQ is specified in the kernel configuration, only the polling device may be used.

Minor Bit	Function
128	Use the interruptless driver. (polling)
64	Do not initialize the device on the port.
32	Automatic LF on CR.

FILES

`/dev/lpt0`	first interrupt-driven parallel port device
`/dev/lpa0`	first polled parallel port device

SEE ALSO

`acpi`(4), `isa`(4), `ofisa`(4), `pnpbios`(4), `puc`(4)

NAME

lpt — parallel port driver

SYNOPSIS

```
lpt0 at pcc? ipl 1
lpt0 at pcctwo? ipl 1
```

DESCRIPTION

This driver provides access to parallel ports. The bits in the minor number select various features of the driver.

Minor Bit	Function
128	Use the interruptless driver. (polling)
64	Do not initialize the device on the port.
32	Automatic LF on CR.
16	Select 1.6uS strobe pulse width (default is 6.4uS)

FILES

`/dev/lpt0`	interrupt-driven parallel port device
`/dev/lpa0`	polled parallel port device

SEE ALSO

`pcc`(4), `pcctwo`(4)

NAME

lxtphy — Driver for Level One LXT-970 10/100 Ethernet PHYs

SYNOPSIS

lxtphy* **at mii? phy ?**

DESCRIPTION

The **lxtphy** driver supports the Level One LXT-970, LXT-970A and LXT-971 10/100 Ethernet PHYs. These PHYs are found on a variety of high-performance Ethernet interfaces.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

m25p — STMicroelectronics M25P family of SPI-connected flash devices

SYNOPSIS

```
m25p* at spi0 slave 0
```

DESCRIPTION

The **m25p** driver provides support for STMicrolelectronics' M25P family of SPI connected NOR flash devices.

SEE ALSO

spi(4)

HISTORY

The machine-independent **m25p** driver was written by Garrett D'Amore for the Champaign-Urbana Community Wireless Network Project (CUWiN), and appeared in NetBSD 4.0.

BUGS

Some important bits that are vital for use as a mass storage device are still missing. Specifically, there is no API to erase the device, and there is no support for device partitioning.

NAME

mace — Multimedia, Audio and Communications Engine I/O ASIC

SYNOPSIS

```
mace0 at mainbus0 addr 0x1f000000
```

DESCRIPTION

The **mace** I/O ASIC takes care of all the basic I/O functions. Examples of interfaces provided by it are the keyboard and mouse, fast Ethernet, video, audio, the PCI bus, and parallel and serial connectors. It is connected to the host bus through the crime(4) controller. **mace** is typically found on O2 machines.

SEE ALSO

crime(4), pci(4)

HISTORY

The **mace** driver first appeared in NetBSD 1.5.

NAME

magma — Magma Sp Serial/Parallel board device driver

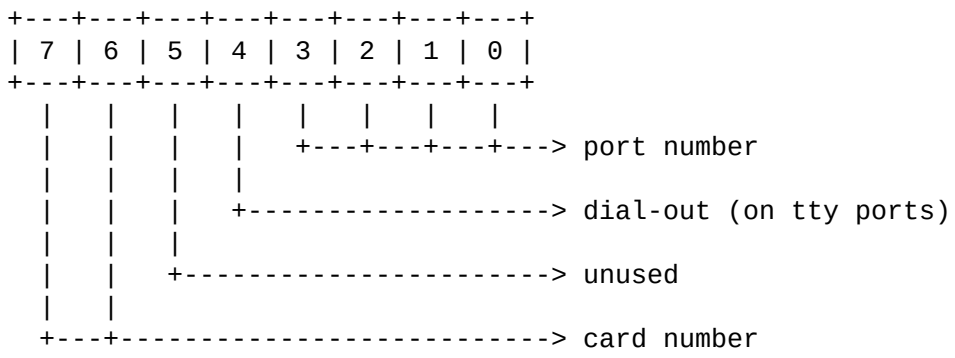
SYNOPSIS

```
magma* at sbus? slot ? offset ?
mtty* at magma?
mbpp* at magma?
```

DESCRIPTION

The **magma** driver provides an interface to Magma LC2+1Sp, 2+1Sp, 4+1Sp, 8+2Sp, 4Sp, 8Sp, 12Sp, 16Sp, 1P and 2P boards. These boards are based around the Cirrus Logic CD1400 serial/parallel communications engine and the Cirrus Logic CD1190 parallel communications engine.

The device minor numbers for this driver are encoded as follows:



Up to four cards are supported in the system.

All tty ports have full automatic hardware (RTS/CTS) flow control available and a 12 byte FIFO on the chip in each direction so errors should be minimal.

FILES

`/dev/tty[0-3][0-a]` Serial ports
`/dev/bpp[0-3][0-1]` Parallel ports

DIAGNOSTICS

mtty%d%x: ring buffer overflow Incoming characters have been discarded due to a buffer overflow. This is caused by the process in control of the device not reading characters fast enough.

If need be you can make the ring buffer bigger by changing the `MAGMA_RBUF_SIZE` #define to something bigger, but it should be a multiple of two.

mtty%d%x: fifo overflow Incoming characters have been discarded due to a CD1400 channel overrun. This is caused by interrupts not being serviced sufficiently quickly to prevent the 12 byte receive FIFO on a serial channel from overflowing.

Reducing the value of either the `MTTY_RX_FIFO_THRESHOLD` or `MTTY_RX_DTR_THRESHOLD` #define's to something smaller may help slow machines avoid this problem.

SEE ALSO

`read`(2), `termios`(4), `tty`(4)

HISTORY

The driver was loosely based upon the `cy`(4) Cyclades Cyclom device driver written by Andrew Herbert and Timo Rossi.

AUTHORS

The driver was written by Iain Hibbert.

TO DO

CD1190 parallel support.

"bpp" input.

Dial-out (cua) devices are not yet supported.

"mdmbuf" is unsupported (see `tty`(4) and `termios`(4)).

Automatic XON/XOFF handshaking could be implemented fairly easily.

It would be good if the tty port waited for the FIFO to empty before allowing a close, so that I could turn off the channel interrupts at that time. It can be done.

NAME

mainbus — pseudo top level bus

SYNOPSIS

```
mainbus0 at root
XX       at mainbus0
```

DESCRIPTION

The **mainbus** is not a real bus, but serves as the top level device to which other busses and drivers attach.

SEE ALSO

config(1), autoconf(4), intro(4)

NAME

mainbus — RiscPC main I/O bus device

SYNOPSIS

```
mainbus0 at root
```

DESCRIPTION

The **mainbus** interface provides access to the RiscPC main I/O bus. It is only used by the autoconfiguration system to location devices attached to the mainbus.

SEE ALSO

podulebus(4)

NAME

mainbus — Processor bus support

SYNOPSIS

```
cpc0 at mainbus0
cs0  at mainbus0 addr 0x7fe00000
rtc0 at mainbus0 addr 0x7ff00000
```

DESCRIPTION

The **mainbus** provides support for the processor bus on the Artesyn PM/PPC.

SEE ALSO

cpc(4), cs(4), rtc(4)

HISTORY

The **mainbus** driver appeared in NetBSD 2.0.

NAME

mainbus — Mac68k main processor bus device

SYNOPSIS

```
mainbus0 at root
```

DESCRIPTION

The **mainbus** interface serves as an abstraction used by the autoconfiguration system to help find and attach busses (e.g. the NuBus expansion bus) or devices (e.g. the math coprocessor) connected to the Macintosh main processor bus.

DIAGNOSTICS

mainbus0 (root). This is the normal autoconfiguration message indicating that the main processor bus has been found.

SEE ALSO

autoconf(4), obio(4)

HISTORY

The **mainbus** first appeared in NetBSD 1.2.

NAME

mainbus — MVME68K main processor bus device

SYNOPSIS

```
mainbus0 at root
```

DESCRIPTION

The **mainbus** interface serves as an abstraction used by the autoconfiguration system to help find and attach busses (e.g. the VME bus) or devices (e.g. the PCC chip) connected to Motorola MVME single board computers' main processor bus.

DIAGNOSTICS

mainbus0 (root). This is the normal autoconfiguration message indicating that the main processor bus has been found.

SEE ALSO

autoconf(4), pcc(4), pcctwo(4)

NAME

makphy — Driver for Marvell Semiconductor 88E1000 “Alaska” 10/100/1000 Ethernet PHY

SYNOPSIS

```
makphy* at mii? phy ?
```

DESCRIPTION

The **makphy** driver supports the Marvell Semiconductor 88E1000 10/100/1000 Ethernet PHY. These PHYs are found on a variety of CAT5 Gigabit Ethernet interfaces.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

maple — Maple bus driver

SYNOPSIS

```
maple0 at shb?
```

DESCRIPTION

The **maple** driver provides support for controlling the Maple bus (controller ports).

IOCTLS

The following ioctl(2) calls apply to Maple bus devices.

MAPLEIO_GDEVINFO *struct maple_devinfo*
: Read, from the kernel, the device information of the device.

```
struct maple_devinfo {
        uint32_t di_func;
        uint32_t di_function_data[3];
        uint8_t di_area_code;
        uint8_t di_connector_direction;
        char di_product_name[30];
        char di_product_license[60];
        uint16_t di_standby_power;
        uint16_t di_max_power;
};
```

FILES

/dev/maple*Ps*
: Maple bus device at port *P* (A-D), subunit *s* (null for base device, 1-5 for expansion device)

SEE ALSO

mkbd(4), mlcd(4), mmem(4), mms(4)

HISTORY

The **maple** device driver appeared in NetBSD 1.6.

NAME

mavb — Moosehead A/V Board audio device

SYNOPSIS

```
mavb0 at mace0 offset 0x300000 intr 6
audio* at audiobus?
```

DESCRIPTION

The **mavb** driver provides support for the Moosehead A/V Board found on SGI O2 machines.

The Moosehead A/V Board uses an AD1843 codec that supports 8- and 16-bit audio sample recording and playback at rates from 5 to 54 kHz, with 1 Hz resolution. The **mavb** driver also makes it possible to control the playback volume by using the buttons on the front of the O2.

SEE ALSO

audio(4), intro(4), mace(4)

HISTORY

The **mavb** driver first appeared in OpenBSD 3.7. It was later ported to NetBSD 5.0.

AUTHORS

The **mavb** driver was written by Mark Kettenis, and ported to NetBSD by Jared D. McNeill.

CAVEATS

The analog mixer in the AD1843 codec does not provide a master volume control. Therefore, the O2 volume buttons only control the output volume of the DAC.

BUGS

Currently only sample rates up to 48 kHz are supported. Recording is not implemented yet. The second DAC on the AD1843 codec sits idle.

NAME

mbe — Fujitsu MB8696x-based PCMCIA and SEGA LAN (HIT-0300) Ethernet driver

SYNOPSIS

PCMCIA

```
mbe* at pcmcia? function ?
```

dreamcast

```
mbe* at g2bus? function ?
```

DESCRIPTION

The **mbe** driver supports PCMCIA Ethernet adapters based on the Fujitsu MB86960A/MB86965A Ethernet controller. Supported PCMCIA cards include:

Fujitsu MBH10302
Fujitsu MBH10304
TDK Corp. LAK-CD021BX
TDK Corp. DFL9610
Fujitsu Towa LA501

It also supports the SEGA LAN (HIT-0300) Ethernet adapter available on the dreamcast.

SEE ALSO

`ifmedia`(4), `intro`(4), `pcmcia`(4), `ifconfig`(8)

HISTORY

The **mbe** driver appeared in NetBSD 1.4.

NAME

mc — Ethernet driver for Am79C940 (MACE) onboard Ethernet

SYNOPSIS

mc* **at obio?**

DESCRIPTION

The **mc** interface provides access to a 10 Mb/s Ethernet network via the AMD Am79C940 (MACE) Ethernet chip set.

HARDWARE

The **mc** interface supports the on-board Ethernet of the following mac68k machines:

- Centris 660AV
- Quadra 660AV
- Quadra 840AV

The **mc** interface supports the on-board Ethernet of the following macppc machines:

- Apple Network Server (500 and 700)
- Apple Power Macintosh (7300, 7500, 7600, 8500, 8600, 9500, and 9600)
- Power Computing (PowerCenter, PowerCenter Pro, PowerCurve, PowerTower, PowerTower Pro, and PowerWave)

DIAGNOSTICS

mc0 at obio0: address %s. This is a normal autoconfiguration message noting the 6 byte physical Ethernet address of the adapter.

SEE ALSO

ae(4), bm(4), gem(4), sn(4), tlp(4), ifconfig(8)

HISTORY

The **mc** interface first appeared in NetBSD 1.3.

AUTHORS

The **mc** Ethernet driver was written by Dave Huang ⟨khym@bga.com⟩. The man page was written by Thomas Klausner ⟨wiz@NetBSD.org⟩ and Michael Wolfson ⟨mbw@NetBSD.org⟩.

NAME

mca — introduction to machine-independent MCA bus support and drivers

SYNOPSIS

```
mca0 at mainbus?

options MCAVERBOSE
```

Machine-dependent; depends on the bus topology and MCA bus interface of your system. Typical MCA buses are connected directly to the main system bus.

DESCRIPTION

NetBSD includes a machine-independent MCA bus subsystem and several machine-independent MCA device drivers.

HARDWARE

NetBSD includes machine-independent MCA drivers, sorted by device type and driver name:

SCSI controllers

aha	Adaptec AHA-1640 SCSI interface
esp	NCR 53C90 SCSI Adapter

Disk and tape controllers

edc	IBM ESDI Fixed Disk Controller

Serial interfaces

com	NS8250-, NS16450-, and NS16550-based serial cards.

Network interfaces

tr	TROPIC based token ring interfaces
ate	Allied-Telesis 1720 Ethernet interface cards
we	WD/SMC WD80x3x Ethernet interface cards and clones
le	SKNET Personal and MC+ Ethernet interface cards
elmc	3Com EtherLink/MC (3c523) Ethernet interface
ep	3Com EtherLink III 3c529 Ethernet interface
tra	Tiara LANCard/E and Standard MicroSystems 3016/MC Ethernet interface

SEE ALSO

aha(4), ate(4), com(4), edc(4), elmc(4), ep(4), esp(4), le(4), ne(4), tr(4), tra(4), we(4)

HISTORY

The machine-independent MCA subsystem appeared in NetBSD 1.5.

NAME

mcbus — MCBUS system bus

SYNOPSIS

```
mcbus* at mainbus0
mcmem* at mcbus? mid ?
```

DESCRIPTION

The **mcbus** driver provides support for MCBUS system bus found on AlphaServer 4100 systems.

The following devices are supported by the **mcbus** driver:

mcmem	memory modules attached to the MCBUS main system bus.
mcpcia	MCPCIA MCBUS-to-PCI bus adapter

The following devices are NOT supported by the **mcbus** driver:

i2c	i2c bus

SEE ALSO

intro(4), mainbus(4), mcpcia(4)

BUGS

Only one MCBUS bus can exist within the system.

NAME

mcclock — DS1287 real-time clock

SYNOPSIS

algor

```
mcclock* at isa? port 0x70
```

alpha

```
mcclock* at gbus? offset ?
mcclock* at ioasic? offset ?
mcclock* at isa? port 0x70
mcclock* at jensenio? port ?
```

arc

```
mcclock* at jazzio?
```

evbmips

```
mcclock* at isa? port 0x70
```

pmax

```
mcclock* at ibus0 addr ?
mcclock* at ioasic? offset ?
```

sgimips

```
mcclock* at mace0 offset 0x3a0000
```

DESCRIPTION

The **mcclock** driver provides support for the DS1287 real-time clock (RTC). Note that the kernel expects the RTC to run in UTC.

SEE ALSO

intro(4), ioasic(4), isa(4)

NAME

mcd — Mitsumi CD-ROM driver

SYNOPSIS

```
mcd0 at isa? port 0x300 irq 10
options MCD_PROMISC
```

DESCRIPTION

The **mcd** driver provides support for Mitsumi CD-ROM controller and drive on the isa(4) bus.

FILES

/dev/cd[0-9][a-h]	block mode Mitsumi CD-ROM devices
/dev/rmcd[0-9][a-h]	raw mode Mitsumi CD-ROM devices

SEE ALSO

intro(4), isa(4), ne(4), we(4)

BUGS

The **mcd** hardware is difficult to probe accurately. Historically, the **mcd** probe would accept any return values as indicating that an **mcd** drive was present. Other devices, particularly ne(4) or we(4), would then be incorrectly claimed by the **mcd** driver. The driver now only accepts result codes known to indicate Mitsumi-compatible CD controllers, but may reject some **mcd** hardware which returns other result codes.

options MCD_PROMISC enables the original promiscuous probe behaviour. Use with extreme caution.

NAME

mcpcia — MCPCIA MCBUS-to-PCI bus adapter

SYNOPSIS

```
mcpcia* at mcbus? mid ?
pci* at mcpcia?
```

DESCRIPTION

The **mcpcia** driver provides support for the MCPCIA MCBUS-to-PCI bus adapter found on AlphaServer 4100 systems.

SEE ALSO

intro(4), mcbus(4), pci(4)

NAME

md — memory disk driver

SYNOPSIS

pseudo-device md [*count*]

DESCRIPTION

The **md** driver enables use of system or user memory as a disk. Memory for the disk must be allocated within the kernel or with `mdconfig`(8) before the **md** device may be used as a disk. Its behaviour is configured by the kernel options **MEMORY_DISK_HOOKS**, **MEMORY_DISK_IS_ROOT**, **MEMORY_DISK_ROOT_SIZE**, **MEMORY_DISK_DYNAMIC**, and **MEMORY_DISK_SERVER**.

FILES

`/dev/md??` block mode memory disk devices.
`/dev/rmd??` raw mode memory disk devices.

SEE ALSO

`options`(4), `mdconfig`(8), `mdsetimage`(8)

NAME

mec — MACE MAC-110 Ethernet driver

SYNOPSIS

```
mec0 at mace0 offset 0x280000 intr 3
```

DESCRIPTION

The **mec** driver provides support for the MACE MAC-110 Ethernet interface found on SGI O2 machines.

SEE ALSO

arp(4), ifmedia(4), intro(4), mace(4), mii(4), netintro(4), ifconfig(8)

HISTORY

The **mec** driver first appeared in NetBSD 2.0.

AUTHORS

The **mec** driver was written by Christopher SEKIYA and
Izumi Tsutsui.

NAME

mem, **kmem** — memory files

DESCRIPTION

The special file **/dev/mem** is an interface to the physical memory of the computer. Byte offsets in this file are interpreted as physical memory addresses. Reading and writing this file is equivalent to reading and writing memory itself. Only offsets within the bounds of **/dev/mem** are allowed.

Kernel virtual memory is accessed through the interface **/dev/kmem** in the same manner as **/dev/mem**. Only kernel virtual addresses that are currently mapped to memory are allowed.

On Acorn RiscPC, A7000(+) and RC7500 systems the I/O memory space (podule space and SuperIO) begins at physical address 0x03000000 and runs to 0x033fffff. On the RiscPC, EASI space runs from 0x08000000 to 0x0fffffff. The per-process data size for the current process is UPAGES long, and starts at VM_MAXUSER_ADDRESS.

FILES

/dev/mem
/dev/kmem

HISTORY

The **mem**, **kmem** files appeared in Version 6 AT&T UNIX.

NAME

mem, **kmem** — main memory

DESCRIPTION

The file **/dev/mem** is an interface to the physical memory of the computer. Byte offsets in this file are interpreted as physical memory addresses. Reading and writing this file is equivalent to reading and writing memory itself. An error will be returned if an attempt is made to reference an offset outside of **/dev/mem**.

Kernel virtual memory is accessed via the file **/dev/kmem** in the same manner as **/dev/mem**. Only kernel virtual addresses that are currently mapped to memory are allowed.

On the Amiga, physical memory may be discontiguous; kernel virtual memory begins at *0x00000000*.

FILES

/dev/mem
/dev/kmem

HISTORY

The files **mem** and **kmem** appeared in Version 6 AT&T UNIX.

NAME

mem, **kmem** — main memory

DESCRIPTION

The file **/dev/mem** is an interface to the physical memory of the computer. Byte offsets in this file are interpreted as physical memory addresses. Reading and writing this file is equivalent to reading and writing memory itself. An error will be returned if an attempt is made to reference an offset outside of **/dev/mem**.

Kernel virtual memory is accessed via the file **/dev/kmem** in the same manner as **/dev/mem**. Only kernel virtual addresses that are currently mapped to memory are allowed.

HP300

On the HP300, the last byte of physical memory is always `0xFFFFFFFF`. Therefore, on an HP300 with 8MB of memory, physical memory would start at `0xFF800000`. On the HP300, kernel virtual memory runs from `0` to about `0x2400000`.

FILES

`/dev/mem`
`/dev/kmem`

HISTORY

The files **mem** and **kmem** appeared in Version 6 AT&T UNIX.

NAME

mem, **kmem** — memory files and memory controller

SYNOPSIS

mem* **at mainbus0**

DESCRIPTION

The **mem** driver controls and restricts access to the systems memory by the hardware buses and the processor.

It also provides an interface to userland through the special files /dev/mem and /dev/kmem. Physical memory is accessed through /dev/mem, while kernel virtual memory is accessed through /dev/kmem. Access to kernel virtual addresses not currently mapped to memory will fail. On hp700, the physical memory range is always contiguous and starts at address 0; kernel virtual memory begins at address 0 as well.

The writeability of the /dev/mem and /dev/kmem special files are controlled by the system securelevel in addition to the filesystem permissions.

FILES

/dev/mem
/dev/kmem

HISTORY

The **mem** driver originates from OpenBSD. It was ported to NetBSD 1.6 by Matthew Fredette.

BUGS

On some systems featuring a “Viper” memory controller, NetBSD may not configure bus arbitration correctly, causing the boot process to freeze during either **mem** or cpu(4) device probe.

NAME

mem, **kmem** — memory files

DESCRIPTION

The special file **/dev/mem** is an interface to the physical memory of the computer. Byte offsets in this file are interpreted as physical memory addresses. Reading and writing this file is equivalent to reading and writing memory itself. Only offsets within the bounds of **/dev/mem** are allowed.

Kernel virtual memory is accessed through the interface **/dev/kmem** in the same manner as **/dev/mem**. Only kernel virtual addresses that are currently mapped to memory are allowed.

On ISA the I/O memory space begins at physical address 0x000a0000 and runs to 0x00100000. The per-process data size for the current process is UPAGES long, and ends at virtual address 0xfe000000.

FILES

/dev/mem
/dev/kmem

HISTORY

The **mem**, **kmem** files appeared in Version 6 AT&T UNIX.

NAME

mem, **kmem** — main memory

DESCRIPTION

The file **/dev/mem** is an interface to the physical memory of the computer. Byte offsets in this file are interpreted as physical memory addresses. Reading and writing this file is equivalent to reading and writing memory itself. An error will be returned if an attempt is made to reference an offset outside of **/dev/mem**.

Kernel virtual memory is accessed via the file **/dev/kmem** in the same manner as **/dev/mem**. Only kernel virtual addresses that are currently mapped to memory are allowed.

On the Macintosh, physical memory may be discontiguous; kernel virtual memory begins at *0x00000000*.

FILES

/dev/mem
/dev/kmem

HISTORY

The files **mem** and **kmem** appeared in Version 6 AT&T UNIX.

NAME

mem, **kmem** — main memory

DESCRIPTION

The file **/dev/mem** is an interface to the physical memory of the computer. Byte offsets in this file are interpreted as physical memory addresses. Reading and writing this file is equivalent to reading and writing memory itself. An error will be returned if an attempt is made to reference an offset outside of **/dev/mem**.

Kernel virtual memory is accessed via the file **/dev/kmem** in the same manner as **/dev/mem**. Only kernel virtual addresses that are currently mapped to memory are allowed.

On the Motorola MVME series of single board computers, physical memory may be discontiguous; kernel virtual memory begins at *0x00008000*.

FILES

/dev/mem
/dev/kmem

HISTORY

The files **mem** and **kmem** appeared in Version 6 AT&T UNIX.

NAME

mem, **kmem** — Sun main memory access driver

DESCRIPTION

The file `/dev/mem` is an interface to the physical memory of the computer. Byte offsets in this file are interpreted as physical memory addresses. Reading and writing this file is equivalent to reading and writing memory itself. An error will be returned if an attempt is made to reference an offset outside of `/dev/mem`.

Kernel virtual memory is accessed via the file `/dev/kmem` in the same manner as `/dev/mem`. Only kernel virtual addresses that are currently mapped to memory are allowed.

SPARC

On the SPARC, physical memory may be discontiguous; kernel virtual memory begins at `0xf0000000`.

FILES

`/dev/mem`
`/dev/kmem`

HISTORY

The files **mem** and **kmem** appeared in Version 6 AT&T UNIX.

NAME

mem, **kmem** — main memory

DESCRIPTION

The file `/dev/mem` is an interface to the physical memory of the computer. Byte offsets in this file are interpreted as physical memory addresses. Reading and writing this file is equivalent to reading and writing memory itself. An error will be returned if an attempt is made to reference an offset outside of `/dev/mem`.

Kernel virtual memory is accessed via the file `/dev/kmem` in the same manner as `/dev/mem`. Only kernel virtual addresses that are currently mapped to memory are allowed.

On the Sun2, kernel virtual memory begins at *0x000000*.

FILES

`/dev/mem`
`/dev/kmem`

HISTORY

The files **mem** and **kmem** appeared in Version 6 AT&T UNIX.

NAME

mem, **kmem** — main memory

DESCRIPTION

The file `/dev/mem` is an interface to the physical memory of the computer. Byte offsets in this file are interpreted as physical memory addresses. Reading and writing this file is equivalent to reading and writing memory itself. An error will be returned if an attempt is made to reference an offset outside of `/dev/mem`.

Kernel virtual memory is accessed via the file `/dev/kmem` in the same manner as `/dev/mem`. Only kernel virtual addresses that are currently mapped to memory are allowed.

On the Sun3, kernel virtual memory begins at *0x0E000000*.

FILES

`/dev/mem`
`/dev/kmem`

HISTORY

The files **mem** and **kmem** appeared in Version 6 AT&T UNIX.

NAME

mem, **kmem**, **kUmem** — memory files

DESCRIPTION

The special file **/dev/mem** is an interface to the physical memory of the computer. Byte offsets in this file are interpreted as physical memory addresses. Reading and writing this file is equivalent to reading and writing memory itself. Only offsets within the bounds of **/dev/mem** are allowed.

Kernel virtual memory is accessed through the interface **/dev/kmem** in the same manner as **/dev/mem**. Only kernel virtual addresses that are currently mapped to memory are allowed.

The file **/dev/kUmem** also refers to kernel virtual memory, but may be used to access areas mapped to UNIBUS address space and other I/O areas. It forces all accesses to use word (short integer) accesses.

On the VAX-11/780, the I/O space base address is 20000000(16); on an 11/750 the I/O space addresses are of the form fxxxxx(16). On all VAX'en the per-process data size for the current process is UPAGES long and ends at the virtual address 80000000(16).

FILES

/dev/mem
/dev/kmem
/dev/kUmem

HISTORY

The **mem**, **kmem** files appeared in Version 6 AT&T UNIX. The file **kUmem** appeared in 3.0 BSD.

NAME

memc — MVME162-MVME177 Memory Controller Chip

SYNOPSIS

```
memc* at mainbus0
```

DESCRIPTION

The **memc** devices are used on MVME162, MVME167, MVME172 and MVME177 boards to manage one or more DRAM modules. Depending on the type of DRAM module fitted, the device will either be a MEMC040 device or an MCECC device. The former manages a Parity DRAM module while the latter manages an ECC DRAM module.

DIAGNOSTICS

memc0 at mainbus0. This is the normal autoconfiguration message indicating that the Memory Controller Chip has been found and attached to the main processor bus.

memc0: Correctable error on CPU read access to 0x12345678. This indicates that an MCECC memory controller detected and corrected a single bit error in one of the DRAM banks. There are a few variations on the message where "CPU" can be replaced with "Peripheral Device" or "Scrubber", and "read" can be substituted for "write". This message is followed by some more details which can help pin-point the error...

memc0: ECC Syndrome 0x23 (DRAM Bank C, Bit#16). Pin-points exactly where the error occurred.

memc0: Uncorrectable error on CPU read access to 0x12345678. Errors like this have the potential to corrupt data. As such, it is likely the system will panic very soon afterwards.

SEE ALSO

mainbus(4)

BUGS

The **memc** driver does not yet fully support the MEMC040 (Parity) version of the device.

NAME

mesh — Apple Macintosh Enhanced SCSI Hardware driver

SYNOPSIS

```
mesh* at obio? flags 0xffff
scsibus* at mesh?
```

DESCRIPTION

The **mesh** driver provides support for the SCSI host adapters found on most supported models with on-board SCSI. The Apple Network Server models use the esp(4) and esiop(4) devices for SCSI, not **mesh**.

Models supported by **mesh** are:

- Apple PowerBook (2400, 3400, G3, and G3 Series)
- Apple PowerBook (G3 Series (bronze keyboard))
- Apple Power Macintosh/Performa (4400, 54xx, 5500, 6300/160, 6360, 6400, and 6500)
- Apple Power Macintosh (7300, 7500, 7600, 8500, 8600, 9500, and 9600)
- Apple Workgroup Server 8550
- Apple Power Macintosh G3 (Beige G3)
- APS Tech (M*Power 604e/200)
- Motorola StarMax (3000, 4000, 5000, and 5500)
- Power Computing (PowerBase, PowerCenter, PowerCenter Pro, PowerCurve, PowerTower, PowerTower Pro, and PowerWave)
- UMAX (Apus 2000, Apus 3000, C500, and C600)
- UMAX (J700, S900)

Some models have a separate SCSI bus dedicated to external devices using the esp(4) device. On these systems, the **mesh** SCSI bus is operated in SCSI 'Fast' mode (10 MB/s) and the external esp(4) bus is operated at 5 MB/s. This applies only to the following models:

- Apple Power Macintosh (7300, 7500, 7600, 8500, 8600, 9500, and 9600)
- Power Computing (PowerCenter, PowerCenter Pro, PowerCurve, PowerTower, PowerTower Pro, and PowerWave)

SEE ALSO

cd(4), ch(4), esiop(4), esp(4), intro(4), obio(4), scsi(4), sd(4), ss(4), st(4), uk(4)

NAME

mfb — PMAG-A MX monochrome unaccelerated 2-D framebuffer

SYNOPSIS

```
mfb* at tc? slot ? offset ?
wsdisplay* at mfb?
```

DESCRIPTION

The **mfb** driver provides support for the PMAG-A MX monochrome framebuffer for the TURBOchannel bus. The PMAG-A is a monochrome framebuffer capable of running at a resolution of 1280-by-1024 at 72 Hz.

SEE ALSO

cfb(4), px(4), pxg(4), sfb(4), tc(4), tfb(4), wscons(4)

BUGS

NetBSD/pmax does not currently support the machine-independent wscons(4) interface and uses a machine-dependent version.

NAME

mfcs — BSC/AlfaData MultiFaceCard II/III serial communications interface

SYNOPSIS

```
mfc0 at mainbus0
mfcs0 at mfc0 unit 0
mfcs1 at mfc0 unit 1
```

DESCRIPTION

The MultiFaceCard II/III controls, among other things, a dual port EIA RS-232C (CCITT V.28) communications interface with a multiple character buffer.

Input and output for each MultiFaceCard III line may set to a maximum baud rates of 1152000. Formula for baud rate: Baud = 230400 / N with 1 < N < 65536.

Input and output for each MultiFaceCard II line may set to a maximum baud rates of 57600. Formula for baud rate: Baud = 115200 / N with 1 < N < 65536.

FILES

`/dev/ttyA?`

DIAGNOSTICS

mfcs0: fifo overflow. The four-character input “fifo” has overflowed and incoming data has been lost.

mfcs0: %d buffer overflows. The software based input ring buffer has overflowed %d times and incoming data has been lost.

SEE ALSO

`tty`(4)

HISTORY

The Amiga **mfcs** device first appeared in NetBSD 1.1

BUGS

The MultiFaceCard II/III serial ports use the level 6 interrupt and may experience fifo overflows if the LEV6_DEFER option is used.

NAME

mfi — LSI Logic & Dell MegaRAID SAS RAID controller

SYNOPSIS

mfi* **at pci? dev ? function ?**

DESCRIPTION

The **mfi** driver provides support for the MegaRAID SAS family of RAID controllers, including:

- Dell PERC 5/e, PERC 5/i, PERC 6/e, PERC 6/i
- Intel RAID Controller SRCSAS18E, SRCSAS144E
- LSI Logic MegaRAID SAS 8208ELP, MegaRAID SAS 8208XLP, MegaRAID SAS 8300XLP, MegaRAID SAS 8308ELP, MegaRAID SAS 8344ELP, MegaRAID SAS 8408E, MegaRAID SAS 8480E, MegaRAID SAS 8888ELP, MegaRAID SAS 8880EM2

These controllers support RAID 0, RAID 1, RAID 5, RAID 10, and RAID 50 using either SAS or SATA II drives.

Although the controllers are actual RAID controllers, the driver makes them look just like SCSI controllers. All RAID configuration is done through the controllers' BIOSes.

mfi supports monitoring of the logical disks in the controller through the `bioctl`(8) and `envstat`(8) commands.

EVENTS

The **mfi** driver is able to send events to `powerd`(8) if a logical drive in the controller is not online. The *state-changed* event will be sent to the `/etc/powerd/scripts/sensor_drive` script when such condition happens.

SEE ALSO

`intro`(4), `pci`(4), `scsi`(4), `sd`(4), `bioctl`(8), `envstat`(8), `powerd`(8)

HISTORY

The **mfi** driver first appeared in NetBSD 4.0.

NAME

mfp — X68K Multi-function Peripherals

SYNOPSIS

```
mfp0 at intio0 addr 0xe88000 size 0x2000 intr 64
```

DESCRIPTION

mfp drives Motorola MC68901 MFP (Multi-function Peripheral). **mfp** driver is always required to run the NetBSD/x68k kernel, because it is connected to important devices such as the display controller, and provides fundamental functions like the system clock tick and interrupt controller. Since **mfp** provides many functions, most of the jobs as a device driver is done by its child drivers such as kbd(4) and clock(4). **mfp** driver itself only provides the common way to access its registers and a few utility functions for other non-child drivers.

SEE ALSO

clock(4), intro(4), kbd(4)

BUGS

Machine-dependent part and machine-independent part should be split.

NAME

mgnsc — Magnum low level SCSI interface

SYNOPSIS

```
mgnsc0 at zbus0
```

DESCRIPTION

The Amiga architecture uses a common machine independent scsi sub-system provided in the kernel source. The machine independent drivers that use this code access the hardware through a common interface. (see scsibus(4)) This common interface interacts with a machine dependent interface, such as **mgnsc**, which then handles the hardware specific issues.

The **mgnsc** interface handles things such as DMA and interrupts as well as actually sending commands, negotiating synchronous or asynchronous transfers and handling disconnect/reconnect of SCSI targets. The hardware that **mgnsc** uses is based on the NCR53c710 SCSI chip.

DIAGNOSTICS

mgnsc%s: abort %s: dstat %02x, sstat0 %02x sbcl %02x The scsi operation %s was aborted due to error. Dstat, sstat and sbcl are registers within the NCR53c710 SCSI chip.

siop id %d reset0 The NCR53c710 SCSI chip has been reset and configure at id %d.

SIOP interrupt: %x sts %x msg %x sbcl %x The NCR53c710 SCSI chip has interrupted unexpectedly.

SIOP: SCSI Gross Error The NCR53c710 SCSI chip has indicated that it is confused.

SIOP: Parity Error The NCR53c710 SCSI chip has indicated that it has detected a parity error on the SCSI bus.

SEE ALSO

scsibus(4)

HISTORY

The **mgnsc** interface first appeared in NetBSD 1.0

NAME

mhzc — Megahertz Ethernet/modem card driver

SYNOPSIS

```
mhzc* at pcmcia? function ?
com*  at mhzc?
sm*   at mhzc?
```

DESCRIPTION

The **mhzc** device driver provides support for the Megahertz combined Ethernet and modem cards.

SEE ALSO

com(4), pcmcia(4), sm(4)

HISTORY

The **mhzc** driver appeared in NetBSD 1.5.

NAME

midi — device-independent MIDI driver layer

SYNOPSIS

```
midi* at midibus?
midi* at pcppi?
pseudo-device sequencer

#include <sys/types.h>
#include <sys/midiio.h>
```

DESCRIPTION

The **midi** driver is the machine independent layer over anything that can source or sink a MIDI data stream, whether a physical MIDI IN or MIDI OUT jack on a soundcard, cabled to some external synthesizer or input controller, an on-board programmable tone generator, or a single jack, synthesizer, or controller component within a complex USB or IEEE1394 MIDI device that has several such components and appears as several MIDI streams.

Concepts

One MIDI data stream is a unidirectional stream of MIDI messages, as could be carried over one MIDI cable in the MIDI 1.0 specification. Many MIDI messages carry a four-bit channel number, creating up to 16 MIDI channels within a single MIDI stream. There may be multiple consumers of a MIDI stream, each configured to react only to messages on specific channels; the sets of channels different consumers react to need not be disjoint. Many modern devices such as multitimbral keyboards and tone generators listen on all 16 channels, or may be viewed as collections of 16 independent consumers each listening on one channel. MIDI defines some messages that take no channel number, and apply to all consumers of the stream on which they are sent. For an inbound stream, **midi** is a promiscuous receiver, capturing all messages regardless of channel number. For an outbound stream, the writer can specify a channel number per message; there is no notion of binding the stream to one destination channel in advance.

A single **midi** device instance is the endpoint of one outbound stream, one inbound stream, or one of each. In the third case, the write and read sides are independent MIDI streams. For example, a soundcard driver may map its MIDI OUT and MIDI IN jacks to the write and read sides of a single device instance, but those jacks can be cabled to completely different pieces of gear. Information from `dmesg`(8), and a diagram of any external MIDI cabling, will help clarify the mapping.

Underlying drivers and MIDI protocol

Drivers **midi** can attach include soundcard drivers, many of which support a UART resembling Roland's MPU401 and handled by `mpu`(4), USB MIDI devices via `umidi`(4), and on-board devices that can make sounds, whether a lowly PC speaker or a Yamaha OPL. Serial port and IEEE1394 connections are currently science fiction.

The MIDI protocol permits some forms of message compression such as running status and hidden note-off. Received messages on inbound streams are always canonicalized by **midi** before presentation to higher layers. Messages for transmission are accepted by **midi** in any valid form.

Device access

Access to **midi** device instances can be through the raw device nodes, `/dev/rmidiN`, or through the sequencer, `/dev/music`.

Raw MIDI access

A `/dev/rmidiN` device supports `read`(2), `write`(2), `ioctl`(2), `select`(2)/`poll`(2) and the corresponding `kevent`(2) filters, and may be opened only when it is not already open. It may be opened in

`O_RDONLY`, `O_WRONLY`, or `O_RDWR` mode, but a later `read`(2) or `write`(2) will return −1 if the device has no associated input or output stream, respectively.

Bytes written are passed as quickly as possible to the underlying driver as complete MIDI messages; a maximum of two bytes at the end of a `write`(2) may remain buffered if they do not complete a message, until completed by a following `write`(2).

A `read`(2) will not block or return `EWOULDBLOCK` when it could immediately return any nonzero count, and MIDI messages received are available to `read`(2) as soon as they are complete, with a maximum of two received bytes remaining buffered if they do not complete a message.

As all MIDI messages are three bytes or fewer except for System Exclusive, which can have arbitrary length, these rules imply that System Exclusive messages are the only ones of which some bytes can be delivered before all are available.

System Realtime messages are passed with minimum delay in either direction, ahead of any possible buffered incomplete message. As a result, they will never interrupt any MIDI message except possibly System Exclusive.

A `read`(2) with a buffer large enough to accommodate the first complete message available will be satisfied with as many complete messages as will fit. A buffer too small for the first complete message will be filled to capacity. Therefore, an application that reads from an `rmidi` device with buffers of three bytes or larger need never parse across read boundaries to assemble a received message, except possibly in the case of a System Exclusive message. However, if the application reads through a buffering layer such as `fread`(3), this property will not be preserved.

The **`midi`** driver itself supports the `ioctl`(2) operations `FIOASYNC`, `FIONBIO`, and `FIONREAD`. Underlying devices may support others. The value returned for `FIONREAD` reflects the size in bytes of complete messages (or System Exclusive chunks) ready to read. If the `ioctl`(2) returns *n* and a `read`(2) of size *n* is issued, *n* bytes will be read, but if a `read`(2) of size $m < n$ is issued, fewer than *m* bytes may be read if *m* does not fall on a message/chunk boundary.

Raw MIDI access can be used to receive bulk dumps from synthesizers, download bulk data to them, and so on. Simple patching of one device to another can be done at the command line, as with

```
$ cat -u 0<>/dev/rmidi0 1>&0
```

which will loop all messages received on the input stream of `rmidi0` input stream back to its output stream in real time. However, an attempt to record and play back music with

```
$ cat /dev/rmidiN >foo; cat foo >/dev/rmidiN
```

will be disappointing. The file `foo` will contain all of the notes that were played, but because MIDI messages carry no explicit timing, the ‘playback’ will reproduce them all at once, as fast as they can be transmitted. To preserve timing information, the sequencer device can be used.

Active Sensing

The MIDI protocol includes a keepalive function called Active Sensing. In any receiver that has *not* received at least one Active Sense MIDI message, the feature is suppressed and no timeout applies. If at least one such message has been received, the lapse of any subsequent 300 ms interval without receipt of any message reflects loss of communication, and the receiver should silence any currently sounding notes and return to non-Active-Sensing behavior. A sender using Active Sensing generally avoids 300 ms gaps in transmission by sending Active Sense messages (which have no other effect) as needed when there is no other traffic to send in the interval. This feature can be important for MIDI, which relies on separate Note On and Note Off messages, to avoid notes stuck on indefinitely if communication is interrupted before a Note Off message arrives.

This protocol is supported in **`midi`**. An outbound stream will be kept alive by sending Active Sense messages as needed, beginning after any real traffic is sent on the stream, and continuing until the stream is closed. On an inbound stream, if any Active Sense has been received, then a process reading an `rmidi`

device will see an end-of-file indication if the input timeout elapses. The stream remains open, the driver reverts to enforcing no timeout, and the process may continue to read for more input. Subsequent receipt of an Active Sense message will re-arm the timeout. As received Active Sense messages are handled by **midi**, they are not included among messages read from the `/dev/rmidiN` device.

These rules support end-to-end Active Sensing behavior in simple cases without special action in an application. For example, in

```
$ cat -u /dev/rmidi0 >/dev/rmidi1
```

if the input stream to `rmidi0` is lost, the `cat`(1) command exits; on the `close`(2) of `rmidi1`, **midi** ceases to send Active Sense messages, and the receiving device will detect the loss and silence any outstanding notes.

Access through the sequencer

To play music using the raw MIDI API would require an application to issue many small writes with very precise timing. The sequencer device, `/dev/music`, can manage the timing of MIDI data in the kernel, to avoid such demanding real-time constraints on a user process.

The `/dev/music` device can be opened only when it is not already open. When opened, the sequencer internally opens all MIDI instances existing in the system that are not already open at their raw nodes; any attempts to open them at their raw nodes while the sequencer is open will fail. All access to the corresponding MIDI streams will then be through the sequencer.

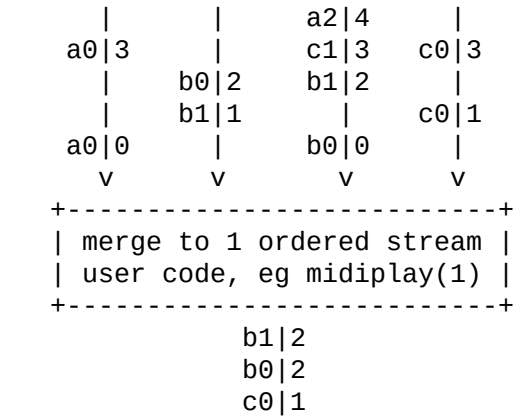
Reads and writes of `/dev/music` pass eight-byte event structures defined in ⟨`sys/midiio.h`⟩ (which see for their documentation and examples of use). Some events correspond to MIDI messages, and carry an integer *device* field to identify one of the MIDI devices opened by the sequencer. Other events carry timing information interpreted or generated by the sequencer itself.

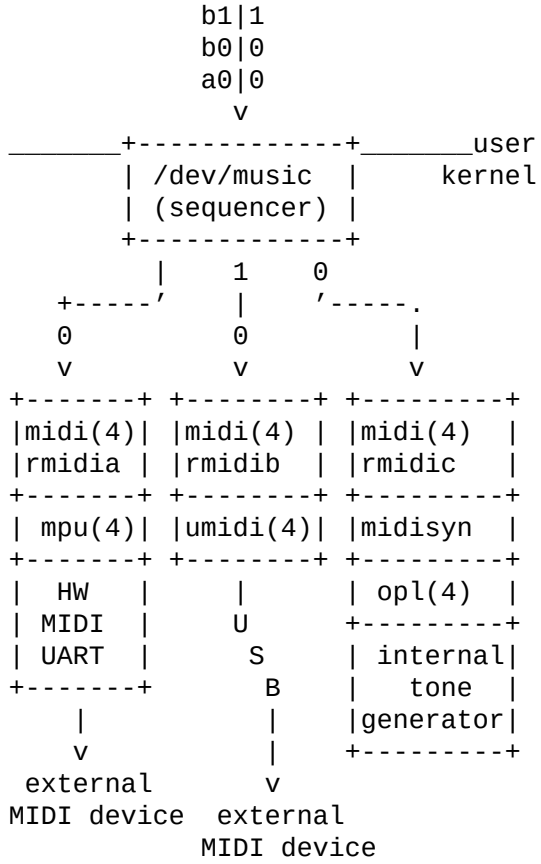
A message received on an input stream is wrapped in a sequencer event along with the device index of the stream it arrived on, and queued for the reader of `/dev/music`. If a measurable time interval passed since the last preceding message, a timing event that represents a delay for that interval is queued ahead of the received event. The sequencer handles output events by interpreting any timing event, and routing any MIDI message event at the proper time to an underlying output stream according to its *device* index. Therefore

```
$ cat /dev/music >foo; cat foo >/dev/music
```

can be expected to capture and reproduce an input performance including timing.

The process of playing back a complex MIDI file is illustrated below. The file may contain several tracks—four, in this example—of MIDI events, each marked with a device index and a time stamp, that may overlap in time. In the example, *a*, *b*, and *c* are device indices of the three output MIDI streams; the left-hand digit in each input event represents a MIDI channel on the selected stream, and the right-hand digit represents a time for the event's occurrence. As illustrated, the input tracks are not firmly associated with output streams; any track may contain events for any stream.





A user process must merge the tracks into a single stream of sequencer MIDI and timing events in order by desired timing. The sequencer obeys the timing events and distributes the MIDI events to the three destinations, in this case two external devices connected to a sound card UART and a USB interface, and an OPL tone generator on a sound card.

NOTES

Use of `select`(2)/`poll`(2) with the sequencer is supported, however, there is no guarantee that a `write`(2) will not block or return `EWOULDBLOCK` if it begins with a timer-wait event, even if `select`(2)/`poll`(2) reported the sequencer writable.

The delivery of a realtime message ahead of buffered bytes of an incomplete message may cause the realtime message to seem, in a saved byte stream, to have arrived up to 640 us earlier than it really did, at MIDI 1.0 data rates. Higher data rates make the effect less significant.

Another sequencer device, `/dev/sequencer`, is provided only for backward compatibility with an obsolete OSS interface in which some sequencer events were four-byte records. It is not further documented here, and the `/dev/music` API should be used in new code. The `/dev/sequencer` emulation is implemented only for writing, and that might not be complete.

IMPLEMENTATION NOTES

Some hardware devices supporting **`midi`** lack transmit-ready interrupts, and some have the capability in hardware but currently lack driver support. They can be recognized by the annotation `(CPU-intensive output)` in `dmesg`(8). While suitable for music playback, they may have an objectionable impact on system responsiveness during bulk transmission such as patch downloads, and are best avoided for that purpose if other suitable devices are present.

Buffer space in **midi** itself is adequate for about 200 ms of traffic at MIDI 1.0 data rates, per stream.

Event counters record bytes and messages discarded because of protocol errors or buffer overruns, and can be viewed with `vmstat -e`. They can be useful in diagnosing flaky cables and other communication problems.

A raw sound generator uses the **midisyn** layer to present a MIDI message-driven interface attachable by **midi**.

While **midi** accepts messages for transmission in any valid mixture of compressed or canonical form, they are always presented to an underlying driver in the form it prefers. Drivers for simple UART-like devices register their preference for a compressed byte stream, while those like `umidi`(4), which uses a packet protocol, or **midisyn**, which interprets complete messages, register for intact canonical messages. This design eliminates the need for compression and canonicalization logic from all layers above and below **midi** itself.

FILES

`/dev/rmidiN`
`/dev/music`
`/dev/sequencer`

ERRORS

In addition to other errors documented for the `write`(2) family of system calls, `EPROTO` can be returned if the bytes to be written on a raw **midi** device would violate MIDI protocol.

SEE ALSO

`midiplay`(1), `ioctl`(2), `ossaudio`(3), `audio`(4), `mpu`(4), `opl`(4), `umidi`(4)
For ports using the ISA bus: `cms`(4), `pcppi`(4), `sb`(4)
For ports using the PCI bus: `autri`(4), `clcs`(4), `eap`(4)

HISTORY

The **midi** driver first appeared in NetBSD 1.4. It was overhauled and this manual page rewritten for NetBSD 4.0.

BUGS

Some OSS sequencer events and `ioctl`(2) operations are unimplemented, as ⟨`sys/midiio.h`⟩ notes.

OSS source-compatible sequencer macros should be added to ⟨`sys/soundcard.h`⟩, implemented with the NetBSD ones in ⟨`sys/midiio.h`⟩, so sources written for OSS can be easily compiled.

The sequencer blocks (or returns `EWOULDBLOCK`) only when its buffer physically fills, which can represent an arbitrary latency because of buffered timing events. As a result, interrupting a process writing the sequencer may not interrupt music playback for a considerable time. The sequencer could enforce a reasonable latency bound by examining timing events as they are enqueued and blocking appropriately.

`FIOASYNC` enables signal delivery to the calling process only; `FIOSETOWN` is not supported.

The sequencer can only be a timing master, but does not send timing messages to synchronize any slave device; it cannot be slaved to timing messages received on any interface (which would presumably require a PLL algorithm similar to NTP's, and expertise in that area to implement it). The sequencer ignores timing messages received on any interface and does not pass them along to the reading process, and the OSS operations to change that behavior are unimplemented.

The `SEQUENCER_TMR_TIMEBASE` `ioctl`(2) will report successfully setting any timebase up to ridiculously high resolutions, though the actual resolution, and therefore jitter, is constrained by `hz`(9). Comparable sequencer implementations typically allow a selection from available sources of time interrupts that may be programmable.

The device number in a sequencer event is treated on `write`(2) as index into the array of MIDI devices the sequencer has opened, but on `read`(2) as the unit number of the source MIDI device; these are usually the same if the sequencer has opened all the MIDI devices (that is, none was already open at its raw node when the sequencer was opened), but might not be the same otherwise.

There is at present no way to make reception nonpromiscuous, should anyone have a reason to want to.

There should be ways to override default Active Sense behavior. As one obvious case, if an application is seen to send Active Sense explicitly, **midi** should refrain from adding its own. On receive, there should be an option to pass Active Sense through rather than interpreting it, for apps that wish to handle or ignore it themselves and never see `EOF`.

When a **midi** stream is open by the sequencer, Active Sense messages received on the stream are passed to the sequencer and not interpreted by **midi**. The sequencer at present neither does anything itself with Active Sense messages received, nor supports the OSS API for making them available to the user process.

System Exclusive messages can be received by reading a raw device, but not by reading the sequencer; they are discarded on receipt when the stream is open by the sequencer, rather than being presented as the OSS-defined sequencer events.

midisyn is too rudimentary at present to get satisfactory results from any onboard synth. It lacks the required special interpretation of the General MIDI percussion channel in GM mode. More devices should be supported; some sound cards with synthesis capability have NetBSD drivers that implement the `audio`(4) but not the **midisyn** interface. Voice stealing algorithm does not follow the General MIDI Developer Guidelines.

ALSA sequencer compatibility is lacking, but becoming important to applications. It would require the function of merging multiple tracks into a single ordered stream to be moved from user space into the sequencer. Assuming the sequencer driven by periodic interrupts, timing wheels could be used as in `hardclock`(9) itself. Similar functionality will be in OSS4; with the right infrastructure it should be possible to support both. When merging MIDI streams, a notion of transaction is needed to group critical message sequences. If ALSA or OSS4 have no such notion, it should be provided as an upward-compatible extension.

I would rather have `open`(2) itself return an error (by the POSIX description `ENODEV` looks most appropriate) if a read or write mode is requested that is not supported by the instance, rather than letting `open`(2) succeed and `read`(2) or `write`(2) return −1, but so help me, the latter seems the more common UNIX practice.

NAME

mii — IEEE 802.3 Media Independent Interface network bus

SYNOPSIS

```
acphy*    at mii? phy ?       # Altima AC101 10/100 PHY
amhphy*   at mii? phy ?       # AMD 79c901 PHY (10BASE-T part)
bmtphy*   at mii? phy ?       # Broadcom BCM5201/5202 PHYs
brgphy*   at mii? phy ?       # Broadcom BCM5400/5401 Gig-E PHYs
ciphy*    at mii? phy ?       # Cicada CS8201 Gig-E PHYs
dmphy*    at mii? phy ?       # Davicom DM9101 PHYs
exphy*    at mii? phy ?       # 3Com internal PHYs
gentbi*   at mii? phy ?       # Generic ten-bit 1000BASE-X PHYs
glxtphy*  at mii? phy ?       # Level One LXT-1000 Gig-E PHYs
gphyter*  at mii? phy ?       # NatSemi DP83861 Gig-E PHYs
icsphy*   at mii? phy ?       # Integrated Circuit Systems ICS1890 PHYs
ikphy*    at mii? phy ?       # Intel 82563 PHYs
inphy*    at mii? phy ?       # Intel 82555 PHYs
iophy*    at mii? phy ?       # Intel 82553 PHYs
lxtphy*   at mii? phy ?       # Level One LXT-970 PHYs
makphy*   at mii? phy ?       # Marvel 88E1000 Gig-E PHYs
nsphy*    at mii? phy ?       # NatSemi DP83840 PHYs
nsphyter* at mii? phy ?       # NatSemi DP83843/DP83815 PHYs
pnaphy*   at mii? phy ?       # Generic HomePNA PHYs
qsphy*    at mii? phy ?       # Quality Semiconductor QS6612 PHYs
rgephy*   at mii? phy ?       # Realtek 8169S/8110S internal PHYs
rlphy*    at mii? phy ?       # Realtek 8139/8201L PHYs
sqphy*    at mii? phy ?       # Seeq 80220/80221/80223/80225 PHYs
tlphy*    at mii? phy ?       # ThunderLAN internal PHYs
tqphy*    at mii? phy ?       # TSC Semiconductor 78Q2120 PHYs
ukphy*    at mii? phy ?       # Generic/unknown PHYs
urlphy*   at mii? phy ?       # Realtek RTL8150L internal PHYs

options MIIVERBOSE
```

DESCRIPTION

Media Independent Interface is an IEEE standard serial bus for connecting MACs (network controllers) to PHYs (physical media interfaces). The **mii** layer allows network device drivers to share support code for various PHY models, and allows unused support for PHYs which are not present in a system to be removed from the kernel.

Network device drivers which use the **mii** layer carry the “mii” autoconfiguration attribute. This allows kernel configuration files to simply specify PHYs as described above in the synopsis.

The following is an example of the messages displayed when a network interface with an attached PHY is detected by the kernel:

```
epic0 at pci0 dev 12 function 0: SMC EPIC/100 Fast Ethernet
epic0: interrupting at kn20aa irq 4
epic0: SMC9432TX, Ethernet address 00:e0:29:07:17:c4
qsphy0 at epic0 phy 3: QS6612 10/100 media interface, rev. 1
qsphy0: 10baseT, 10baseT-FDX, 100baseTX, 100baseTX-FDX, auto
```

All PHY drivers display the media types supported by the PHY when it is detected by the kernel. These media types are valid media keywords for use with the `ifconfig`(8) program.

SEE ALSO

acphy(4), amhphy(4), bmtphy(4), brgphy(4), ciphy(4), dmphy(4), exphy(4), gentbi(4), glxtphy(4), gphyter(4), icsphy(4), ifmedia(4), ikphy(4), inphy(4), iophy(4), lxtphy(4), makphy(4), nsphy(4), nsphyter(4), pnaphy(4), qsphy(4), rgephy(4), rlphy(4), sqphy(4), tlphy(4), tqphy(4), ukphy(4), urlphy(4), ifconfig(8)

The OUI assignments list can be found at:

http://standards.ieee.org/regauth/oui/index.shtml

NAME

mk48txx — Mostek time-of-day clock driver

SYNOPSIS

```
#include <dev/ic/mk48txxreg.h>
#include <dev/ic/mk48txxvar.h>
define mk48txx
file   dev/ic/mk48txx.c    mk48txx
```

DESCRIPTION

The **mk48txx** driver provides access to several models of Mostek time-of-day clock chips. Access methods to retrieve and set date and time are provided through the *TODR* interface defined in todr(9).

To tie an instance of this device to the system, use the **mk48txx_attach**() function and the mk48txx_softc structure defined as follows:

void **mk48txx_attach**(*struct mk48txx_softc **)

```
typedef uint8_t (*mk48txx_nvrd_t)(struct mk48txx_softc *, int off);
typedef void (*mk48txx_nvwr_t)(struct mk48txx_softc *, int off,
    uint8_t datum);

struct mk48txx_softc {
        struct device   sc_dev;
        bus_space_tag_t sc_bst;
        bus_space_handle_t sc_bsh;
        struct todr_chip_handle sc_handle;
        const char     *sc_model;
        bus_size_t     sc_nvramsz;
        bus_size_t     sc_clkoffset;
        u_int          sc_year0;
        u_int          sc_flag;
        mk48txx_nvrd_t sc_nvrd;
        mk48txx_nvwr_t sc_nvwr;
};
```

sc_bst

sc_bsh Specify bus space access to the chip's non-volatile memory (including the clock registers).

sc_handle TODR handle passed to the **todr_attach**() function to register todr(9) interface.

sc_model The chip model which this instance should serve. Must be one of "mk48t02", "mk48t08", "mk48t18", or "mk48t59".

sc_nvramsz Size of non-volatile RAM in the Mostek chip. This value is set by **mk48txx_attach**().

sc_clkoffset
Offset into the control registers of the Mostek chip. This value is set by **mk48txx_attach**().

sc_year0 The actual year represented by the clock's 'year' counter. This is generally dependent on the system configuration in which the clock device is mounted. For instance, on Sun Microsystems machines the convention is to have clock's two-digit year represent the year 1968.

sc_flag This flag is used to specify machine-dependent features.

sc_nvread

sc_nvwrite Specify alternate access methods for reading resp. writing clock device registers. The default, when NULL is passed as an access method, is to access the chip memory (and clock registers) as if they were direct-mapped with using the specified bus space.

Otherwise, the driver will call the respective function to perform the access, passing it the specified bus space and the offset *off* of the chip memory (or clock register) location to be read from or written to, respectively.

Note that if the resulting date retrieved with the todr_gettime() method is earlier that January 1, 1970, the driver will assume that the chip's year counter actually represents a year in the 21st century. This behaviour can be overridden by setting the *MK48TXX_NO_CENT_ADJUST* flag in *sc_flag*.

HARDWARE

The following models are supported:

Mostek MK48T02
Mostek MK48T08
Mostek MK48T18
Mostek MK48T59

SEE ALSO

intro(4), todr(9)

HISTORY

The **mk48txx** driver first appeared in NetBSD 1.5.

AUTHORS

The **mk48txx** driver was written by Paul Kranenburg ⟨pk@NetBSD.org⟩.

NAME

mkbd — Maple bus keyboard driver

SYNOPSIS

```
mkbd*  at maple? port ? subunit ?
wskbd* at mkbd? console ?
```

DESCRIPTION

The **mkbd** driver provides support for Maple bus keyboards. Access to the keyboard is through the wscons(4) driver.

SEE ALSO

maple(4), wskbd(4)

HISTORY

The **mkbd** device driver appeared in NetBSD 1.6.

NAME

mlcd — Maple bus monochrome LCD driver

SYNOPSIS

mlcd* **at maple? port ? subunit ?**

DESCRIPTION

The **mlcd** driver provides support for Maple bus monochrome LCDs (liquid crystal displays).

IOCTLS

Maple bus LCDs accept all `ioctl`(2) calls described in `maple`(4).

FILES

`/dev/mlcd`*u*.*t* Maple bus LCD device unit *u*, PT number *t* (usually 0 only)

SEE ALSO

`ioctl`(2), `maple`(4)

HISTORY

The **mlcd** device driver appeared in NetBSD 2.0.

NAME

mlx — Mylex DAC960 RAID controller driver

SYNOPSIS

```
mlx* at eisa? slot ?
mlx* at pci? dev ? function ?
```

DESCRIPTION

The **mlx** driver provides support for the Mylex DAC960 family of RAID controllers, including OEM versions from Compaq and DEC. Attached disk arrays are supported by the **ld** driver.

The **mlx** driver will warn if a controller firmware upgrade may be necessary, although as a matter of course, the latest available firmware should always be used.

HARDWARE

Supported controllers include:

DEC/Compaq SWXCR
Mylex AcceleRAID 150
Mylex AcceleRAID 200
Mylex AcceleRAID 250
Mylex DAC1164P (eXtremeRAID 1100)
Mylex DAC960P
Mylex DAC960PD
Mylex DAC960PG
Mylex DAC960PJ
Mylex DAC960PL
Mylex DAC960PR
Mylex DAC960PRL
Mylex DAC960PT
Mylex DAC960PTL0
Mylex DAC960PTL1

SEE ALSO

intro(4), ld(4), mly(4), mlxctl(8)

HISTORY

The **mlx** driver first appeared in NetBSD 1.5.3, and was derived from the FreeBSD driver of the same name.

NAME

mly — Mylex AcceleRAID/eXtremeRAID family driver

SYNOPSIS

```
mly* at pci? dev ? function ?
scsibus* at mly?
```

DESCRIPTION

The **mly** driver provides support for Mylex AcceleRAID and eXtremeRAID family of PCI to SCSI RAID controllers with version 6.00 and later firmware. Supported controllers include:

- AcceleRAID 160
- AcceleRAID 170
- AcceleRAID 352
- eXtremeRAID 2000
- eXtremeRAID 3000

Compatible Mylex controllers not listed should work, but have not been tested.

Logical devices (disk arrays) attached to the controller are presented to the SCSI subsystem as though they were direct-access devices on a virtual SCSI bus. Physical devices which are not claimed by a logical device are presented on SCSI channels which match the physical channels on the controller.

The results of the SCSI ''INQUIRY'' command from logical devices are overwritten with status information by the **mly** driver. The vendor field is the string ''MYLEX'', the product field indicates the type of logical device, and the revision field contains a four letter status code. The possible status codes and their meanings are as follows:

OFLN
: offline

UNCF
: unconfigured

ONLN
: online - optimal

CRIT critical - one or more disks in the array has failed

NORD
: write only

STBY
: standby

MISS missing

DIAGNOSTICS

Controller initialization phase

mly%d: controller initialization started
mly%d: initialization complete

The controller firmware has started initialization. Normally this process is performed by the controller BIOS, but the driver may need to do this in cases where the BIOS has failed, or is not compatible (e.g. on non-x86 systems).

mly%d: drive spinup in progress

Drive startup is in progress; this may take several minutes.

mly%d: mirror race recovery failed, one or more drives offline
mly%d: mirror race recovery in progress
mly%d: mirror race recovery on a critical drive

These error codes are undocumented.

mly%d: FATAL MEMORY PARITY ERROR

Firmware detected a fatal memory error; the driver will not attempt to attach to this controller.

mly%d: unknown initialization code %x

An unknown error occurred during initialization; it will be ignored.

Operational diagnostics

mly%d: physical device %d:%d online
mly%d: physical device %d:%d standby
mly%d: physical device %d:%d automatic rebuild started
mly%d: physical device %d:%d manual rebuild started
mly%d: physical device %d:%d rebuild completed
mly%d: physical device %d:%d rebuild cancelled
mly%d: physical device %d:%d rebuild failed for unknown reasons
mly%d: physical device %d:%d rebuild failed due to new physical device
mly%d: physical device %d:%d rebuild failed due to logical drive failure
mly%d: physical device %d:%d found
mly%d: physical device %d:%d gone
mly%d: physical device %d:%d unconfigured
mly%d: physical device %d:%d expand capacity started
mly%d: physical device %d:%d expand capacity completed
mly%d: physical device %d:%d expand capacity failed
mly%d: physical device %d:%d parity error
mly%d: physical device %d:%d soft error
mly%d: physical device %d:%d miscellaneous error
mly%d: physical device %d:%d reset
mly%d: physical device %d:%d active spare found
mly%d: physical device %d:%d warm spare found
mly%d: physical device %d:%d initialization started
mly%d: physical device %d:%d initialization completed
mly%d: physical device %d:%d initialization failed
mly%d: physical device %d:%d initialization cancelled
mly%d: physical device %d:%d write recovery failed
mly%d: physical device %d:%d scsi bus reset failed
mly%d: physical device %d:%d double check condition
mly%d: physical device %d:%d device cannot be accessed
mly%d: physical device %d:%d gross error on scsi processor
mly%d: physical device %d:%d bad tag from device
mly%d: physical device %d:%d command timeout
mly%d: physical device %d:%d system reset
mly%d: physical device %d:%d busy status or parity error
mly%d: physical device %d:%d host set device to failed state
mly%d: physical device %d:%d selection timeout
mly%d: physical device %d:%d scsi bus phase error

mly%d: physical device %d:%d device returned unknown status
mly%d: physical device %d:%d device not ready
mly%d: physical device %d:%d device not found at startup
mly%d: physical device %d:%d COD write operation failed
mly%d: physical device %d:%d BDT write operation failed
mly%d: physical device %d:%d missing at startup
mly%d: physical device %d:%d start rebuild failed due to physical drive too small
mly%d: physical device %d:%d sense data received
mly%d: sense key %d asc %02x ascq %02x
mly%d: info %4D csi %4D
mly%d: physical device %d:%d offline
mly%d: sense key %d asc %02x ascq %02x
mly%d: info %4D csi %4D

The reported event refers to the physical device at the given channel:target address.

mly%d: logical device %d:%d consistency check started
mly%d: logical device %d:%d consistency check completed
mly%d: logical device %d:%d consistency check cancelled
mly%d: logical device %d:%d consistency check completed with errors
mly%d: logical device %d:%d consistency check failed due to logical drive failure
mly%d: logical device %d:%d consistency check failed due to physical device failure
mly%d: logical device %d:%d automatic rebuild started
mly%d: logical device %d:%d manual rebuild started
mly%d: logical device %d:%d rebuild completed
mly%d: logical device %d:%d rebuild cancelled
mly%d: logical device %d:%d rebuild failed for unknown reasons
mly%d: logical device %d:%d rebuild failed due to new physical device
mly%d: logical device %d:%d rebuild failed due to logical drive failure
mly%d: logical device %d:%d offline
mly%d: logical device %d:%d critical
mly%d: logical device %d:%d online
mly%d: logical device %d:%d initialization started
mly%d: logical device %d:%d initialization completed
mly%d: logical device %d:%d initialization cancelled
mly%d: logical device %d:%d initialization failed
mly%d: logical device %d:%d found
mly%d: logical device %d:%d gone
mly%d: logical device %d:%d expand capacity started
mly%d: logical device %d:%d expand capacity completed
mly%d: logical device %d:%d expand capacity failed
mly%d: logical device %d:%d bad block found
mly%d: logical device %d:%d size changed
mly%d: logical device %d:%d type changed
mly%d: logical device %d:%d bad data block found
mly%d: logical device %d:%d read of data block in bdt
mly%d: logical device %d:%d write back data for disk block lost

The reported event refers to the logical device at the given channel:target address.

mly%d: enclosure %d fan %d failed

mly%d: enclosure %d fan %d ok
mly%d: enclosure %d fan %d not present
mly%d: enclosure %d power supply %d failed
mly%d: enclosure %d power supply %d ok
mly%d: enclosure %d power supply %d not present
mly%d: enclosure %d temperature sensor %d failed
mly%d: enclosure %d temperature sensor %d critical
mly%d: enclosure %d temperature sensor %d ok
mly%d: enclosure %d temperature sensor %d not present
mly%d: enclosure %d unit %d access critical
mly%d: enclosure %d unit %d access ok
mly%d: enclosure %d unit %d access offline

These events refer to external enclosures by number. The driver does not attempt to name the enclosures.

mly%d: controller cache write back error
mly%d: controller battery backup unit found
mly%d: controller battery backup unit charge level low
mly%d: controller battery backup unit charge level ok
mly%d: controller installation aborted
mly%d: controller mirror race recovery in progress
mly%d: controller mirror race on critical drive
mly%d: controller memory soft ecc error
mly%d: controller memory hard ecc error
mly%d: controller battery backup unit failed

These events report controller status changes.

SEE ALSO

cd(4), ch(4), intro(4), mlx(4), scsi(4), sd(4), st(4), scsictl(8)

HISTORY

The **mly** driver first appeared in NetBSD 1.6, and was based on the FreeBSD driver of the same name.

BUGS

The **mly** driver currently assumes that all busses support at most 16 targets and 1 logical unit per target.

Enclosures are not named or otherwise identified in event messages.

The transfer speed for devices is always reported to the kernel as 20MHz.

NAME

mmem — Maple bus storage device driver

SYNOPSIS

mmem* **at maple? port ? subunit ?**

DESCRIPTION

The **mmem** driver provides support for Maple bus storage devices (memory cards).

IOCTLS

The following `ioctl`(2) calls apply to Maple bus storage devices.

`DIOCGDINFO` *`struct disklabel`*
Read, from the kernel, the `disklabel`(5) of the device.

In addition, Maple bus storage devices accept all `ioctl`(2) calls described in `maple`(4).

FILES

`/dev/mmem`*`u.tp`*	block mode Maple bus storage device unit *u*, PT number *t* (usually 0 only), partition *p*
`/dev/rmmem`*`u.tp`*	raw mode Maple bus storage device unit *u*, PT number *t* (usually 0 only), partition *p*

SEE ALSO

`ioctl`(2), `maple`(4), `disklabel`(5)

HISTORY

The **mmem** device driver appeared in NetBSD 2.0.

NAME

mms — Maple bus mouse driver

SYNOPSIS

```
mms*     at maple? port ? subunit ?
wsmouse* at mms?
```

DESCRIPTION

The **mms** driver provides support for Maple bus mice. Access to the mouse is through the wscons(4) driver.

SEE ALSO

maple(4), wsmouse(4)

HISTORY

The **mms** device driver appeared in NetBSD 1.6.

NAME

mms — Microsoft-style bus mouse driver

SYNOPSIS

```
mms0 at isa? port 0x23c irq 5
mms1 at isa? port 0x238 irq 5
wsmouse* at mms?
```

DESCRIPTION

This driver provides an interface to a Microsoft-style bus mouse. Mouse related data are accessed by wsmouse(4) devices.

SEE ALSO

lms(4), pms(4), wsmouse(4)

NAME

mongoose — EISA Bus adapter

SYNOPSIS

```
mongoose* at mainbus?
eisa*     at mongoose?
```

DESCRIPTION

Provides an interface from the CPU-memory bus to EISA and ISA devices. Two variations exist providing the same functionality based on Intel i82350 or Texas Instruments chips. Depending on the model the bus clock is either 25 MHz or 33 MHz.

MACHINES

An incomplete list of machines that use the mongoose bus controller:

- 715/{33,50,75}
- 720, 730, 750
- 725/{50,75}
- 735/*
- 755/*
- 742i
- 745i/{50,75}
- 747i/{50,75}

SEE ALSO

eisa(4), intro(4)

HISTORY

The **mongoose** driver appeared in OpenBSD 2.6. It was ported to NetBSD 1.6 by Matthew Fredette.

BUGS

Has some.

NAME

mpt — LSI Fusion-MPT SCSI/Fibre Channel driver

SYNOPSIS

```
mpt* at pci? dev ? function ?
scsibus* at mpt?
```

DESCRIPTION

The **mpt** driver provides support for the LSI Logic Fusion-MPT family of SCSI and Fibre Channel controllers:

- 53c1020 (Ultra320 SCSI)
- 53c1030 (Dual Ultra320 SCSI)
- FC909 (1Gb/s Fibre Channel)
- FC909A (Dual 1Gb/s Fibre Channel)
- FC919 (2Gb/s Fibre Channel)
- FC919X (2Gb/s Fibre Channel, PCI-X)
- FC929 (Dual 2Gb/s Fibre Channel)
- FC929X (Dual 2Gb/s Fibre Channel, PCI-X)

SEE ALSO

cd(4), ch(4), intro(4), pci(4), scsi(4), sd(4), siop(4), st(4), uk(4)

HISTORY

The **mpt** driver first appeared in NetBSD 2.0.

AUTHORS

The **mpt** driver was originally written for FreeBSD by Greg Ansley . It was ported to NetBSD by Jason R. Thorpe and contributed by Wasabi Systems, Inc.

NAME

mpu — Roland MPU401 (and compatible) MIDI UART driver

SYNOPSIS

```
mpu*  at acpi?
mpu*  at eso?
mpu*  at fms?
mpu*  at isa? port 0x330 irq 9
mpu*  at sb?
mpu*  at ym?
mpu*  at yds?
midi* at mpu?
```

DESCRIPTION

The **mpu** driver provides support for the Roland MPU401 (and compatible) MIDI UART cards.

Access to the device is through the MIDI driver.

SEE ALSO

eso(4), fms(4), isa(4), midi(4), sb(4), yds(4), ym(4)

HISTORY

The **mpu** device driver appeared in NetBSD 1.5.

NAME

mr — Guillemot Maxi Radio FM 2000 FM radio device driver

SYNOPSIS

```
mr*    at pci? dev ? function ?
radio* at mr?
```

DESCRIPTION

The **mr** driver provides support for the Maxi Radio FM radio tuners.

The Maxi Radio cards are stereo FM tuners that allow to tune in the range 87.5 - 108.0 MHz, report signal status on the current frequency, force audio output to mono, perform hardware signal search, and have an internal AFC.

SEE ALSO

pci(4), radio(4)

HISTORY

The **mr** device driver appeared in OpenBSD 3.0 and NetBSD 1.6.

AUTHORS

The **mr** driver was written by Vladimir Popov and Maxim Tsyplakov. The man page was written by Vladimir Popov.

NAME

ms — Atari mouse interface

SYNOPSIS

```
pseudo-device mouse 1
```

DESCRIPTION

The Atari mouse driver supports both the original Atari mouse and the third party 3-button mouse that has its middle button connected to the up-switch of the second joystick port. To accommodate X11 users with a standard mouse, the driver is able to emulate the middle button. See the section on ioctls for more info.

Supported ioctls

MIOCS3B_EMUL

This ioctl turns the middle button emulation on or off depending on its argument. The middle button event is triggered by simultaneously pressing the left and right buttons. The default emulation mode is on.

Note that the emulation status is retained across multiple open/close calls.

MIOCG3B_EMUL

This ioctl allows you to get the actual status of the emulation mode.

Interface description

The Atari mouse interface works on a minimal emulation of Sun's Firm_event structures. The primary reason for this is easy interfacing with X11.

The movement and button events are read as structures of the form:

```
typedef struct Firm_event {
        u_int_16_t      id;       /* key or MS_* or LOC_[XY]_DELTA   */
        u_int_16_t      pad;      /* unused                          */
        int_16_t        value;    /* VKEY_{UP,DOWN} or locator delta */
        struct timeval  time;     /* time stamp of the event         */
}
```

The values of 'id' concerning the mouse:

```
#define MS_LEFT         0x7f20    /* left mouse button   */
#define MS_MIDDLE       0x7f21    /* middle mouse button */
#define MS_RIGHT        0x7f22    /* right mouse button  */
#define LOC_X_DELTA     0x7f80    /* mouse delta-X       */
#define LOC_Y_DELTA     0x7f81    /* mouse delta-Y       */
```

The values of 'value' concerning a button event:

```
#define VKEY_UP         0         /* a button went up   */
#define VKEY_DOWN       1         /* a button went down */
```

FILES

`/dev/mouse0` The real mouse device
`/dev/mouse` The currently active mouse device

BUGS

The time interval that defines 'simultaneous' cannot be set.

NAME

ms — Sun workstation mouse driver

SYNOPSIS

```
pseudo-device mouse
```

DESCRIPTION

The **ms** driver provides an interface to the workstation console mouse. This Mouse Systems three-button device produces five-byte blobs of the form:

```
b dx dy dx dy
```

where “b” is the button state, encoded as `0x80|(~buttons)` -- there are three buttons (4=left, 2=middle, 1=right) -- and “dx” and “dy” are X and Y delta values, none of which are in the range `[0x80..0x87]`.

The device special file `/dev/mouse` is used to get direct access to the mouse input stream. The following ioctl’s are supported (mostly just enough to keep the X(1) server going):

`VUIDSFORMAT` Set translation mode. The argument is of type *`int *`*, the only value supported is `VUID_FIRM_EVENT`.

`VUIDGFORMAT` Get translation mode. The argument is of type *`int *`*. `VUID_FIRM_EVENT` is always returned.

OPTIONS

The mouse driver can be configured using the following kernel configuration file options.

`options SUN_MS_BPS=integer`

This option causes the kernel to communicate with the mouse using the serial baud rate *integer*. It is useful for mice which do not communicate at 1200 baud.

SEE ALSO

`kbd`(4)

BUGS

ms is hardwired to the built-in *zs1* serial port.

NAME

ms — Sun workstation mouse driver

SYNOPSIS

```
ms0 at zstty?
```

DESCRIPTION

The **ms** driver provides an interface to the workstation console mouse. This Mouse Systems three-button device produces five-byte blobs of the form:

```
b dx dy dx dy
```

where “b” is the button state, encoded as `0x80|(~buttons)` -- there are three buttons (4=left, 2=middle, 1=right) -- and “dx” and “dy” are X and Y delta values, none of which are in the range `[0x80..0x87]`.

The device special file `/dev/mouse` is used to get direct access to the mouse input stream. The following ioctl’s are supported (mostly just enough to keep the X(1) server going):

`VUIDSFORMAT` Set translation mode. The argument is of type *`int *`*, the only value supported is `VUID_FIRM_EVENT`.

`VUIDGFORMAT` Get translation mode. The argument is of type *`int *`*. `VUID_FIRM_EVENT` is always returned.

OPTIONS

The mouse driver can be configured using the following kernel configuration file options.

`options SUN_MS_BPS=integer`

This option causes the kernel to communicate with the mouse using the serial baud rate *integer*. It is useful for mice which do not communicate at 1200 baud.

SEE ALSO

`kbd`(4)

BUGS

ms is hardwired to the built-in *zs1* serial port.

NAME

ms — Sun workstation mouse driver

SYNOPSIS

```
pseudo-device mouse
```

DESCRIPTION

The **ms** driver provides an interface to the workstation console mouse. This Mouse Systems three-button device produces five-byte blobs of the form:

```
b dx dy dx dy
```

where “b” is the button state, encoded as `0x80|(~buttons)` -- there are three buttons (4=left, 2=middle, 1=right) -- and “dx” and “dy” are X and Y delta values, none of which are in the range `[0x80..0x87]`.

The device special file `/dev/mouse` is used to get direct access to the mouse input stream. The following ioctl’s are supported (mostly just enough to keep the X(1) server going):

`VUIDSFORMAT` Set translation mode. The argument is of type *`int *`*, the only value supported is `VUID_FIRM_EVENT`.

`VUIDGFORMAT` Get translation mode. The argument is of type *`int *`*. `VUID_FIRM_EVENT` is always returned.

OPTIONS

The mouse driver can be configured using the following kernel configuration file options.

`options SUN_MS_BPS=integer`

This option causes the kernel to communicate with the mouse using the serial baud rate *integer*. It is useful for mice which do not communicate at 1200 baud.

SEE ALSO

kbd(4)

BUGS

ms is hardwired to the built-in *zs1* serial port.

NAME

mt — TM-78/TU-78 MASSBUS mag tape interface

SYNOPSIS

```
mt0 at mba? drive ? tape mu0 at mt0 slave 0
```

DESCRIPTION

The TM-78/TU-78 combination provides a standard tape drive interface as described in mtio(4). Only 1600 and 6250 BPI are supported; the TU-78 runs at 125 IPS and autoloads tapes.

DIAGNOSTICS

mu%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

mu%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

mu%d: can't change density in mid-tape. An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.

mu%d: hard error bn%d mbsr=%b er=%x ds=%b. A tape error occurred at block *bn*; the mt error register and drive status register are printed in octal with the bits symbolically decoded. Any error is fatal on non-raw tape; when possible the driver will have retried the operation which failed several times before reporting the error.

mu%d: blank tape. An attempt was made to read a blank tape (a tape without even end-of-file marks).

mu%d: offline. During an i/o operation the device was set offline. If a non-raw tape was used in the access it is closed.

SEE ALSO

mt(1), tar(1), mtio(4), tm(4), ts(4), ut(4)

HISTORY

The **mt** driver appeared in 4.1BSD.

BUGS

If a physical error (non-data) occurs, **mt** may hang ungracefully.

Because 800 BPI tapes are not supported, the numbering of minor devices is inconsistent with triple-density tape units. Unit 0 is drive 0, 1600 BPI.

NAME

mtc — UNIBUS MSCP tape controller driver

SYNOPSIS

```
mtc0 at uba? csr 0174500
mscpbus* at mtc?
```

DESCRIPTION

The **mtc** driver is for UNIBUS tape controllers that use MSCP. Among these controllers are:

DEC KLESI-U UNIBUS ctlr
DEC TK50 Q22 bus ctlr

The **mtc** communicates with the host through a packet protocol known as the Mass Storage Control Protocol (MSCP). Consult the file ⟨mscp/mscp.h⟩ for a detailed description of this protocol.

SEE ALSO

intro(4)

HISTORY

The **mtc** driver appeared in NetBSD 1.2.

NAME

mtd — Driver for Myson Technologies MTD803 3-in-1 Fast Ethernet board

SYNOPSIS

```
mtd* at pci?
```

DESCRIPTION

The **mtd** device driver supports the MTD803 PCI Ethernet chip.

Supported models include:

- Safeway Lancard SW-10/100 PCI (model 117204). Please note that some cards sold under this name are supported by rtk(4) instead.

SEE ALSO

intro(4), mii(4), pci(4), rtk(4), ifconfig(8)

HISTORY

The **mtd** driver appeared in NetBSD 2.0.

AUTHORS

Peter Bex ⟨Peter.Bex@student.kun.nl⟩

BUGS

This driver has not been tested on any architecture besides i386. It does not handle DMA cache flushes at all, so it will very likely not work on other architectures that require this.

Power management is missing.

A cardbus variant is rumored to exist, but support for it has not been added to the driver yet.

NAME

mtio — generic magnetic tape I/O interface

SYNOPSIS

```
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/mtio.h>
```

DESCRIPTION

Magnetic tape has been the computer system backup and data transfer medium of choice for decades, because it has historically been cheaper in cost per bit stored, and the formats have been designed for portability and storage. However, tape drives have generally been the slowest mass storage devices attached to any computer system.

Magnetic tape comes in a wide variety of formats, from classic 9-track, through various Quarter Inch Cartridge (QIC) variants, to more modern systems using 8mm video tape, and Digital Audio Tape (DAT). There have also been a variety of proprietary tape systems, including DECtape, and IBM 3480.

UNIX TAPE I/O

Regardless of the specific characteristics of the particular tape transport mechanism (tape drive), UNIX tape I/O has two interfaces: "block" and "raw". I/O through the block interface of a tape device is similar to I/O through the block special device for a disk driver: the individual `read`(2) and `write`(2) calls can be done in any amount of bytes, but all data is buffered through the system buffer cache, and I/O to the device is done in 1024 byte sized blocks. This limitation is sufficiently restrictive that the block interface to tape devices is rarely used.

The "raw" interface differs in that all I/O can be done in arbitrary sized blocks, within the limitations for the specific device and device driver, and all I/O is synchronous. This is the most flexible interface, but since there is very little that is handled automatically by the kernel, user programs must implement specific magnetic tape handling routines, which puts the onus of correctness on the application programmer.

DEVICE NAME CONVENTIONS

Each magnetic tape subsystem has a couple of special devices associated with it.

The block device is usually named for the driver, e.g. `/dev/st0` for unit zero of a `st`(4) SCSI tape drive.

The raw device name is the block device name with an "r" prepended, e.g. `/dev/rst0`.

By default, the tape driver will rewind the tape drive when the device is closed. To make it possible for multiple program invocations to sequentially write multiple files on the same tape, a "no rewind on close" device is provided, denoted by the letter "n" prepended to the name of the device, e.g. `/dev/nst0`, `/dev/nrst0`.

The `mt`(1) command can be used to explicitly rewind, or otherwise position a tape at a particular point with the no-rewind device.

FILE MARK HANDLING

Two end-of-file (EOF) markers mark the end of a tape (EOT), and one end-of-file marker marks the end of a tape file.

By default, the tape driver will write two End Of File (EOF) marks and rewind the tape when the device is closed after the last write.

If the tape is not to be rewound it is positioned with the head in between the two tape marks, where the next write will over write the second end-of-file marker.

All of the magnetic tape devices may be manipulated with the `mt`(1) command.

A number of `ioctl`(2) operations are available on raw magnetic tape. Please see ⟨`sys/mtio.h`⟩ for their definitions.

The manual pages for specific tape device drivers should list their particular capabilities and limitations.

SEE ALSO

`dd`(1), `mt`(1), `pax`(1), `tar`(1), `st`(4), `wt`(4)

HISTORY

The **`mtio`** manual appeared in 4.2BSD.

BUGS

The status should be returned in a device independent format.

If and when NetBSD is updated to deal with non-512 byte per sector disk media through the system buffer cache, perhaps a more sane tape interface can be implemented.

NAME

multicast — Multicast Routing

SYNOPSIS

```
options MROUTING

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip_mroute.h>
#include <netinet6/ip6_mroute.h>

int
getsockopt(int s, IPPROTO_IP, MRT_INIT, void *optval, socklen_t *optlen);

int
setsockopt(int s, IPPROTO_IP, MRT_INIT, const void *optval,
    socklen_t optlen);

int
getsockopt(int s, IPPROTO_IPV6, MRT6_INIT, void *optval, socklen_t *optlen);

int
setsockopt(int s, IPPROTO_IPV6, MRT6_INIT, const void *optval,
    socklen_t optlen);
```

DESCRIPTION

Multicast routing is used to efficiently propagate data packets to a set of multicast listeners in multipoint networks. If unicast is used to replicate the data to all listeners, then some of the network links may carry multiple copies of the same data packets. With multicast routing, the overhead is reduced to one copy (at most) per network link.

All multicast-capable routers must run a common multicast routing protocol. The Distance Vector Multicast Routing Protocol (DVMRP) was the first developed multicast routing protocol. Later, other protocols such as Multicast Extensions to OSPF (MOSPF), Core Based Trees (CBT), Protocol Independent Multicast - Sparse Mode (PIM-SM), and Protocol Independent Multicast - Dense Mode (PIM-DM) were developed as well.

To start multicast routing, the user must enable multicast forwarding in the kernel (see **SYNOPSIS** about the kernel configuration options), and must run a multicast routing capable user-level process. From developer's point of view, the programming guide described in the **Programming Guide** section should be used to control the multicast forwarding in the kernel.

Programming Guide

This section provides information about the basic multicast routing API. The so-called "advanced multicast API" is described in the **Advanced Multicast API Programming Guide** section.

First, a multicast routing socket must be open. That socket would be used to control the multicast forwarding in the kernel. Note that most operations below require certain privilege (i.e., root privilege):

```
/* IPv4 */
int mrouter_s4;
mrouter_s4 = socket(AF_INET, SOCK_RAW, IPPROTO_IGMP);

int mrouter_s6;
mrouter_s6 = socket(AF_INET6, SOCK_RAW, IPPROTO_ICMPV6);
```

Note that if the router needs to open an IGMP or ICMPv6 socket (in case of IPv4 and IPv6 respectively) for sending or receiving of IGMP or MLD multicast group membership messages, then the same *mrouter_s4* or *mrouter_s6* sockets should be used for sending and receiving respectively IGMP or MLD messages. In case of BSD-derived kernel, it may be possible to open separate sockets for IGMP or MLD messages only. However, some other kernels (e.g., Linux) require that the multicast routing socket must be used for sending and receiving of IGMP or MLD messages. Therefore, for portability reason the multicast routing socket should be reused for IGMP and MLD messages as well.

After the multicast routing socket is open, it can be used to enable or disable multicast forwarding in the kernel:

```
/* IPv4 */
int v = 1;        /* 1 to enable, or 0 to disable */
setsockopt(mrouter_s4, IPPROTO_IP, MRT_INIT, (void *)&v, sizeof(v));

/* IPv6 */
int v = 1;        /* 1 to enable, or 0 to disable */
setsockopt(mrouter_s6, IPPROTO_IPV6, MRT6_INIT, (void *)&v, sizeof(v));
...
/* If necessary, filter all ICMPv6 messages */
struct icmp6_filter filter;
ICMP6_FILTER_SETBLOCKALL(&filter);
setsockopt(mrouter_s6, IPPROTO_ICMPV6, ICMP6_FILTER, (void *)&filter,
           sizeof(filter));
```

After multicast forwarding is enabled, the multicast routing socket can be used to enable PIM processing in the kernel if we are running PIM-SM or PIM-DM (see `pim`(4)).

For each network interface (e.g., physical or a virtual tunnel) that would be used for multicast forwarding, a corresponding multicast interface must be added to the kernel:

```
/* IPv4 */
struct vifctl vc;
memset(&vc, 0, sizeof(vc));
/* Assign all vifctl fields as appropriate */
vc.vifc_vifi = vif_index;
vc.vifc_flags = vif_flags;
vc.vifc_threshold = min_ttl_threshold;
vc.vifc_rate_limit = max_rate_limit;
memcpy(&vc.vifc_lcl_addr, &vif_local_address, sizeof(vc.vifc_lcl_addr));
if (vc.vifc_flags & VIFF_TUNNEL)
    memcpy(&vc.vifc_rmt_addr, &vif_remote_address,
           sizeof(vc.vifc_rmt_addr));
setsockopt(mrouter_s4, IPPROTO_IP, MRT_ADD_VIF, (void *)&vc,
           sizeof(vc));
```

The *vif_index* must be unique per vif. The *vif_flags* contains the `VIFF_`* flags as defined in ⟨`netinet/ip_mroute.h`⟩. The *min_ttl_threshold* contains the minimum TTL a multicast data packet must have to be forwarded on that vif. Typically, it would have value of 1. The *max_rate_limit* contains the maximum rate (in bits/s) of the multicast data packets forwarded on that vif. Value of 0 means no limit. The *vif_local_address* contains the local IP address of the corresponding local interface. The *vif_remote_address* contains the remote IP address in case of DVMRP multicast tunnels.

```
/* IPv6 */
struct mif6ctl mc;
memset(&mc, 0, sizeof(mc));
```

```
/* Assign all mif6ctl fields as appropriate */
mc.mif6c_mifi = mif_index;
mc.mif6c_flags = mif_flags;
mc.mif6c_pifi = pif_index;
setsockopt(mrouter_s6, IPPROTO_IPV6, MRT6_ADD_MIF, (void *)&mc,
           sizeof(mc));
```

The *mif_index* must be unique per vif. The *mif_flags* contains the `MIFF_*` flags as defined in ⟨`netinet6/ip6_mroute.h`⟩. The *pif_index* is the physical interface index of the corresponding local interface.

A multicast interface is deleted by:

```
/* IPv4 */
vifi_t vifi = vif_index;
setsockopt(mrouter_s4, IPPROTO_IP, MRT_DEL_VIF, (void *)&vifi,
           sizeof(vifi));

/* IPv6 */
mifi_t mifi = mif_index;
setsockopt(mrouter_s6, IPPROTO_IPV6, MRT6_DEL_MIF, (void *)&mifi,
           sizeof(mifi));
```

After the multicast forwarding is enabled, and the multicast virtual interfaces are added, the kernel may deliver upcall messages (also called signals later in this text) on the multicast routing socket that was open earlier with `MRT_INIT` or `MRT6_INIT`. The IPv4 upcalls have *`struct igmpmsg`* header (see ⟨`netinet/ip_mroute.h`⟩) with field *im_mbz* set to zero. Note that this header follows the structure of *`struct ip`* with the protocol field *ip_p* set to zero. The IPv6 upcalls have *`struct mrt6msg`* header (see ⟨`netinet6/ip6_mroute.h`⟩) with field *im6_mbz* set to zero. Note that this header follows the structure of *`struct ip6_hdr`* with the next header field *ip6_nxt* set to zero.

The upcall header contains field *im_msgtype* and *im6_msgtype* with the type of the upcall `IGMPMSG_*` and `MRT6MSG_*` for IPv4 and IPv6 respectively. The values of the rest of the upcall header fields and the body of the upcall message depend on the particular upcall type.

If the upcall message type is `IGMPMSG_NOCACHE` or `MRT6MSG_NOCACHE`, this is an indication that a multicast packet has reached the multicast router, but the router has no forwarding state for that packet. Typically, the upcall would be a signal for the multicast routing user-level process to install the appropriate Multicast Forwarding Cache (MFC) entry in the kernel.

An MFC entry is added by:

```
/* IPv4 */
struct mfcctl mc;
memset(&mc, 0, sizeof(mc));
memcpy(&mc.mfcc_origin, &source_addr, sizeof(mc.mfcc_origin));
memcpy(&mc.mfcc_mcastgrp, &group_addr, sizeof(mc.mfcc_mcastgrp));
mc.mfcc_parent = iif_index;
for (i = 0; i < maxvifs; i++)
    mc.mfcc_ttls[i] = oifs_ttl[i];
setsockopt(mrouter_s4, IPPROTO_IP, MRT_ADD_MFC,
           (void *)&mc, sizeof(mc));

/* IPv6 */
struct mf6cctl mc;
memset(&mc, 0, sizeof(mc));
memcpy(&mc.mf6cc_origin, &source_addr, sizeof(mc.mf6cc_origin));
```

```
memcpy(&mc.mf6cc_mcastgrp, &group_addr, sizeof(mf6cc_mcastgrp));
mc.mf6cc_parent = iif_index;
for (i = 0; i < maxvifs; i++)
    if (oifs_ttl[i] > 0)
        IF_SET(i, &mc.mf6cc_ifset);
setsockopt(mrouter_s4, IPPROTO_IPV6, MRT6_ADD_MFC,
           (void *)&mc, sizeof(mc));
```

The *source_addr* and *group_addr* are the source and group address of the multicast packet (as set in the upcall message). The *iif_index* is the virtual interface index of the multicast interface the multicast packets for this specific source and group address should be received on. The *oifs_ttl[]* array contains the minimum TTL (per interface) a multicast packet should have to be forwarded on an outgoing interface. If the TTL value is zero, the corresponding interface is not included in the set of outgoing interfaces. Note that in case of IPv6 only the set of outgoing interfaces can be specified.

An MFC entry is deleted by:

```
/* IPv4 */
struct mfcctl mc;
memset(&mc, 0, sizeof(mc));
memcpy(&mc.mfcc_origin, &source_addr, sizeof(mc.mfcc_origin));
memcpy(&mc.mfcc_mcastgrp, &group_addr, sizeof(mc.mfcc_mcastgrp));
setsockopt(mrouter_s4, IPPROTO_IP, MRT_DEL_MFC,
           (void *)&mc, sizeof(mc));
```

```
/* IPv6 */
struct mf6cctl mc;
memset(&mc, 0, sizeof(mc));
memcpy(&mc.mf6cc_origin, &source_addr, sizeof(mc.mf6cc_origin));
memcpy(&mc.mf6cc_mcastgrp, &group_addr, sizeof(mf6cc_mcastgrp));
setsockopt(mrouter_s4, IPPROTO_IPV6, MRT6_DEL_MFC,
           (void *)&mc, sizeof(mc));
```

The following method can be used to get various statistics per installed MFC entry in the kernel (e.g., the number of forwarded packets per source and group address):

```
/* IPv4 */
struct sioc_sg_req sgreq;
memset(&sgreq, 0, sizeof(sgreq));
memcpy(&sgreq.src, &source_addr, sizeof(sgreq.src));
memcpy(&sgreq.grp, &group_addr, sizeof(sgreq.grp));
ioctl(mrouter_s4, SIOCGETSGCNT, &sgreq);
```

```
/* IPv6 */
struct sioc_sg_req6 sgreq;
memset(&sgreq, 0, sizeof(sgreq));
memcpy(&sgreq.src, &source_addr, sizeof(sgreq.src));
memcpy(&sgreq.grp, &group_addr, sizeof(sgreq.grp));
ioctl(mrouter_s6, SIOCGETSGCNT_IN6, &sgreq);
```

The following method can be used to get various statistics per multicast virtual interface in the kernel (e.g., the number of forwarded packets per interface):

```
/* IPv4 */
struct sioc_vif_req vreq;
memset(&vreq, 0, sizeof(vreq));
```

```
vreq.vifi = vif_index;
ioctl(mrouter_s4, SIOCGETVIFCNT, &vreq);

/* IPv6 */
struct sioc_mif_req6 mreq;
memset(&mreq, 0, sizeof(mreq));
mreq.mifi = vif_index;
ioctl(mrouter_s6, SIOCGETMIFCNT_IN6, &mreq);
```

Advanced Multicast API Programming Guide

If we want to add new features in the kernel, it becomes difficult to preserve backward compatibility (binary and API), and at the same time to allow user-level processes to take advantage of the new features (if the kernel supports them).

One of the mechanisms that allows us to preserve the backward compatibility is a sort of negotiation between the user-level process and the kernel:

1. The user-level process tries to enable in the kernel the set of new features (and the corresponding API) it would like to use.
2. The kernel returns the (sub)set of features it knows about and is willing to be enabled.
3. The user-level process uses only that set of features the kernel has agreed on.

To support backward compatibility, if the user-level process does not ask for any new features, the kernel defaults to the basic multicast API (see the **Programming Guide** section). Currently, the advanced multicast API exists only for IPv4; in the future there will be IPv6 support as well.

Below is a summary of the expandable API solution. Note that all new options and structures are defined in ⟨`netinet/ip_mroute.h`⟩ and ⟨`netinet6/ip6_mroute.h`⟩, unless stated otherwise.

The user-level process uses new **`getsockopt`**()/**`setsockopt`**() options to perform the API features negotiation with the kernel. This negotiation must be performed right after the multicast routing socket is open. The set of desired/allowed features is stored in a bitset (currently, in *`uint32_t`*; i.e., maximum of 32 new features). The new **`getsockopt`**()/**`setsockopt`**() options are `MRT_API_SUPPORT` and `MRT_API_CONFIG`. Example:

```
uint32_t v;
getsockopt(sock, IPPROTO_IP, MRT_API_SUPPORT, (void *)&v, sizeof(v));
```

would set in *v* the pre-defined bits that the kernel API supports. The eight least significant bits in *`uint32_t`* are same as the eight possible flags `MRT_MFC_FLAGS_*` that can be used in *mfcc_flags* as part of the new definition of *`struct mfcctl`* (see below about those flags), which leaves 24 flags for other new features. The value returned by **`getsockopt`**(*`MRT_API_SUPPORT`*) is read-only; in other words, **`setsockopt`**(*`MRT_API_SUPPORT`*) would fail.

To modify the API, and to set some specific feature in the kernel, then:

```
uint32_t v = MRT_MFC_FLAGS_DISABLE_WRONGVIF;
if (setsockopt(sock, IPPROTO_IP, MRT_API_CONFIG, (void *)&v, sizeof(v))
    != 0) {
    return (ERROR);
}
if (v & MRT_MFC_FLAGS_DISABLE_WRONGVIF)
    return (OK);       /* Success */
else
    return (ERROR);
```

In other words, when **setsockopt**(*MRT_API_CONFIG*) is called, the argument to it specifies the desired set of features to be enabled in the API and the kernel. The return value in *v* is the actual (sub)set of features that were enabled in the kernel. To obtain later the same set of features that were enabled, then:

```
getsockopt(sock, IPPROTO_IP, MRT_API_CONFIG, (void *)&v, sizeof(v));
```

The set of enabled features is global. In other words, **setsockopt**(*MRT_API_CONFIG*) should be called right after **setsockopt**(*MRT_INIT*).

Currently, the following set of new features is defined:

```
#define MRT_MFC_FLAGS_DISABLE_WRONGVIF (1 << 0) /* disable WRONGVIF signals */
#define MRT_MFC_FLAGS_BORDER_VIF   (1 << 1)  /* border vif              */
#define MRT_MFC_RP                  (1 << 8)  /* enable RP address   */
#define MRT_MFC_BW_UPCALL           (1 << 9)  /* enable bw upcalls   */
```

The advanced multicast API uses a newly defined *struct mfcctl2* instead of the traditional *struct mfcctl*. The original *struct mfcctl* is kept as is. The new *struct mfcctl2* is:

```
/*
 * The new argument structure for MRT_ADD_MFC and MRT_DEL_MFC overlays
 * and extends the old struct mfcctl.
 */
struct mfcctl2 {
        /* the mfcctl fields */
        struct in_addr  mfcc_origin;       /* ip origin of mcasts       */
        struct in_addr  mfcc_mcastgrp;     /* multicast group associated*/
        vifi_t          mfcc_parent;       /* incoming vif              */
        u_char          mfcc_ttls[MAXVIFS];/* forwarding ttls on vifs   */

        /* extension fields */
        uint8_t         mfcc_flags[MAXVIFS];/* the MRT_MFC_FLAGS_* flags*/
        struct in_addr  mfcc_rp;            /* the RP address           */
};
```

The new fields are *mfcc_flags[MAXVIFS]* and *mfcc_rp*. Note that for compatibility reasons they are added at the end.

The *mfcc_flags[MAXVIFS]* field is used to set various flags per interface per (S,G) entry. Currently, the defined flags are:

```
#define MRT_MFC_FLAGS_DISABLE_WRONGVIF (1 << 0) /* disable WRONGVIF signals */
#define MRT_MFC_FLAGS_BORDER_VIF       (1 << 1) /* border vif          */
```

The MRT_MFC_FLAGS_DISABLE_WRONGVIF flag is used to explicitly disable the IGMPMSG_WRONGVIF kernel signal at the (S,G) granularity if a multicast data packet arrives on the wrong interface. Usually, this signal is used to complete the shortest-path switch in case of PIM-SM multicast routing, or to trigger a PIM assert message. However, it should not be delivered for interfaces that are not in the outgoing interface set, and that are not expecting to become an incoming interface. Hence, if the MRT_MFC_FLAGS_DISABLE_WRONGVIF flag is set for some of the interfaces, then a data packet that arrives on that interface for that MFC entry will NOT trigger a WRONGVIF signal. If that flag is not set, then a signal is triggered (the default action).

The MRT_MFC_FLAGS_BORDER_VIF flag is used to specify whether the Border-bit in PIM Register messages should be set (in case when the Register encapsulation is performed inside the kernel). If it is set for the special PIM Register kernel virtual interface (see pim(4)), the Border-bit in the Register messages sent to the RP will be set.

The remaining six bits are reserved for future usage.

The *mfcc_rp* field is used to specify the RP address (in case of PIM-SM multicast routing) for a multicast group G if we want to perform kernel-level PIM Register encapsulation. The *mfcc_rp* field is used only if the MRT_MFC_RP advanced API flag/capability has been successfully set by **setsockopt**(*MRT_API_CONFIG*).

If the MRT_MFC_RP flag was successfully set by **setsockopt**(*MRT_API_CONFIG*), then the kernel will attempt to perform the PIM Register encapsulation itself instead of sending the multicast data packets to user level (inside IGMPMSG_WHOLEPKT upcalls) for user-level encapsulation. The RP address would be taken from the *mfcc_rp* field inside the new *struct mfcctl2*. However, even if the MRT_MFC_RP flag was successfully set, if the *mfcc_rp* field was set to INADDR_ANY, then the kernel will still deliver an IGMPMSG_WHOLEPKT upcall with the multicast data packet to the user-level process.

In addition, if the multicast data packet is too large to fit within a single IP packet after the PIM Register encapsulation (e.g., if its size was on the order of 65500 bytes), the data packet will be fragmented, and then each of the fragments will be encapsulated separately. Note that typically a multicast data packet can be that large only if it was originated locally from the same hosts that performs the encapsulation; otherwise the transmission of the multicast data packet over Ethernet for example would have fragmented it into much smaller pieces.

Typically, a multicast routing user-level process would need to know the forwarding bandwidth for some data flow. For example, the multicast routing process may want to timeout idle MFC entries, or in case of PIM-SM it can initiate (S,G) shortest-path switch if the bandwidth rate is above a threshold for example.

The original solution for measuring the bandwidth of a dataflow was that a user-level process would periodically query the kernel about the number of forwarded packets/bytes per (S,G), and then based on those numbers it would estimate whether a source has been idle, or whether the source's transmission bandwidth is above a threshold. That solution is far from being scalable, hence the need for a new mechanism for bandwidth monitoring.

Below is a description of the bandwidth monitoring mechanism.

- If the bandwidth of a data flow satisfies some pre-defined filter, the kernel delivers an upcall on the multicast routing socket to the multicast routing process that has installed that filter.
- The bandwidth-upcall filters are installed per (S,G). There can be more than one filter per (S,G).
- Instead of supporting all possible comparison operations (i.e., < <= == != > >=), there is support only for the <= and >= operations, because this makes the kernel-level implementation simpler, and because practically we need only those two. Further, the missing operations can be simulated by secondary user-level filtering of those <= and >= filters. For example, to simulate !=, then we need to install filter "bw <= 0xffffffff", and after an upcall is received, we need to check whether "measured_bw != expected_bw".
- The bandwidth-upcall mechanism is enabled by **setsockopt**(*MRT_API_CONFIG*) for the MRT_MFC_BW_UPCALL flag.
- The bandwidth-upcall filters are added/deleted by the new **setsockopt**(*MRT_ADD_BW_UPCALL*) and **setsockopt**(*MRT_DEL_BW_UPCALL*) respectively (with the appropriate *struct bw_upcall* argument of course).

From application point of view, a developer needs to know about the following:

```
/*
 * Structure for installing or delivering an upcall if the
 * measured bandwidth is above or below a threshold.
 *
 * User programs (e.g. daemons) may have a need to know when the
```

```
 * bandwidth used by some data flow is above or below some threshold.
 * This interface allows the userland to specify the threshold (in
 * bytes and/or packets) and the measurement interval. Flows are
 * all packet with the same source and destination IP address.
 * At the moment the code is only used for multicast destinations
 * but there is nothing that prevents its use for unicast.
 *
 * The measurement interval cannot be shorter than some Tmin (currently, 3s).
 * The threshold is set in packets and/or bytes per_interval.
 *
 * Measurement works as follows:
 *
 * For >= measurements:
 * The first packet marks the start of a measurement interval.
 * During an interval we count packets and bytes, and when we
 * pass the threshold we deliver an upcall and we are done.
 * The first packet after the end of the interval resets the
 * count and restarts the measurement.
 *
 * For <= measurement:
 * We start a timer to fire at the end of the interval, and
 * then for each incoming packet we count packets and bytes.
 * When the timer fires, we compare the value with the threshold,
 * schedule an upcall if we are below, and restart the measurement
 * (reschedule timer and zero counters).
 */

struct bw_data {
        struct timeval  b_time;
        uint64_t        b_packets;
        uint64_t        b_bytes;
};

struct bw_upcall {
        struct in_addr  bu_src;         /* source address            */
        struct in_addr  bu_dst;         /* destination address       */
        uint32_t        bu_flags;       /* misc flags (see below)    */
#define BW_UPCALL_UNIT_PACKETS (1 << 0) /* threshold (in packets)    */
#define BW_UPCALL_UNIT_BYTES   (1 << 1) /* threshold (in bytes)      */
#define BW_UPCALL_GEQ          (1 << 2) /* upcall if bw >= threshold */
#define BW_UPCALL_LEQ          (1 << 3) /* upcall if bw <= threshold */
#define BW_UPCALL_DELETE_ALL   (1 << 4) /* delete all upcalls for s,d*/
        struct bw_data  bu_threshold;   /* the bw threshold          */
        struct bw_data  bu_measured;    /* the measured bw           */
};

/* max. number of upcalls to deliver together */
#define BW_UPCALLS_MAX                          128
/* min. threshold time interval for bandwidth measurement */
#define BW_UPCALL_THRESHOLD_INTERVAL_MIN_SEC 3
#define BW_UPCALL_THRESHOLD_INTERVAL_MIN_USEC       0
```

The *bw_upcall* structure is used as an argument to **setsockopt**(*MRT_ADD_BW_UPCALL*) and **setsockopt**(*MRT_DEL_BW_UPCALL*). Each **setsockopt**(*MRT_ADD_BW_UPCALL*) installs a filter in the kernel for the source and destination address in the *bw_upcall* argument, and that filter will trigger an upcall according to the following pseudo-algorithm:

```
 if (bw_upcall_oper IS ">=") {
    if (((bw_upcall_unit & PACKETS == PACKETS) &&
         (measured_packets >= threshold_packets)) ||
        ((bw_upcall_unit & BYTES == BYTES) &&
         (measured_bytes >= threshold_bytes)))
       SEND_UPCALL("measured bandwidth is >= threshold");
  }
  if (bw_upcall_oper IS "<=" && measured_interval >= threshold_interval) {
    if (((bw_upcall_unit & PACKETS == PACKETS) &&
         (measured_packets <= threshold_packets)) ||
        ((bw_upcall_unit & BYTES == BYTES) &&
         (measured_bytes <= threshold_bytes)))
       SEND_UPCALL("measured bandwidth is <= threshold");
  }
```

In the same *bw_upcall* the unit can be specified in both BYTES and PACKETS. However, the GEQ and LEQ flags are mutually exclusive.

Basically, an upcall is delivered if the measured bandwidth is >= or <= the threshold bandwidth (within the specified measurement interval). For practical reasons, the smallest value for the measurement interval is 3 seconds. If smaller values are allowed, then the bandwidth estimation may be less accurate, or the potentially very high frequency of the generated upcalls may introduce too much overhead. For the >= operation, the answer may be known before the end of *threshold_interval*, therefore the upcall may be delivered earlier. For the <= operation however, we must wait until the threshold interval has expired to know the answer.

Example of usage:

```
struct bw_upcall bw_upcall;
/* Assign all bw_upcall fields as appropriate */
memset(&bw_upcall, 0, sizeof(bw_upcall));
memcpy(&bw_upcall.bu_src, &source, sizeof(bw_upcall.bu_src));
memcpy(&bw_upcall.bu_dst, &group, sizeof(bw_upcall.bu_dst));
bw_upcall.bu_threshold.b_data = threshold_interval;
bw_upcall.bu_threshold.b_packets = threshold_packets;
bw_upcall.bu_threshold.b_bytes = threshold_bytes;
if (is_threshold_in_packets)
    bw_upcall.bu_flags |= BW_UPCALL_UNIT_PACKETS;
if (is_threshold_in_bytes)
    bw_upcall.bu_flags |= BW_UPCALL_UNIT_BYTES;
do {
    if (is_geq_upcall) {
        bw_upcall.bu_flags |= BW_UPCALL_GEQ;
        break;
    }
    if (is_leq_upcall) {
        bw_upcall.bu_flags |= BW_UPCALL_LEQ;
        break;
    }
    return (ERROR);
```

```
} while (0);
setsockopt(mrouter_s4, IPPROTO_IP, MRT_ADD_BW_UPCALL,
          (void *)&bw_upcall, sizeof(bw_upcall));
```

To delete a single filter, then use `MRT_DEL_BW_UPCALL`, and the fields of bw_upcall must be set exactly same as when `MRT_ADD_BW_UPCALL` was called.

To delete all bandwidth filters for a given (S,G), then only the *bu_src* and *bu_dst* fields in *`struct bw_upcall`* need to be set, and then just set only the `BW_UPCALL_DELETE_ALL` flag inside field *bw_upcall.bu_flags*.

The bandwidth upcalls are received by aggregating them in the new upcall message:

```
#define IGMPMSG_BW_UPCALL  4  /* BW monitoring upcall */
```

This message is an array of *`struct bw_upcall`* elements (up to `BW_UPCALLS_MAX` = 128). The upcalls are delivered when there are 128 pending upcalls, or when 1 second has expired since the previous upcall (whichever comes first). In an *`struct upcall`* element, the *bu_measured* field is filled-in to indicate the particular measured values. However, because of the way the particular intervals are measured, the user should be careful how *bu_measured.b_time* is used. For example, if the filter is installed to trigger an upcall if the number of packets is >= 1, then *bu_measured* may have a value of zero in the upcalls after the first one, because the measured interval for >= filters is "clocked" by the forwarded packets. Hence, this upcall mechanism should not be used for measuring the exact value of the bandwidth of the forwarded data. To measure the exact bandwidth, the user would need to get the forwarded packets statistics with the **`ioctl`**(*`SIOCGETSGCNT`*) mechanism (see the **Programming Guide** section) .

Note that the upcalls for a filter are delivered until the specific filter is deleted, but no more frequently than once per *bu_threshold.b_time*. For example, if the filter is specified to deliver a signal if bw >= 1 packet, the first packet will trigger a signal, but the next upcall will be triggered no earlier than *bu_threshold.b_time* after the previous upcall.

SEE ALSO

`getsockopt`(2), `recvfrom`(2), `recvmsg`(2), `setsockopt`(2), `socket`(2), `icmp6`(4), `inet`(4), `inet6`(4), `intro`(4), `ip`(4), `ip6`(4), `pim`(4)

AUTHORS

The original multicast code was written by David Waitzman (BBN Labs), and later modified by the following individuals: Steve Deering (Stanford), Mark J. Steiglitz (Stanford), Van Jacobson (LBL), Ajit Thyagarajan (PARC), Bill Fenner (PARC). The IPv6 multicast support was implemented by the KAME project (`http://www.kame.net`), and was based on the IPv4 multicast code. The advanced multicast API and the multicast bandwidth monitoring were implemented by Pavlin Radoslavov (ICSI) in collaboration with Chris Brown (NextHop).

This manual page was written by Pavlin Radoslavov (ICSI).

NAME

nadb — protocol abstraction and device discovery for the Apple Desktop Bus

SYNOPSIS

```
nadb* at cuda?
nadb* at pmu?
adbkbd* at nadb? console ? mux 1
adbms* at nadb?
adbbt* at nadb?
```

DESCRIPTION

The **nadb** driver provides an abstract interface for talking to ADB devices. It also scans the bus during startup, resolves address conflicts and attaches drivers for known ADB hardware.

SEE ALSO

adbbt(4), adbkbd(4), adbms(4), cuda(4), pmu(4)

NAME

nca — NCR-5380/NCR-53C400 SCSI driver

SYNOPSIS

```
nca0     at isa? port 0x360 irq 15   # Port-mapped NCR 53C80 controller
nca1     at isa? iomem 0xd8000 irq 5 # Memory-mapped controller (T128...)
scsibus* at nca?
```

DESCRIPTION

The **nca** driver provides support for the NCR-5380/NCR-53C400 SCSI controllers.

SEE ALSO

isa(4), scsi(4)

HISTORY

The **nca** driver appeared in NetBSD 1.4.

NAME

ncrsc — NCR 53c710 SCSI I/O Processor

SYNOPSIS

```
ncrsc0 at pcctwo? ipl 2
```

DESCRIPTION

The **ncrsc** driver provides an abstraction layer between the SCSI hardware fitted to Motorola MVME167 and MVME177 single board computers (NCR73C710), and the machine independent SCSI drivers described in scsibus(4).

In addition to sending the required SCSI commands to target devices on the SCSI bus, the **ncrsc** driver deals with DMA, device interrupts, sync/async negotiation and target disconnects/reconnects.

DIAGNOSTICS

Too many to mention.

SEE ALSO

pcctwo(4), scsibus(4)

BUGS

The current **ncrsc** driver should be obsoleted and replaced with a machine independent version.

NAME

ndis — NDIS miniport driver wrapper

SYNOPSIS

ndis* at pci? dev ? function ?

DESCRIPTION

The **ndis** wrapper is designed to allow binary Windows® NDIS miniport network drivers to be used with NetBSD. The **ndis** driver is provided in source code form (`sys/dev/if_ndis`) and must be combined with the Windows® driver binary supplied with your network adapter. The **ndis** driver uses the ndisapi kernel subsystem to relocate and link the Windows® binary so that it can be used in conjunction with native code. The ndisapi subsystem provides an interface between the NDIS API and the NetBSD networking infrastructure. The Windows® driver is essentially fooled into thinking it is running on Windows®. Note that this means the **ndis** driver is only useful on x86 machines.

To build a functional driver, the user must have a copy of the driver distribution media for his or her card. From this distribution, the user must extract two files: the `.SYS` file containing the driver binary code, and its companion `.INF` file, which contains the definitions for driver-specific registry keys and other installation data such as device identifiers. These two files are converted into a `ndis_driver_data.h` file using the `ndiscvt`(8) utility. The resulting file contains a binary image of the driver plus registry key data. The `ndis_driver_data.h` is included in several files located in `sys/dev/if_ndis` so it is needed to compile an NDIS kernel. When the **ndis** driver is loaded, it will create `sysctl`(3) nodes for each registry key extracted from the `.INF` file.

The **ndis** wrapper is designed to support mainly Ethernet and wireless network devices/drivers with PCI bus attachments. It can support many different media types and speeds.

One limitation however, is that there is no consistent way to learn if an Ethernet device is operating in full or half duplex mode. The NDIS API allows for a generic means for determining link state and speed, but not the duplex setting. There may be driver-specific registry keys to control the media setting which can be configured via the `sysctl`(8) command.

EXAMPLES

Assuming you have obtained the appropriate `.INF` and `.SYS` files for your device, the `ndis_driver_data.h` file can be built as follows:

```
$ ndiscvt -i your_card.INF -s your_card.SYS -o ndis_driver_data.h
```

Next uncomment the following lines from `sys/arch/i386/conf/GENERIC`

```
$ cd /usr/src/sys/arch/i386/conf
$ cp GENERIC NDIS
$ vi NDIS
  .
  .
  .
options         COMPAT_NDIS # NDIS network driver
  .
  .
  .
ndis*   at pci? dev ? function ? # Experimental - NDIS Network Driver
```

Configure your kernel, then copy the `ndis_driver_data.h` file into the kernel build directory before compiling the kernel:

```
$ cd src/sys/arch/i386/conf
$ config NDIS
$ cd ../compile/NDIS
$ cp /path/to/ndis_driver_data.h .
$ make depend && make
$ su
$ mv /netbsd /onetbsd
$ cp netbsd /netbsd
$ reboot
```

The ndis0 device should be detected at boot:

```
Matching vendor: 14e4, product: 4324, name: Dell TrueMobile 1400 Dual Band WLAN
ndis0 at pci2 dev 3 function 0
```

The device can then be configured with `ifconfig`(8):

```
ndis0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ssid WWUwireless
        powersave on (100ms sleep)
        chan 6
        address: 00:90:4b:69:94:f0
        media: IEEE802.11 autoselect
        status: no network
        inet 140.160.129.226 netmask 0xfffffc00 broadcast 140.160.131.255
        inet6 fe80::290:4bff:fe69:94f0%ndis0 prefixlen 64 scopeid 0x3
```

If the **`ndis`** driver creates any sysctl nodes, they can be viewed and altered with `sysctl`(8):

```
$ sysctl ndis0
 .
 .
 .
ndis0.ndis_10280001 = Dell TrueMobile 1300 WLAN Mini-PCI Card
ndis0.ndis_Environment = 1
ndis0.ndis_NdisVersion = 0x00050001
ndis0.ndis_BusType = 5
 .
 .
 .
```

DIAGNOSTICS

ndis%d: watchdog timeout A packet was queued for transmission and a transmit command was issued, however the device failed to acknowledge the transmission before a timeout expired.

SEE ALSO

`arp`(4), `netintro`(4), `ifconfig`(8), `ndiscvt`(8)

NDIS 5.1 specification, http://www.microsoft.com.

HISTORY

The **`ndis`** driver wrapper first appeared in FreeBSD 5.3 and was ported to NetBSD 4.0.

AUTHORS

The **ndis** driver was written by Bill Paul ⟨wpaul@windriver.com⟩. It was ported to NetBSD by
Alan Ritter ⟨rittera@NetBSD.org⟩ with help from
Phil Nelson ⟨phil@NetBSD.org⟩ as part of Google's Summer of Code 2005.

NAME

ne — NE2000 and compatible Ethernet cards device driver

SYNOPSIS

```
ne0 at isa? port 0x300 irq 9
ne* at isapnp?
ne* at mca? slot ?
ne* at pci? dev ? function ?
ne* at pcmcia? function ?
```

DESCRIPTION

The **ne** device driver supports NE2000 and compatible (including NE1000) Ethernet cards.

MEDIA SELECTION

The Realtek 8019 (ISA, ISAPnP, some PCMCIA) and Realtek 8029 (PCI) NE2000-compatible Ethernet chips include support for software media selection. If one of these chips is detected by the driver, the list of supported media will be displayed.

For all other chips supported by the **ne** driver, media selection must be performed either via card jumper settings or by vendor-supplied configuration programs.

DIAGNOSTICS

ne0: where did the card go? The driver found the card, but was unable to make the card respond to complete the configuration sequence.

SEE ALSO

`ifmedia`(4), `intro`(4), `isa`(4), `isapnp`(4), `mca`(4), `pci`(4), `pcmcia`(4), `ifconfig`(8)

NAME

nell — sbus pcmcia bridge

SYNOPSIS

only needed for 32bit sparcs (remove the options line on sparc64):
options FULL_SPARC_BUS_SPACE
nell* **at sbus? slot ? offset ? flags 0**
pcmcia* **at nell? controller ? socket ?**

DESCRIPTION

The **nell** is a pcmcia bridge for the sbus. It is also known as STP4020, the name of the chipset used to implement it, and has SUN part number 501-2367.

The firmware assigns two interrupt levels to the nell, but the driver only needs a single interrupt. Depending on distribution of interrupt levels on your machine you might prefer the driver to use either the first (use flags value 0) or the second (use flags value 1).

KERNEL THREAD

Each **nell** instance creates a kernel thread, named like the instance. This thread is used to watch for card insertions and removals, and handling the attachment and initialization of the card's driver.

SEE ALSO

pcmcia(4), sbus(4)

NAME

neo — NeoMagic MagicMedia 256 audio device driver

SYNOPSIS

```
neo*   at pci? dev ? function ?
audio* at audiobus?
```

DESCRIPTION

The **neo** driver provides support for the NeoMagic MagicMedia 256AV and 256ZX AC'97 audio devices, found on many laptops.

The MagicMedia 256AV also comes in a variant (usually found on Dell and HP laptops) that works in Windows Sound System emulation mode, not in AC'97 mode. That variant of the chip must be used with the wss(4) driver. The **neo** driver will not attach to such chips.

SEE ALSO

ac97(4), audio(4), intro(4), midi(4), pci(4), wss(4)

HISTORY

The **neo** device driver appeared in NetBSD 1.5.1.

BUGS

The MagicMedia 256 series is not well-documented.

No MIDI or FM synthesizer capability is provided with the MagicMedia 256 in AC'97 mode. While those capabilities are provided by the Windows driver for the chip, they are emulated by the Windows driver, and not directly supported by the hardware.

NAME

neptune — Neptune-X ISA bridge driver

SYNOPSIS

```
neptune0 at intio0 addr 0xece000 irq 239
ne0 at neptune? addr 0x300
```

DESCRIPTION

The Neptune-X is a Ethernet interface card initially designed by Hirofumi Shimada, which uses popular NE2000 or its clone with its own x68k-proprietary bus to ISA bus bridge.

The NetBSD **neptune** driver takes charge of the ISA bridge part of the Neptune-X. It is implemented as a more generic 'bus' with its own bus_space(9) interface, and is intended to be used with ne(4) driver.

SEE ALSO

intro(4), isa(4), ne(4), bus_space(9)

HISTORY

The **neptune** device appeared in NetBSD 1.4.

BUGS

neptune itself is always detected when it is specified in the kernel config file. This is because the Neptune-X ISA bridge is transparent to software. The attached device is detected appropriately.

NAME

netdock — Ethernet driver for Asante NetDock and Newer Ether MicroDock

SYNOPSIS

netdock* at nubus?

DESCRIPTION

The **netdock** interface provides access to a 10 Mb/s Ethernet network via the Asante NetDock and Newer Ether MicroDock, for PowerBook Duo series.

Each of the host's network addresses is specified at boot time with an SIOCSIFADDR ioctl(2). The **netdock** interface employs the address resolution protocol described in arp(4) to dynamically map between Internet and Ethernet addresses on the local network.

HARDWARE

The **netdock** interface is currently known to support the following interfaces for PowerBook Duo series:

Asante NetDock
Newer Ether MicroDock

DIAGNOSTICS

netdock%d at nubus%d: address %s, type %s %dKB memory. This is a normal autoconfiguration message noting the 6 byte physical Ethernet address of the adapter, its manufacturer, and how much buffer memory it has.

SEE ALSO

arp(4), inet(4), netintro(4), ifconfig(8)

HISTORY

The **netdock** interface first appeared in NetBSD 2.0.

NAME

netintro — introduction to networking facilities

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <net/route.h>
#include <net/if.h>
```

DESCRIPTION

This section is a general introduction to the networking facilities available in the system. Documentation in this part of section 4 is broken up into three areas: *protocol families* (domains), *protocols*, and *network interfaces*.

All network protocols are associated with a specific *protocol family*. A protocol family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. A protocol family may support multiple methods of addressing, though the current protocol implementations do not. A protocol family normally comprises a number of protocols, one per `socket`(2) type. It is not required that a protocol family support all socket types. A protocol family may contain multiple protocols supporting the same socket abstraction.

A protocol supports one of the socket abstractions detailed in `socket`(2). A specific protocol may be accessed either by creating a socket of the appropriate type and protocol family, or by requesting the protocol explicitly when creating a socket. Protocols normally accept only one type of address format, usually determined by the addressing structure inherent in the design of the protocol family/network architecture. Certain semantics of the basic socket abstractions are protocol specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide non-standard facilities or extensions to a mechanism. For example, a protocol supporting the `SOCK_STREAM` abstraction may allow more than one byte of out-of-band data to be transmitted per out-of-band message.

A network interface is similar to a device interface. Network interfaces comprise the lowest layer of the networking subsystem, interacting with the actual transport hardware. An interface may support one or more protocol families and/or address formats. The *SYNOPSIS* section of each network interface entry gives a sample specification of the related drivers for use in providing a system description to the `config`(1) program.

The *DIAGNOSTICS* section lists messages which may appear on the console and/or in the system error log, `/var/log/messages` (see `syslogd`(8)), due to errors in device operation.

PROTOCOLS

The system currently supports the Internet protocols and some of the ISO OSI protocols. Raw socket interfaces are provided to the IP protocol layer of the Internet, and to the IDP protocol of Xerox NS. Consult the appropriate manual pages in this section for more information regarding the support for each protocol family.

ADDRESSING

Associated with each protocol family is an address format. All network address adhere to a general structure, called a sockaddr, described below. However, each protocol imposes finer and more specific structure, generally renaming the variant, which is discussed in the protocol family manual page alluded to above.

```
struct sockaddr {
        u_char  sa_len;
        u_char  sa_family;
        char    sa_data[14];
```

```
};
```

The field *sa_len* contains the total length of the of the structure, which may exceed 16 bytes. The following address values for *sa_family* are known to the system (and additional formats are defined for possible future implementation):

```
#define    AF_LOCAL     1    /* local to host (pipes, portals) */
#define    AF_INET      2    /* internetwork: UDP, TCP, etc. */
#define    AF_NS        6    /* Xerox NS protocols */
#define    AF_CCITT     10   /* CCITT protocols, X.25 etc */
#define    AF_HYLINK    15   /* NSC Hyperchannel */
#define    AF_ISO       18   /* ISO protocols */
```

ROUTING

UNIX provides some packet routing facilities. The kernel maintains a routing information database, which is used in selecting the appropriate network interface when transmitting packets.

A user process (or possibly multiple co-operating processes) maintains this database by sending messages over a special kind of socket. This supplants fixed size `ioctl`(2) used in earlier releases.

This facility is described in `route`(4).

INTERFACES

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, `lo`(4), do not.

The following `ioctl`(2) calls may be used to manipulate network interfaces. The `ioctl`(2) is made on a socket (typically of type `SOCK_DGRAM`) in the desired domain. Most of the requests supported in earlier releases take an *ifreq* structure as its parameter. This structure has the form

```
struct  ifreq {
#define    IFNAMSIZ    16
    char    ifr_name[IFNAMSIZ];         /* if name, e.g. "en0" */
    union {
        struct    sockaddr ifru_addr;
        struct    sockaddr ifru_dstaddr;
        struct    sockaddr ifru_broadaddr;
        short     ifru_flags;
        int       ifru_metric;
        void   *ifru_data;
    } ifr_ifru;
#define ifr_addr      ifr_ifru.ifru_addr    /* address */
#define ifr_dstaddr   ifr_ifru.ifru_dstaddr /* other end of p-to-p link */
#define ifr_broadaddr ifr_ifru.ifru_broadaddr /* broadcast address */
#define ifr_flags     ifr_ifru.ifru_flags   /* flags */
#define ifr_metric    ifr_ifru.ifru_metric  /* metric */
#define ifr_data      ifr_ifru.ifru_data    /* for use by interface */
};
```

Calls which are now deprecated are:

`SIOCSIFADDR` Set interface address for protocol family. Following the address assignment, the ‘‘initialization’’ routine for the interface is called.

`SIOCSIFDSTADDR` Set point to point address for protocol family and interface.

`SIOCSIFBRDADDR` Set broadcast address for protocol family and interface.

`ioctl`(2) requests to obtain addresses and requests both to set and retrieve other data are still fully supported and use the *`ifreq`* structure:

`SIOCGIFADDR` Get interface address for protocol family.

`SIOCGIFDSTADDR` Get point to point address for protocol family and interface.

`SIOCGIFBRDADDR` Get broadcast address for protocol family and interface.

`SIOCSIFFLAGS` Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified; some interfaces may be reset so that incoming packets are no longer received. When marked up again, the interface is reinitialized.

`SIOCGIFFLAGS` Get interface flags.

`SIOCSIFMETRIC` Set interface routing metric. The metric is used only by user-level routers.

`SIOCGIFMETRIC` Get interface metric.

There are two requests that make use of a new structure:

`SIOCAIFADDR` An interface may have more than one address associated with it in some protocols. This request provides a means to add additional addresses (or modify characteristics of the primary address if the default address for the address family is specified). Rather than making separate calls to set destination or broadcast addresses, or network masks (now an integral feature of multiple protocols) a separate structure, *`ifaliasreq`*, is used to specify all three facets simultaneously (see below). One would use a slightly tailored version of this struct specific to each family (replacing each sockaddr by one of the family-specific type). Where the sockaddr itself is larger than the default size, one needs to modify the `ioctl`(2) identifier itself to include the total size, as described in `ioctl`(2).

`SIOCDIFADDR` This requests deletes the specified address from the list associated with an interface. It also uses the *`ifaliasreq`* structure to allow for the possibility of protocols allowing multiple masks or destination addresses, and also adopts the convention that specification of the default address means to delete the first address for the interface belonging to the address family in which the original socket was opened.

Request making use of the *`ifconf`* structure:

`SIOCGIFCONF` Get interface configuration list. This request takes an *`ifconf`* structure (see below) as a value-result parameter. The *`ifc_len`* field should be initially set to the size of the buffer pointed to by *`ifc_buf`*. On return it will contain the length, in bytes, of the configuration list.

```
/*
* Structure used in SIOC[AD]IFADDR request.
*/
struct ifaliasreq {
        char    ifra_name[IFNAMSIZ];   /* if name, e.g. "en0" */
        struct  sockaddr        ifra_addr;
        struct  sockaddr        ifra_dstaddr;
#define ifra_broadaddr  ifra_dstaddr
        struct  sockaddr        ifra_mask;
```

```
};
/*
* Structure used in SIOCGIFCONF request.
* Used to retrieve interface configuration
* for machine (useful for programs which
* must know all networks accessible).
*/
struct ifconf {
    int   ifc_len;            /* size of associated buffer */
    union {
        void    *ifcu_buf;
        struct     ifreq *ifcu_req;
    } ifc_ifcu;
#define ifc_buf ifc_ifcu.ifcu_buf /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req /* array of structures returned */
};
```

SEE ALSO

`config`(1), `ioctl`(2), `socket`(2), `intro`(4), `routed`(8)

HISTORY

The **`netintro`** manual appeared in 4.3BSD−Tahoe.

NAME

newport — SGI NG1 graphics controller

SYNOPSIS

```
newport* at gio? slot ?
wsdisplay* at newport? console ?
```

DESCRIPTION

The **newport** driver supports the SGI NG1 (a.k.a. Indy 8-bit, Indy 24-bit, and XL) graphics controllers, often found on Indy and Indigo2 machines.

SEE ALSO

gio(4), grtwo(4), light(4), wscons(4)

HISTORY

The **newport** driver first appeared in NetBSD 2.0.

NAME

nfe — NVIDIA nForce MCP Ethernet driver

SYNOPSIS

```
nfe* at pci?
ciphy* at mii?
icsphy* at mii?
makphy* at mii?
rlphy* at mii?
```

DESCRIPTION

The **nfe** driver supports PCI Ethernet adapters based on the NVIDIA nForce Media and Communications Processors (MCP), such as the nForce, nForce 2, nForce 3, CK804, MCP04, MCP51, MCP55, MCP61, MCP65, MCP67 and MCP73 Ethernet controller chips.

The **nfe** driver supports the following *media* types:

autoselect	Enable autoselection of the media type and options.
10baseT	Set 10Mbps operation.
100baseTX	Set 100Mbps (Fast Ethernet) operation.
1000baseT	Set 1000Mbps (Gigabit Ethernet) operation (recent models only).

SEE ALSO

arp(4), ciphy(4), icsphy(4), ifmedia(4), intro(4), netintro(4), pci(4), rlphy(4), ifconfig(8)

HISTORY

The **nfe** device driver first appeared in OpenBSD 3.9. It was added to NetBSD 3.1.

AUTHORS

The **nfe** driver was written by Jonathan Gray ⟨jsg@openbsd.org⟩ and Damien Bergamini ⟨damien@openbsd.org⟩.

CAVEATS

NVIDIA refuse to release any documentation on their products.

NAME

nfsmb, **nfsmbc** — NVIDIA nForce 2/3/4 SMBus controller and SMBus driver

SYNOPSIS

```
nfsmbc* at pci? dev ? function ?
nfsmb* at nfsmbc?
```

DESCRIPTION

The **nfsmbc** provides support for the NVIDIA nForce 2/3/4 SMBus controller. The **nfsmbc** has two SMBus (**nfsmb**).

SEE ALSO

pci(4)

HISTORY

The **nfsmb** driver appeared in NetBSD 4.0.

NAME

njata — Workbit NinjaATA-32 CardBus IDE controller driver

SYNOPSIS

```
njata* at cardbus? function ?
njata* at cardbus? function ? flags 0x01  # with wait 0x01
```

DESCRIPTION

The **njata** driver provides support for the following Workbit Bus-Master CardBus IDE controller chips:

NinjaATA-32Bi	CardBus / PCMCIA dual mode IDE controller ("DuoATA"). This controller is mainly used for portable drives. This driver supports the CardBus mode.
NPATA-32	CardBus IDE controller. This controller is widely used for CardBus CompactFlash adapters.

These controllers are capable of bus-mastering for ATA PIO transfer. The **njata** driver uses the bus-mastering PIO transfer unless transfer buffer is unaligned, and significantly reduces CPU usage for PIO-only ATA devices compared with usual PIO transfer.

CONFIGURATION

The optional flags parameter is the "wait" value for ATA transfers. Some combinations of host and device may fail without flags parameter or flags 0x00. In this case try adding wait values like flags 0x01 or flags 0x11 (the wait parameter is composed of two 4bit values). Smaller wait values should be faster. Too long waits may cause transfer errors.

SEE ALSO

ata(4), atapi(4), cardbus(4), intro(4), wd(4), wdc(4)

HISTORY

The **njata** device driver first appeared in NetBSD 4.0.

AUTHORS

ITOH Yasufumi ⟨itohy@NetBSD.org⟩

NAME

njs — Workbit NinjaSCSI-32 PCI/CardBus SCSI driver

SYNOPSIS

```
njs* at pci? dev ? function ?
njs* at cardbus? function ?
```

DESCRIPTION

The **njs** driver provides support for the following Workbit Bus-Master PCI/CardBus Ultra Narrow SCSI controller chips:

NinjaSCSI-32Bi	CardBus / PCMCIA dual mode device (“DuoSCSI”). This driver supports the CardBus mode.
NinjaSCSI-32UDE	PCI / CardBus device with DualEdge transfer (40MB/s max.) capability.

SEE ALSO

cardbus(4), cd(4), ch(4), pci(4), scsi(4), sd(4), st(4), uk(4)

HISTORY

The **njs** device driver first appeared in NetBSD 1.6.3.

AUTHORS

ITOH Yasufumi ⟨itohy@NetBSD.org⟩

BUGS

DualEdge transfer is not currently supported.

NAME

np — Interlan Np100 10 Mb/s Ethernet interface

SYNOPSIS

```
np0 at uba0 csr 166000 vector npintr
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **np** device provides access to an Interlan Np100 Ethernet interface for control functions.

This interface is unusual in that it requires loading firmware into the controller before it may be used as a network link-level interface. This is accomplished by opening a character special device, and writing data to it. It is also possible to do post-mortem debugging of firmware failures by reading the local memory of the device.

Multiple control processes are allowed by opening separate minor devices; secondary interfaces are specified by shifting the interface number by 4 bits.

The device also responds to commands passed through the driver by the following `ioctl`(2)s:

`NPRESET` kills off all active network processes.

`NPSTART` begins execution of the board at the specified address (usually `0x400`).

`NPNETBOOT` downloads the image from a server on the network. [Contact MICOM-INTERLAN for details.]

DIAGNOSTICS

np%d: Bad Maintenance command: %x! An invalid `ioctl`(2) was passed to the np driver.

np%d: Panic NP100 bad buffer chain. An error occurred in an read or write operation causing it to run out of buffers before it finished the operation. This indicates a kernel failure rather than a device failure.

NP100 unit %d not found! A failure occurred during initialization, such that the UNIBUS address expected for the board was found to be bad. Probably indicates hardware problems with the board, as do the following:

> **NP100 Unit %d timed out!**
> **NP100 Unit %d Failed diagnostics!**
> **Status from CSR0: %x.**

Panic from NP100 unit %d!
Panic Message: %s. An occurrence on the board was deemed serious enough to have the VAX print it out.

NP100 unit #%d available! The board was successfully loaded and started.

np%d: Bad Req: %x. The board made a maintenance request to the VAX that it did not understand.

np%d: No more room on Command Queue! The np driver allowed an internal resource to be exhausted. This should never happen.

There are 110 other diagnostic messages that can be enabled by setting bits in a debugging mask. Consult the driver for details.

SEE ALSO

`arp`(4), `inet`(4), `ix`(4), `netintro`(4)

HISTORY

The **np** driver appeared in 4.3BSD.

NAME

npx — Numeric Processing Extension coprocessor and emulator

SYNOPSIS

```
npx0 at isa? port "IO_NPX0" irq 13
npx* at acpi?
npx* at pnpbios? index ?
```

DESCRIPTION

The **npx** driver enables the use of the system's Numeric Processing Extension coprocessor. The **npx** driver is required for proper system functioning regardless of whether or not an NPX is present.

SEE ALSO

acpi(4), isa(4), pnpbios(4)

NAME

nsclpcsio — National Semiconductor PC87366 LPC Super I/O

SYNOPSIS

```
nsclpcsio* at isa?
gpio* at nsclpcsio?
```

DESCRIPTION

The **nsclpcsio** driver provides support for the National Semiconductor PC87366 LPC Super I/O. The Super I/O incorporates several logical devices, the following ones are supported: GPIO, VLM and TMS.

The GPIO logical device provides 29 I/O pins which can be accessed through the gpio(4) framework. The gpioctl(8) program allows easy manipulation of the pins from userland.

VLM and TMS logical devices provides hardware monitoring capabilities to be used with the envsys(4) API. The following 17 monitoring sensors are available:

Sensor	Units	Typical Use
TSENS1	uK	Remote diode
TSENS2	uK	Remote diode
TNSC	uK	Local diode
VSENS0	uV DC	External source
VSENS1	uV DC	External source
VSENS2	uV DC	External source
VSENS3	uV DC	External source
VSENS4	uV DC	External source
VSENS5	uV DC	External source
VSENS6	uV DC	External source
VSB	uV DC	VSB
VDD	uV DC	VDD
VBAT	uV DC	VBAT
AVDD	uV DC	AVDD
TS1	uV DC	Thermistor
TS2	uV DC	Thermistor
TS3	uV DC	Thermistor

SEE ALSO

envsys(4), envstat(8)

HISTORY

The **nsclpcsio** device appeared in NetBSD 2.0.

BUGS

The chip decodes address ranges which are not obvious and cannot be controlled by the kernel configuration file (must be set up by the BIOS).

NAME

nsmb — kernel SMB protocol communicator

SYNOPSIS

```
pseudo-device nsmb
```

DESCRIPTION

This virtual device is used by SMBFS filesystem for actual communication with SMB servers. It provides the physical transportation, encapsulating the networking part of SMBFS. Only SMB over TCP/IP is supported at this moment, SMB over NetBIOS is not supported.

For the SMBFS kernel support to work, it's necessary to have both this pseudo-device and file-system SMBFS configured into kernel.

SEE ALSO

`mount_smbfs`(8)

HISTORY

This driver first appeared in FreeBSD 4.4. In NetBSD, this first appeared in NetBSD 2.0, together with rest of SMBFS support.

AUTHORS

Boris Popov ⟨bp@butya.kz⟩, ⟨bp@FreeBSD.org⟩. NetBSD port done by Matt Debergalis ⟨deberg@NetBSD.org⟩ and Jaromir Dolecek ⟨jdolecek@NetBSD.org⟩.

NAME

nsphy — Driver for National Semiconductor DP83840 10/100 Ethernet PHYs

SYNOPSIS

```
nsphy* at mii? phy ?
```

DESCRIPTION

The **nsphy** driver supports the National Semiconductor DP83840 and DP83840A 10/100 Ethernet PHYs. These PHYs are found on a variety of high-performance Ethernet interfaces.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

nsphyter — Driver for National Semiconductor DP83843 10/100 Ethernet PHYs

SYNOPSIS

```
nsphyter* at mii? phy ?
```

DESCRIPTION

The **nsphyter** driver supports the National Semiconductor DP83843 (PHYTER) 10/100 Ethernet PHY, which is found in a variety of high-performance Ethernet interfaces, and DP83815 (MacPHYTER) internal PHY. The DP83815 is a 10/100 Ethernet chip supported by the sip(4) driver.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

ntwoc — Riscom/N2, N2pci, WANic 400 synchronous serial interfaces

SYNOPSIS

```
ntwoc* at pci? dev ? function ? flags 0
ntwoc0 at isa? port 0x300 irq 5 iomem 0xc8000 flags 1
```

DESCRIPTION

The **ntwoc** device driver supports bit-synchronous serial communication using Cisco HDLC framing. The cards are capable of being driven by the line clock or from an internal baud rate generator. The devices all use the Hitachi hd64570 serial chip. The hd64570 supports 2 asynchronous/byte-synchronous/bit-synchronous serial ports, and has a 4-channel DMA controller for loading the serial port FIFOs.

The ISA Riscom/N2 card has a jumper block to set the IRQ and a DIP switch to set the port address the card will use. The values programmed into the card must be specified with the **port** and **irq** locators in the kernel configuration line. The **iomem** locator must be specified and must occur on a 16k boundary. The driver uses a 16k region of io memory. Bit 0 of the **flags** locator indicates if there is a second serial port available on the card.

Currently clock source and speed information is specified with the **flags** locator in the kernel configuration file. The flags field has the following format.

```
  3                   2                   1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-------------+ +-----+ +-----+ + +---+ +-+     + +---+ +-+   +
      tmc         tdiv    rdiv  e1 rxs1 ts1    e0 rxs0  txs0  np(*)
```

tmc Defines the timer constant. The base clock frequency is divided by *tmc* to generate the main clock for receiving and sending. Further division is possible with the *tdiv* and *rdiv* divisor options. A value of 0 is treated as 256.

tdiv Defines the transmit divisor as 2^(*tdiv*). The internal transmit clock frequency is determined by dividing the base clock frequency by *tmc* and then dividing by 2^(*tdiv*).

rdiv Defines the receive divisor as 2^(*rdiv*). The internal receive clock frequency is determined by dividing the base clock frequency by *tmc* and then dividing by 2^(*rdiv*).

e0 e1 If true the internal clock source is used to drive the line clock for port 0 or port 1 respectively.

rxs0 rxs1 Specifies which clock source to use for receiving data on port 0 and port 1 respectively. The following values are accepted:

0
Line clock.
1
Line clock with noise suppression.
2
Internal clock.

txs0 txs1 Specifies which clock source to use for transmitting data on port 0 and port 1 respectively. The following values are accepted:

0
Line clock.
1
Internal clock.

2
Receive clock.

np (For the ISA card only) A value of 1 indicates there is a second serial port present on the card. This is auto-detected on the PCI card and need not be specified.

HARDWARE

Cards supported by the **ntwoc** driver include:

SDL Communications Riscom/N2
SDL Communications N2pci
SDL Communications WANic 400 (untested)

DIAGNOSTICS

ntwoc0: TXDMA underrun - fifo depth maxed Indicates that the serial port's FIFO is being drained faster than DMA can fill it. The driver automatically increases the low-water mark at which to begin DMA transfers when underruns occur. This diagnostic is issued when the low-water mark is maximized (i.e., 1 less than the depth of the FIFO).

ntwoc0: RXDMA buffer overflow Indicates that a frame is being received by the card, but there are no free receive buffers.

SEE ALSO

intro(4), isa(4), pci(4), ifconfig(8)

HISTORY

The PCI driver first appeared in NetBSD 1.4. Much of the ISA driver was adapted from the FreeBSD **sr** driver and first appeared in NetBSD 1.5.

BUGS

Use of the **flags** locator for setting the clock sources and speeds should be replaced with ioctl's and a control program.

NAME

null — the null device

DESCRIPTION

The **null** device accepts and reads data as any ordinary (and willing) file − but throws it away. The length of the **null** device is always zero.

FILES

`/dev/null`

HISTORY

A **null** device appeared in Version 7 AT&T UNIX.

NAME

nvram — PReP nvram interface

SYNOPSIS

```
#include <machine/nvram.h>
```

DESCRIPTION

The file **/dev/nvram** is an interface to the PReP NVRAM, including the Global Environment Area. This interface is highly stylized; ioctls are used for all operations. These ioctls refer to individual variables in the Global Environment Area and thier values.

The calls that take and/or return a variable use a pointer to an `int` variable for this purpose; others use a pointer to an `struct pnviocdesc` descriptor, which contains a variable and two counted strings. The first string comprises the fields `pnv_namelen` (an `int`) and `pnv_name` (a `char *`), giving the name of a field. The second string comprises the fields `pnv_buflen` and `pnv_buf`, used analogously. These two counted strings work in a "value-result" fashion. At entry to the ioctl, the counts are expected to reflect the buffer size; on return, the counts are updated to reflect the buffer contents.

The following ioctls are supported:

PNVIOCGETNEXTNAME
: Takes a variable name and returns the name of the following variable. If a `NULL` is passed as the variable name, the first variable name will be returned. If the last variable is given as an argument, the ioctl will return `EINVAL`.

`PNVIOCGET`
: Fills in the value of the named property for the given variable. If no such property is associated with that variable, the value length is set to −1. If the named property exists but has no value, the value length is set to 0.

`PNVIOCSET`
: Writes the given value under the given name.

`PNVIOCGETNUMGE`
: Returns the number of variables in the Global Environment Area.

FILES

`/dev/nvram`

ERRORS

The following may result in rejection of an operation:

`[EINVAL]`
: The given variable name does not exist.

`[EINVAL]`
: The buffer set up to retrieve values from the nvram was not large enough to hold the result.

SEE ALSO

`ioctl`(2)

`PowerPC Reference Platform Specification Version 1.1, Section 5.5`

BUGS

Due to limitations within the **nvram** itself, these functions run at elevated priority and may adversely affect system performance.

`PNVIOCSET` is not currently supported, making the **nvram** driver read-only at this time.

NAME

oak — Oak SCSI I Card device interface

SYNOPSIS

```
oak0 at podulebus?
```

DESCRIPTION

The **oak** interface provides access to Oak SCSI I interfaces.

SEE ALSO

asc(4), cosc(4), csc(4), podulebus(4), ptsc(4)

NAME

obio — introduction to Macintosh On-Board IO bus support and drivers

SYNOPSIS

```
obio0 at mainbus?
```

DESCRIPTION

The **obio** interface serves as an abstraction used by the autoconfiguration system to help find and attach devices (e.g. the Ethernet or disk controllers) connected to the Macintosh onboard I/O bus.

HARDWARE

NetBSD includes machine-dependent On-Board drivers, sorted by device type and driver name:

SCSI interfaces

esp	NCR 53C9x SCSI interfaces.
ncrscsi	NCR 5380 SCSI interface.

Disk and tape controllers

iwm	Integrated Woz Machine - Sony based floppy drives.
wdc	Standard IDE/ATAPI type hard drive controllers.

Network interfaces

mc	Apple MACE ethernet interface.
sn	Sonic (DP83932, DP83916) based ethernet interfaces.

Serial interfaces

zsc	Zilog 8530 serial communications interfaces.

Audio devices

asc	Apple Sound Chip as found on 68k based Macintosh computers.

Miscellaneous devices

adb	Apple Desktop Bus for keyboards, mice, and other input devices.
intvid	Internal video hardware.

SEE ALSO

adb(4), asc(4), autoconf(4), esp(4), intro(4), intvid(4), iwm(4), mc(4), ncrscsi(4), sn(4), wdc(4), zsc(4)

HISTORY

obio first appeared in NetBSD 1.2.

NAME

obio — introduction to Macintosh On-Board IO bus support and drivers

SYNOPSIS

```
obio0 at pci? dev ? function ?
```

DESCRIPTION

The **obio** interface serves as an abstraction used by the autoconfiguration system to help find and attach devices (e.g. the Ethernet or disk controllers) connected to the Macintosh onboard I/O bus.

HARDWARE

NetBSD includes machine-dependent On-Board drivers, sorted by device type and driver name:

SCSI interfaces

esp	NCR 53C9x SCSI interfaces.
mesh	Apple Macintosh Enhanced SCSI Hardware SCSI interfaces.

Disk and tape controllers

wdc	Standard IDE/ATAPI type hard drive controllers.
mediabay	Standard IDE/ATAPI type CD-ROM drive controllers in PowerBooks.

Network interfaces

bm	Apple BMac ethernet interface.
mc	Apple MACE ethernet interface.
gem	GMAC ethernet interface.
wi	WaveLAN/IEEE and PRISM-II 802.11 wireless interfaces.

Serial interfaces

zsc	Zilog 8530 serial communications interfaces.

Audio devices

awacs	Apple's 'audio waveform amplifier and converter for sound' audio device found on most macppc models. May not work on several G4 and iBook models.

Miscellaneous devices

adb	Apple Desktop Bus for keyboards, mice, and other input devices.
nvram	Placeholder device for the persistant system settings.

SEE ALSO

adb(4), autoconf(4), awacs(4), bm(4), esp(4), gem(4), intro(4), mc(4), mediabay(4), mesh(4), wdc(4), wi(4), zsc(4)

HISTORY

obio first appeared in NetBSD 1.2.

NAME

oboe — Toshiba OBOE IrDA SIR/FIR driver

SYNOPSIS

```
oboe*    at pci? dev ? function ?
irframe* at oboe?
```

DESCRIPTION

The **oboe** driver provides support for the Toshiba Oboe IrDA chip.

Access to the device is through the `irframe`(4) driver.

SEE ALSO

`irframe`(4), `pci`(4)

HISTORY

The **oboe** driver appeared in NetBSD 1.6.

AUTHORS

Jan Sparud ⟨jan@sparud.net⟩

NAME

ofisa — introduction to machine-independent Open Firmware ISA bus support and drivers

SYNOPSIS

```
ofisa* at mainbus0
XX*    at ofisa?
```

DESCRIPTION

NetBSD includes a machine-independent ISA bus subsystem and several machine-independent ISA device drivers. The **ofisa** bus uses Open Firmware to determine how to attach devices.

HARDWARE

NetBSD includes machine-independent Open Firmware ISA drivers, sorted by device type and driver name:

Disk and tape controllers

wdc Standard Western Digital type hard drive controllers: MFM, RLL, ESDI, and IDE/ATAPI.

Serial and parallel interfaces

com NS8250-, NS16450-, and NS16550-based serial ports.

lpt Standard ISA parallel port interface.

Network interfaces

cs Cirrus Logic Crystal CS8900 Ethernet interfaces.

Sound cards and MIDI interfaces

ess ESS Technology AudioDrive 1788-, 1888-, 1887-, and 888-based sound cards.

Miscellaneous devices

joy Game (joystick) adapters.

SEE ALSO

com(4), cs(4), ess(4), intro(4), isa(4), joy(4), lpt(4), wdc(4)

HISTORY

The Open Firmware ISA subsystem appeared in NetBSD 1.4.

NAME

ohci — USB Open Host Controller driver

SYNOPSIS

```
ohci*    at cardbus? function ?
ohci*    at pci? dev ? function ?
usb*     at ohci?
```

DESCRIPTION

The **ohci** driver provides support for USB Open Host Controller Interface.

SEE ALSO

cardbus(4), pci(4), usb(4)

HISTORY

The **ohci** driver appeared in NetBSD 1.4.

NAME

onewire — 1-Wire bus

SYNOPSIS

```
onewire* at gpioow?

option ONEWIREVERBOSE
```

DESCRIPTION

1-Wire bus was originally developed by Dallas Semiconductor for connecting integrated circuits. It is commonly used for connecting devices such as electronic keys, EEPROMs, temperature sensors, real-time clocks, security chips, etc.

The **onewire** driver provides a uniform programming interface layer between 1-Wire master controllers and various 1-Wire slave devices. Each 1-Wire master controller attaches a **onewire** framework; several slave devices can then be attached to the **onewire** bus.

The driver supports plugging and unplugging slave devices on the fly.

SUPPORTED MASTERS

`gpioow`(4) 1-Wire bus bit-banging through GPIO pin

SUPPORTED SLAVES

`owid`(4) ID family type device
`owtemp`(4) temperature family type device

SEE ALSO

`intro`(4)

HISTORY

The **onewire** driver first appeared in OpenBSD 4.0 and NetBSD 4.0.

AUTHORS

The **onewire** driver was written by Alexander Yurchenko ⟨grange@openbsd.org⟩ and ported to NetBSD by Jeff Rizzo ⟨riz@NetBSD.org⟩.

NAME

oosiop — Symbios/NCR 53C700 SCSI driver

SYNOPSIS

arc

```
oosiop* at jazzio?
```

hp700

```
oosiop0 at gsc?

scsibus* at oosiop?
```

DESCRIPTION

The **oosiop** driver provides support for the Symbios/NCR 53C700 SCSI controller chip.

For the Symbios/NCR 53C710 SCSI host adapters, use the osiop(4) driver.

For the Symbios/NCR 53C8xx PCI SCSI host adapters, use the siop(4) or esiop(4) driver.

SEE ALSO

cd(4), ch(4), esiop(4), intro(4), osiop(4), scsi(4), sd(4), siop(4), ss(4), st(4), uk(4), scsipi(9)

HISTORY

oosiop driver first appeared in NetBSD 2.0.

AUTHORS

The **oosiop** driver was originally written by Shuichiro URATA for the arc port. Izumi Tsutsui modified the driver to make it really machine independent for hp700.

NAME

openprom — Sun OPENPROM and EEPROM interface

SYNOPSIS

```
#include <machine/openpromio.h>
```

DESCRIPTION

The file **/dev/openprom** is an interface to the SPARC OPENPROM, including the EEPROM area. This interface is highly stylized; ioctls are used for all operations. These ioctls refer to "nodes", which are simply "magic" integer values describing data areas. Occasionally the number 0 may be used or returned instead, as described below. A special distinguished "options" node holds the EEPROM settings.

The calls that take and/or return a node use a pointer to an `int` variable for this purpose; others use a pointer to an `struct opiocdesc` descriptor, which contains a node and two counted strings. The first string comprises the fields `op_namelen` (an `int`) and `op_name` (a `char *`), giving the name of a field. The second string comprises the fields `op_buflen` and `op_buf`, used analogously. These two counted strings work in a "value-result" fashion. At entry to the ioctl, the counts are expected to reflect the buffer size; on return, the counts are updated to reflect the buffer contents.

The following ioctls are supported:

`OPIOCGETOPTNODE` Takes nothing, and fills in the options node number.

OPIOCGETNEXT Takes a node number and returns the number of the following node. The node following the last node is number 0; the node following number 0 is the first node.

`OPIOCGETCHILD` Takes a node number and returns the number of the first "child" of that node. This child may have siblings; these can be discovered by using `OPIOCGETNEXT`.

`OPIOCGET` Fills in the value of the named property for the given node. If no such property is associated with that node, the value length is set to -1. If the named property exists but has no value, the value length is set to 0.

`OPIOCSET` Writes the given value under the given name. The OPENPROM may refuse this operation; in this case `EINVAL` is returned.

`OPIOCNEXTPROP` Finds the property whose name follows the given name in OPENPROM internal order. The resulting name is returned in the value field. If the named property is the last, the "next" name is the empty string. As with `OPIOCGETNEXT`, the next name after the empty string is the first name.

FILES

`/dev/openprom`

ERRORS

The following may result in rejection of an operation:

`[EINVAL]` The given node number is not zero and does not correspond to any valid node, or is zero where zero is not allowed.

`[EBADF]` The requested operation requires permissions not specified at the call to **open**().

`[ENAMETOOLONG]` The given name or value field exceeds the maximum allowed length (8191 bytes).

SEE ALSO

`ioctl`(2)

`http://playground.sun.com/1275/`

BUGS

Due to limitations within the **openprom** itself, these functions run at elevated priority and may adversely affect system performance.

The Sun **openprom** is what became the Open Firmware (IEEE 1275) standard for processor and system independent boot firmware.

NAME

opl — Yamaha OPL2 and OPL3 FM MIDI synthesizer driver

SYNOPSIS

```
opl*  at cmpci? flags 1
opl*  at esl?
opl*  at eso?
opl*  at ess?
opl*  at fms?
opl0  at isa? port 0x388
opl*  at sb?
opl*  at sv?
opl*  at wss?
opl*  at yds?
opl*  at ym?
midi* at opl?
```

DESCRIPTION

The **opl** driver provides support for the Yamaha OPL2 (YM3812) and OPL3 (YMF262) chips. The chips are FM synthesizers and are capable of producing a wide range of sounds.

Access to the device is through the MIDI driver.

The **opl** driver usually attaches to a sound card, but it can also sit directly on the ISA bus.

If “flags 1” is specified, the **opl** driver handles left and right channels of OPL3 swapped. Use this flag if your device has such problem.

SEE ALSO

cmpci(4), esl(4), eso(4), ess(4), fms(4), isa(4), midi(4), sb(4), sv(4), wss(4), yds(4), ym(4)

HISTORY

The **opl** device driver appeared in NetBSD 1.4.

BUGS

The OPL3 chip is operated in OPL2 mode despite being more capable.

NAME

optiide — OPTi IDE disk controllers driver

SYNOPSIS

```
optiide* at pci? dev ? function ? flags 0x0000
```

DESCRIPTION

The **optiide** driver supports the OPTi 82c621, 82c568 and 82d568 IDE controllers, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **optiide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

The timings used for the PIO and DMA modes for controllers listed above are for a PCI bus running at 25 or 33 MHz. This driver may not work properly on overclocked systems.

NAME

options — Miscellaneous kernel configuration options

SYNOPSIS

```
cinclude ...
config ...
[no] file-system ...
ident ...
include ...
[no] makeoptions ...
maxusers ...
[no] options ...
[no] pseudo-device ...
```

DESCRIPTION

This manual page describes a number of miscellaneous kernel configuration options that may be specified in a kernel config file. See config(1) and config(5) for information on how to configure and build kernels.

The *no* form removes a previously specified option.

Keywords

The following keywords are recognized in a kernel configuration file:

cinclude "*filename*"
Conditionally includes another kernel configuration file whose name is *filename*, which may be double-quoted and may be an explicit path or relative to the kernel source directory. Failure to open the named file is ignored.

config *exec_name* root on *rootdev* [type *fstype*] [dumps on *dumpdev*]
Defines a configuration whose kernel executable is named *exec_name*, normally "netbsd", with its root file system of type *fstype* on the device *rootdev*, and optionally specifying the location of kernel core dumps on the device *dumpdev*. *dev* or *dumpdev* and *fstype* may be specified as "?", which is a wild card. The root *fstype* and *dumpdev* are optional and assumed to be wild carded if they are not specified.

device_instance at *attachment* [*locators value* [...]] [flags *value*]
Define an instance of the device driver *device_instance* that attaches to the bus or device named *attachment*. An *attachment* may require additional information on where the device can be found, such as an address, channel, function, offset, and/or slot, referred to as *locators*, whose *value* often may be a wild card, "?". Some device drivers have one or more *flags* that can be adjusted to affect the way they operate.

file-system *fs_name* [, *fs_name* [...]]
Include support for the file-system *fs_name*.

ident "*string*"
Sets the kernel identification string to *string*.

include "*filename*"
Functions the same as *cinclude*, except failure to open *filename* produces a fatal error.

options *option_name* [, *option_name=value* [...]]
Specifies (or sets) the option, or comma-separated list of options, *option_name*. Some options expect to be assigned a value, which may be an integer, a double-quoted word, a bare word, or an empty string ("""). Note that those are eventually handled by the C compiler, so the rules of that language apply.

Note: Options that are not defined by device definition files are passed to the compile process as **-D** flags to the C compiler.

makeoptions *name=value*
Defines a make(1) macro *name* with the value *value* in the kernel Makefile.

maxusers *integer*
Set the maxusers variable in the kernel.

no *keyword name* [*arguments* [...]]
For the config(1) *keywords* file-system, makeoptions, options, and pseudo-device, *no* removes the file-system, makeoption, options, or pseudo-device, *name*. This is useful when a kernel configuration file includes another which has undesired options.

For example, a local configuration file that wanted the kitchen sink, but not COMPAT_09 or bridging, might be:

```
include "arch/i386/conf/GENERIC"
no options COMPAT_09
no pseudo-device bridge
```

pseudo-device *name* [*N*]
Includes support for the pseudo-device *name*. Some pseudo-devices can have multiple or *N* instances.

Compatibility Options

options COMPAT_09
Enable binary compatibility with NetBSD 0.9. This enables support for 16-bit user, group, and process IDs (following revisions support 32-bit identifiers). It also allows the use of the deprecated getdomainname(3), setdomainname(3), and uname(3) syscalls. This option also allows using numeric file system identifiers rather than strings. Post NetBSD 0.9 versions use string identifiers.

options COMPAT_10
Enable binary compatibility with NetBSD 1.0. This option allows the use of the file system name of "ufs" as an alias for "ffs". The name "ffs" should be used post 1.0 in /etc/fstab and other files. It also adds old syscalls for the AT&T System V UNIX shared memory interface. This was changed post 1.0 to work on 64-bit architectures. This option also enables "sgtty" compatibility, without which programs using the old interface produce an "inappropriate ioctl" error, and /dev/io only works when this option is set in the kernel, see io(4) on ports that support it.

options COMPAT_11
Enable binary compatibility with NetBSD 1.1. This allows binaries running on the i386 port to gain direct access to the io ports by opening /dev/io read/write. This functionality was replaced by i386_iopl(2) post 1.1. On the Atari port, the location of the disk label was moved after 1.1. When the *COMPAT_11* option is set, the kernel will read (pre) 1.1 style disk labels as a last resort. When a disk label is re-written, the old style label will be replaced with a post 1.1 style label. This also enables the *EXEC_ELF_NOTELESS* option.

options COMPAT_12
Enable binary compatibility with NetBSD 1.2. This allows the use of old syscalls for **reboot**() and **swapon**(). The syscall numbers were changed post 1.2 to add functionality to the reboot(2) syscall, and the new swapctl(2) interface was introduced. This also enables the *EXEC_ELF_NOTELESS* option.

options COMPAT_13
Enable binary compatibility with NetBSD 1.3. This allows the use of old syscalls for **sigaltstack**(), and also enables the old swapctl(2) command SWAP_STATS (now called SWAP_OSTATS), which does not include the *se_path* member of *struct swapent*.

options COMPAT_14
Enable binary compatibility with NetBSD 1.4. This allows some old `ioctl`(2) on `wscons`(4) to be performed, and allows the `NFSSVC_BIOD` mode of the `nfssvc`(2) system call to be used for compatibility with the deprecated nfsiod program.

options COMPAT_15
Enable binary compatibility with NetBSD 1.5. Since there were no API changes from NetBSD 1.5 and NetBSD 1.6, this option does nothing.

options COMPAT_16
Enable binary compatibility with NetBSD 1.6. This allows the use of old signal trampoline code which has been deprecated with the addition of `siginfo`(2).

options COMPAT_20
Enable binary compatibility with NetBSD 2.0. This allows the use of old syscalls for **statfs**(), **fstatfs**(), **getfsstat**() and **fhstatfs**(), which have been deprecated with the addition of the `statvfs`(2), `fstatvfs`(2), `getvfsstat`(2) and `fhstatvfs`(2) system calls.

options COMPAT_30
Enable binary compatibility with NetBSD 3.0. See `compat_30`(8) for details about the changes made after the NetBSD 3.0 release.

options COMPAT_43
Enables compatibility with 4.3BSD. This adds an old syscall for `lseek`(2). It also adds the ioctls for `TIOCGETP` and `TIOCSETP`. The return values for `getpid`(2), `getgid`(2), and `getuid`(2) syscalls are modified as well, to return the parent's PID and UID as well as the current process's. It also enables the deprecated `NTTYDISC` terminal line discipline. It also provides backwards compatibility with "old" SIOC[GS]IF{ADDR,DSTADDR,BRDADDR,NETMASK} interface ioctls, including binary compatibility with code written before the introduction of the sa_len field in sockaddrs. It also enables support for some older pre 4.4BSD socket calls.

options COMPAT_BSDPTY
This option is currently on by default and enables the pty multiplexer `ptm`(4) and `ptmx`(4) to find and use ptys named `/dev/ptyXX` (master) and `/dev/ttyXX` (slave). Eventually this option will become optional as ptyfs based pseudo-ttys become the default, see `mount_ptyfs`(8).

options COMPAT_SVR4
On those architectures that support it, this enables binary compatibility with AT&T System V.4 UNIX applications built for the same architecture. This currently includes the i386, m68k, and sparc ports.

options COMPAT_LINUX
On those architectures that support it, this enables binary compatibility with Linux ELF and `a.out`(5) applications built for the same architecture. This currently includes the alpha, arm, i386, m68k, mips, powerpc and x86_64 ports.

options COMPAT_LINUX32
On those 64 bit architectures that support it, this enables binary compatibility with 32 bit Linux binaries. For now this is limited to running i386 ELF Linux binaries on amd64.

options COMPAT_SUNOS
On those architectures that support it, this enables binary compatibility with SunOS 4.1 applications built for the same architecture. This currently includes the sparc, sparc64 and most or all m68k ports. Note that the sparc64 requires the *COMPAT_NETBSD32* option for 64-bit kernels, in addition to this option.

options COMPAT_ULTRIX
On those architectures that support it, this enables binary compatibility with ULTRIX applications built for the same architecture. This currently is limited to the pmax. The functionality of this option is unknown.

options COMPAT_DARWIN
On those architectures that support it, this enables binary compatibility with Darwin applications built for the same architecture. This feature is highly experimental, it requires COMPAT_MACH and EXEC_MACHO and it is currently limited to i386 and powerpc ports of NetBSD.

options COMPAT_FREEBSD
On those architectures that support it, this enables binary compatibility with FreeBSD applications built for the same architecture. At the moment this is limited to the i386 port.

options COMPAT_IBCS2
On those architectures that support it, this enables binary compatibility with iBCS2 or SVR3 applications built for the same architecture. This is currently limited to the i386 and vax ports.

options COMPAT_IRIX
On those architectures that support it, this enables binary compatibility with IRIX o32 binaries built for the same architecture. This feature is experimental, and it is currently limited to the sgimips port.

options COMPAT_MACH
On those architectures that support it, this enables the emulation of Mach kernel traps for binaries built for the same architecture. This feature is highly experimental and it is currently limited to the i386 and powerpc ports of NetBSD.

options COMPAT_OSF1
On those architectures that support it, this enables binary compatibility with Digital UNIX (formerly OSF/1) applications built for the same architecture. This is currently limited to the alpha port.

options COMPAT_NOMID
Enable compatibility with `a.out`(5) executables that lack a machine ID. This includes NetBSD 0.8's ZMAGIC format, and 386BSD and BSDI's QMAGIC, NMAGIC, and OMAGIC `a.out`(5) formats.

options COMPAT_NETBSD32
On those architectures that support it, this enables binary compatibility with 32-bit applications built for the same architecture. This is currently limited to the amd64 and sparc64 ports, and only applicable for 64-bit kernels.

options COMPAT_SVR4_32
On those architectures that support it, this enables binary compatibility with 32-bit SVR4 applications built for the same architecture. This is currently limited to the sparc64 port, and only applicable for 64-bit kernels.

options COMPAT_AOUT_M68K
On m68k architectures which have switched to ELF, this enables binary compatibility with NetBSD/m68k `a.out`(5) executables on NetBSD/m68k ELF kernels. This handles alignment incompatibility of m68k ABI between a.out and ELF which causes the structure padding differences. Currently only some system calls which use *struct stat* are adjusted and some binaries which use `sysctl`(3) to retrieve network details would not work properly.

options EXEC_MACHO
On those architectures that support it, this adds support for running Mach-O executables. This is currently limited to the i386 and powerpc ports of NetBSD.

options EXEC_ELF_NOTELESS
Run unidentified ELF binaries as NetBSD binaries. This might be needed for very old NetBSD ELF binaries on some archs. These old binaries didn't contain an appropriate `.note.netbsd.ident` section, and thus can't be identified by the kernel as NetBSD binaries otherwise. Beware - if this option is on, the kernel would run *any* unknown ELF binaries as if they were NetBSD binaries.

options P1003_1B_SEMAPHORE
Includes kernel support for the standard C library (libc) functions that implement semaphores as specified in ISO/IEC 9945-1:1996 ("POSIX.1").

Debugging Options

options DDB
Compiles in a kernel debugger for diagnosing kernel problems. See ddb(4) for details. *NOTE*: not available on all architectures.

options DDB_FROMCONSOLE=integer
If set to non-zero, DDB may be entered by sending a break on a serial console or by a special key sequence on a graphics console. A value of "0" ignores console breaks or key sequences. If not explicitly specified, the default value is "1". Note that this sets the value of the *ddb.fromconsole* sysctl(3) variable which may be changed at run time -- see sysctl(8) for details.

options DDB_HISTORY_SIZE=integer
If this is non-zero, enable history editing in the kernel debugger and set the size of the history to this value.

options DDB_ONPANIC
The default if not specified is "1" - just enter into DDB. If set to "2" the kernel will attempt to print out a stack trace before entering into DDB. If set to "0" the kernel will attempt to print out a stack trace and reboot the system. If set to "-1" then neither a stack trace is printed or DDB entered - it is as if DDB were not compiled into the kernel. Note that this sets the value of the *ddb.onpanic* sysctl(3) variable which may be changed at run time -- see sysctl(8) for details.

options DDB_COMMANDONENTER=string
This option specify commands which will be executed on each entry to DDB. This sets the default value of the *ddb.commandonenter* sysctl(3) variable which may be changed at run time.

options DDB_BREAK_CHAR=integer
This option overrides using break to enter the kernel debugger on the serial console. The value given is the ASCII value to be used instead. This is currently only supported by the com driver.

options DDB_VERBOSE_HELP
This option adds more verbose descriptions to the *help* command.

options KGDB
Compiles in a remote kernel debugger stub for diagnosing kernel problems using the "remote target" feature of gdb. See gdb(1) for details. *NOTE*: not available on all architectures.

options KGDB_DEV
Device number (as a dev_t) of kgdb device.

options KGDB_DEVADDR
Memory address of kgdb device.

options KGDB_DEVMODE
Permissions of kgdb device.

options KGDB_DEVNAME
Device name of kgdb device.

options KGDB_DEVRATE
Baud rate of kgdb device.

makeoptions DEBUG="-g"
The **-g** flag causes netbsd.gdb to be built in addition to netbsd. netbsd.gdb is useful for debugging kernel crash dumps with gdb. See gdb(1) for details. This also turns on *options DEBUG* (which see).

options DEBUG
Turns on miscellaneous kernel debugging. Since options are turned into preprocessor defines (see above), *options DEBUG* is equivalent to doing a *#define DEBUG* throughout the kernel. Much of the kernel has *#ifdef DEBUG* conditionalized debugging code. Note that many parts of the kernel (typically device drivers) include their own *#ifdef XXX_DEBUG* conditionals instead. This option also turns on certain other options, which may decrease system performance.

options DIAGNOSTIC
Adds code to the kernel that does internal consistency checks. This code will cause the kernel to panic if corruption of internal data structures is detected. These checks can decrease performance up to 15%.

options KSTACK_CHECK_MAGIC
Check kernel stack usage and panic if stack overflow is detected. This check is performance sensitive because it scans stack on each context switch.

options KTRACE
Add hooks for the system call tracing facility, which allows users to watch the system call invocation behavior of processes. See `ktrace`(1) for details.

options MSGBUFSIZE=integer
This option sets the size of the kernel message buffer. This buffer holds the kernel output of **printf**() when not (yet) read by `syslogd`(8). This is particularly useful when the system has crashed and you wish to lookup the kernel output from just before the crash. Also, since the autoconfig output becomes more and more verbose, it sometimes happens that the message buffer overflows before `syslogd`(8) was able to read it. Note that not all systems are capable of obtaining a variable sized message buffer. There are also some systems on which memory contents are not preserved across reboots.

options MALLOCLOG
Enables an event log for `malloc`(9). Useful for tracking down “Data modified on freelist” and “multiple free” problems.

options MALLOCLOGSIZE=integer
Defines the number of entries in the malloc log. Default is 100000 entries.

options UVMHIST
Enables the UVM history logs, which create in-memory traces of various UVM activities. These logs can be displayed be calling **uvmhist_dump**() or **uvm_hist**() with appropriate arguments from DDB. See the kernel source file sys/uvm/uvm_stat.c for details.

options UVMHIST_PRINT
Prints the UVM history logs on the system console as entries are added. Note that the output is *extremely* voluminous, so this option is really only useful for debugging the very earliest parts of kernel initialization.

File Systems

file-system FFS
Includes code implementing the Berkeley Fast File System (*FFS*). Most machines need this if they are not running diskless.

file-system EXT2FS
Includes code implementing the Second Extended File System (*EXT2FS*), revision 0 and revision 1 with the *filetype* and *sparse_super* options. This is the most commonly used file system on the Linux operating system, and is provided here for compatibility. Some of the specific features of *EXT2FS* like the "behavior on errors" are not implemented. This file system can’t be used with UID or GID greater than 65535. See `mount_ext2fs`(8) for details.

file-system LFS
[*EXPERIMENTAL*] Include the Log-structured File System (*LFS*). See mount_lfs(8) and newfs_lfs(8) for details.

file-system MFS
Include the Memory File System (*MFS*). This file system stores files in swappable memory, and produces notable performance improvements when it is used as the file store for /tmp and similar file systems. See mount_mfs(8) for details.

file-system NFS
Include the client side of the Network File System (NFS) remote file sharing protocol. Although the bulk of the code implementing NFS is kernel based, several user level daemons are needed for it to work. See mount_nfs(8) for details.

file-system CD9660
Includes code for the ISO 9660 + Rock Ridge file system, which is the standard file system on many CD-ROM discs. Useful primarily if you have a CD-ROM drive. See mount_cd9660(8) for details.

file-system MSDOSFS
Includes the MS-DOS FAT file system, which is reportedly still used by unfortunate people who have not heard about NetBSD. Also implements the Windows 95 extensions to the same, which permit the use of longer, mixed case file names. See mount_msdos(8) and fsck_msdos(8) for details.

file-system NTFS
[*EXPERIMENTAL*] Includes code for the Microsoft Windows NT file system. See mount_ntfs(8) for details.

file-system FDESC
Includes code for a file system, conventionally mounted on /dev/fd, which permits access to the per-process file descriptor space via special files in the file system. See mount_fdesc(8) for details. Note that this facility is redundant, and thus unneeded on most NetBSD systems, since the fd(4) pseudo-device driver already provides identical functionality. On most NetBSD systems, instances of fd(4) are mknoded under /dev/fd/ and on /dev/stdin, /dev/stdout, and /dev/stderr.

file-system KERNFS
Includes code which permits the mounting of a special file system (normally mounted on /kern) in which files representing various kernel variables and parameters may be found. See mount_kernfs(8) for details.

file-system NULLFS
Includes code for a loopback file system. This permits portions of the file hierarchy to be re-mounted in other places. The code really exists to provide an example of a stackable file system layer. See mount_null(8) for details.

file-system OVERLAY
Includes code for a file system filter. This permits the overlay file system to intercept all access to an underlying file system. This file system is intended to serve as an example of a stacking file system which has a need to interpose itself between an underlying file system and all other access. See mount_overlay(8) for details.

file-system PORTAL
[*EXPERIMENTAL*] Includes the portal file system. This permits interesting tricks like opening TCP sockets by opening files in the file system. The portal file system is conventionally mounted on /p and is partially implemented by a special daemon. See mount_portal(8) for details.

file-system PROCFS
Includes code for a special file system (conventionally mounted on `/proc`) in which the process space becomes visible in the file system. Among other things, the memory spaces of processes running on the system are visible as files, and signals may be sent to processes by writing to `ctl` files in the procfs namespace. See `mount_procfs`(8) for details.

file-system UDF
[*EXPERIMENTAL*] Includes code for the UDF file system commonly found on CD and DVD media but also more and more on USB sticks. Useful primarily if you have a CD or a DVD drive, be it a read-only or a rewritable device. Currently only supports read-access. See `mount_udf`(8) for details.

file-system UMAPFS
Includes a loopback file system in which user and group IDs may be remapped -- this can be useful when mounting alien file systems with different UIDs and GIDs than the local system. See `mount_umap`(8) for details.

file-system UNION
[*EXPERIMENTAL*] Includes code for the union file system, which permits directories to be mounted on top of each other in such a way that both file systems remain visible -- this permits tricks like allowing writing (and the deleting of files) on a read-only file system like a CD-ROM by mounting a local writable file system on top of the read-only file system. See `mount_union`(8) for details.

file-system CODA
[*EXPERIMENTAL*] Includes code for the Coda file system. Coda is a distributed file system like NFS and AFS. It is freely available, like NFS, but it functions much like AFS in being a "stateful" file system. Both Coda and AFS cache files on your local machine to improve performance. Then Coda goes a step further than AFS by letting you access the cached files when there is no available network, viz. disconnected laptops and network outages. In Coda, both the client and server are outside the kernel which makes them easier to experiment with. Coda is available for several UNIX and non-UNIX platforms. See http://www.coda.cs.cmu.edu for more details. *NOTE*: You also need to enable the pseudo-device, vcoda, for the Coda file system to work.

file-system SMBFS
[*EXPERIMENTAL*] Includes code for the SMB/CIFS file system. See `mount_smbfs`(8) for details. *NOTE*: You also need to enable the pseudo-device, nsmb, for the SMB file system to work.

file-system PTYFS
[*EXPERIMENTAL*] Includes code for a special file system (normally mounted on `/dev/pts`) in which pseudo-terminal slave devices become visible in the file system. See `mount_ptyfs`(8) for details.

file-system TMPFS
Includes code for the efficient memory file system, normally used over `/tmp`. See `mount_tmpfs`(8) for details.

file-system PUFFS
[*EXPERIMENTAL*] Includes kernel support for the pass-to-userspace framework file system. It can be used to implement file system functionality in userspace. See `puffs`(3) and `puffs`(4) for more details. This option is required for sshfs: `mount_psshfs`(8).

File System Options

options MAGICLINKS
Enables the expansion of special strings (beginning with "@") when traversing symbolic links. See `symlink`(7) for a list of supported strings. Note that this option only controls the enabling of this feature by the kernel at boot-up. This feature can still be manipulated with the `sysctl`(8) command regardless of the setting of this option.

options NFSSERVER
Include the server side of the *NFS* (Network File System) remote file sharing protocol. Although the bulk of the code implementing *NFS* is kernel based, several user level daemons are needed for it to work. See `mountd`(8) and `nfsd`(8) for details.

options QUOTA
Enables kernel support for file system quotas. See `quotaon`(8), `edquota`(8), and `quota`(1) for details. Note that quotas only work on "ffs" file systems, although `rpc.rquotad`(8) permits them to be accessed over *NFS*.

options FFS_EI
Enable ''Endian-Independent'' FFS support. This allows a system to mount an FFS file system created for another architecture, at a small performance cost for all FFS file systems. See also `newfs`(8), `fsck_ffs`(8), `dumpfs`(8) for file system byte order status and manipulation.

options FFS_NO_SNAPSHOT
Disable the "file system snapshot" support in FFS file systems. Maybe useful for install media kernels, small memory systems and embedded systems which don't require the snapshot support.

options NVNODE=integer
This option sets the size of the cache used by the name-to-inode translation routines, (a.k.a. the **namei**() cache, though called by many other names in the kernel source). By default, this cache has `NPROC` (set as 20 + 16 ∗ MAXUSERS) ∗ (80 + NPROC / 8) entries. A reasonable way to derive a value of `NVNODE`, should you notice a large number of namei cache misses with a tool such as `systat`(1), is to examine your system's current computed value with `sysctl`(8), (which calls this parameter "kern.maxvnodes") and to increase this value until either the namei cache hit rate improves or it is determined that your system does not benefit substantially from an increase in the size of the namei cache.

options NAMECACHE_ENTER_REVERSE
Causes the namei cache to always enter a reverse mapping (vnode -> name) as well as a normal one. Normally, this is already done for directory vnodes, to speed up the getcwd operation. This option will cause longer hash chains in the reverse cache, and thus slow down getcwd somewhat. However, it does make vnode -> path translations possible in some cases. For now, only useful if strict /proc/#/maps emulation for Linux binaries is required.

options EXT2FS_SYSTEM_FLAGS
This option changes the behavior of the APPEND and IMMUTABLE flags for a file on an *EXT2FS* file system. Without this option, the superuser or owner of the file can set and clear them. With this option, only the superuser can set them, and they can't be cleared if the securelevel is greater than 0. See also `chflags`(1).

options NFS_BOOT_BOOTP
Enable use of the BOOTP protocol (RFCs 951 and 1048) to get configuration information if NFS is used to mount the root file system. See `diskless`(8) for details.

options NFS_BOOT_DHCP
Same as "NFS_BOOT_BOOTP", but use the DHCP extensions to the BOOTP protocol (RFC 1541).

options NFS_BOOT_BOOTP_REQFILE
Specifies the string sent in the bp_file field of the BOOTP / DHCP request packet.

options NFS_BOOT_BOOTPARAM
Enable use of the BOOTPARAM protocol, consisting of RARP and BOOTPARAM RPC, to get configuration information if NFS is used to mount the root file system. See `diskless`(8) for details.

options NFS_BOOT_RWSIZE=value
Set the initial NFS read and write sizes for diskless-boot requests. The normal default is 8Kbytes. This option provides a way to lower the value (e.g., to 1024 bytes) as a workaround for buggy network interface

cards or boot PROMs. Once booted, the read and write request sizes can be increased by remounting the file system. See mount_nfs(8) for details.

options NFS_V2_ONLY
Reduce the size of the NFS client code by omitting code that's only required for NFSv3 and NQNFS support, leaving only that code required to use NFSv2 servers.

options SOFTDEP
Enable kernel support for soft-dependencies in FFS file systems. Softdep improves write performance by aggregating and properly ordering disk metadata writes, achieving near-asynchronous write performance while maintaining the file system consistency of synchronous writes. Soft-dependencies are enabled on a per-mount basis. See mount(8) for details.

options UFS_DIRHASH
Increase lookup performance by maintaining in-core hash tables for large directories.

Buffer queue strategy options
The following options enable alternative buffer queue strategies.

options BUFQ_READPRIO
Enable experimental buffer queue strategy for disk I/O. In the default strategy, outstanding disk requests are ordered by sector number and sent to the disk, regardless of whether the operation is a read or write; this option gives priority to issuing read requests over write requests. Although requests may therefore be issued out of sector-order, causing more seeks and thus lower overall throughput, interactive system responsiveness under heavy disk I/O load may be improved, as processes blocking on disk reads are serviced sooner (file writes typically don't cause applications to block). The performance effect varies greatly depending on the hardware, drive firmware, file system configuration, workload, and desired performance trade-off. Systems using drive write-cache (most modern IDE disks, by default) are unlikely to benefit and may well suffer; such disks acknowledge writes very quickly, and optimize them internally according to physical layout. Giving these disks as many requests to work with as possible (the standard strategy) will typically produce the best results, especially if the drive has a large cache; the drive will silently complete writes from cache as it seeks for reads. Disks that support a large number of concurrent tagged requests (SCSI disks and many hardware RAID controllers) expose this internal scheduling with tagged responses, and don't block for reads; such disks may not see a noticeable difference with either strategy. However, if IDE disks are run with write-cache disabled for safety, writes are not acknowledged until actually completed, and only one request can be outstanding; a large number of small writes in one locality can keep the disk busy, starving reads elsewhere on the disk. Such systems are likely to see the most benefit from this option. Finally, the performance interaction of this option with ffs soft dependencies can be subtle, as that mechanism can drastically alter the workload for file system metadata writes.

options BUFQ_PRIOCSCAN
Enable another buffer queue strategy for disk I/O, per-priority cyclical scan.

options NEW_BUFQ_STRATEGY
Synonym of *BUFQ_READPRIO*.

Miscellaneous Options
options LKM
Enable loadable kernel modules. See lkm(4) for details. *NOTE*: not available on all architectures.

options MEMORY_DISK_DYNAMIC
This option makes the md(4) RAM disk size dynamically sized. It is incompatible with mdsetimage(8).

options MEMORY_DISK_HOOKS
This option allows for some machine dependent functions to be called when the md(4) RAM disk driver is configured. This can result in automatically loading a RAM disk from floppy on open (among other things).

options MEMORY_DISK_IS_ROOT
Forces the md(4) RAM disk to be the root device. This can only be overridden when the kernel is booted in the 'ask-for-root' mode.

options MEMORY_DISK_ROOT_SIZE=integer
Allocates the given number of 512 byte blocks as memory for the md(4) RAM disk, to be populated with mdsetimage(8).

options MEMORY_DISK_SERVER=0
Do not include the interface to a userland memory disk server process. Per default, this option is set to 1, including the support code. Useful for install media kernels.

options MODULAR
Enables the new framework for kernel modules, which will eventually replace LKMs. This adds an in-kernel linker and loader, and requires userland support to be useful. See the **MKMODULAR** variable in mk.conf(5) for more details.

options VND_COMPRESSION
Enables the vnd(4) driver to also handle compressed images. See vndcompress(1), vnd(4) and vnconfig(8) for more information.

options TFTPROOT
Download the root memory disk through TFTP at root mount time. This enables the use of a root RAM disk without requiring it to be embedded in the kernel using mdsetimage(8). The RAM disk name is obtained using DHCP's filename parameter. This option requires *MEMORY_DISK_HOOKS*, *MEMORY_DISK_DYNAMIC*, and *MEMORY_DISK_IS_ROOT*. It is incompatible with *MEMORY_DISK_ROOT_SIZE*.

options MALLOC_NOINLINE
Time critical fixed size memory allocation is performed with **MALLOC**() and **FREE**(). Normally these expand to inline code, but with *MALLOC_NOINLINE* these call the normal **malloc**() and **free**() functions. Useful for install media kernels, small memory systems and embedded systems.

options HZ=integer
On ports that support it, set the system clock frequency (see hz(9)) to the supplied value. Handle with care.

options NTP
Turns on in-kernel precision timekeeping support used by software implementing *NTP* (Network Time Protocol, RFC 1305). The *NTP* option adds an in-kernel Phase-Locked Loop (PLL) for normal *NTP* operation, and a Frequency-Locked Loop (FLL) for intermittently-connected operation. ntpd(8) will employ a user-level PLL when kernel support is unavailable, but the in-kernel version has lower latency and more precision, and so typically keeps much better time. The interface to the kernel *NTP* support is provided by the ntp_adjtime(2) and ntp_gettime(2) system calls, which are intended for use by ntpd(8) and are enabled by the option. On systems with sub-microsecond resolution timers, or where (HZ / 100000) is not an integer, the *NTP* option also enables extended-precision arithmetic to keep track of fractional clock ticks at NTP time-format precision.

options PPS_SYNC
This option enables a kernel serial line discipline for receiving time phase signals from an external reference clock such as a radio clock. (The *NTP* option (which see) must be on if the *PPS_SYNC* option is used.) Some reference clocks generate a Pulse Per Second (PPS) signal in phase with their time source. The *PPS* line discipline receives this signal on either the data leads or the DCD control lead of a serial port. *NTP* uses the PPS signal to discipline the local clock oscillator to a high degree of precision (typically less than 50 microseconds in time and 0.1 ppm in accuracy). *PPS* can also generate a serial output pulse when the system receives a PPS interrupt. This can be used to measure the system interrupt latency and thus calibrate *NTP* to account for it. Using *PPS* usually requires a gadget box to convert from TTL to RS-232 signal levels. The

gadget box and PPS are described in more detail in the HTML documentation for `ntpd`(8) in `/usr/share/doc/html/ntp`.

`options NO_TSC_TIME`
Don't use TSC microtime, even if available (i386 only). Improves time behavior under VMware.

`options SETUIDSCRIPTS`
Allows scripts with the setuid bit set to execute as the effective user rather than the real user, just like binary executables.

NOTE: Using this option will also enable *options FDSCRIPTS*

`options FDSCRIPTS`
Allows execution of scripts with the execute bit set, but not the read bit, by opening the file and passing the file descriptor to the shell, rather than the filename.

NOTE: Execute only (non-readable) scripts will have *argv[0]* set to `/dev/fd/`∗. What this option allows as far as security is concerned, is the ability to safely ensure that the correct script is run by the interpreter, as it is passed as an already open file.

`options PUCCN`
Enables treating serial ports found on PCI boards `puc`(4) as potential console devices. The method for choosing such a console device is port dependent.

`options RTC_OFFSET=integer`
The kernel (and typically the hardware battery backed-up clock on those machines that have one) keeps time in *UTC* (Universal Coordinated Time, once known as *GMT*, or Greenwich Mean Time) and not in the time of the local time zone. The *RTC_OFFSET* option is used on some ports (such as the i386) to tell the kernel that the hardware clock is offset from *UTC* by the specified number of minutes. This is typically used when a machine boots several operating systems and one of them wants the hardware clock to run in the local time zone and not in *UTC*, e.g. *RTC_OFFSET=300* means the hardware clock is set to US Eastern Time (300 minutes behind *UTC*), and not *UTC*. (Note: *RTC_OFFSET* is used to initialize a kernel variable named *rtc_offset* which is the source actually used to determine the clock offset, and which may be accessed via the kern.rtc_offset sysctl variable. See `sysctl`(8) and `sysctl`(3) for details. Since the kernel clock is initialized from the hardware clock very early in the boot process, it is not possible to meaningfully change *rtc_offset* in system initialization scripts. Changing this value currently may only be done at kernel compile time or by patching the kernel and rebooting).

NOTE: Unfortunately, in many cases where the hardware clock is kept in local time, it is adjusted for Daylight Savings Time; this means that attempting to use *RTC_OFFSET* to let NetBSD coexist with such an operating system, like Windows, would necessitate changing *RTC_OFFSET* twice a year. As such, this solution is imperfect.

`options KMEMSTATS`
The kernel memory allocator, `malloc`(9), will keep statistics on its performance if this option is enabled. Unfortunately, this option therefore essentially disables the **`MALLOC`**() and **`FREE`**() forms of the memory allocator, which are used to enhance the performance of certain critical sections of code in the kernel. This option therefore can lead to a significant decrease in the performance of certain code in the kernel if enabled. Examples of such code include the **`namei`**() routine, the `ccd`(4) driver, and much of the networking code.

`options MAXUPRC=integer`
Sets the soft `RLIMIT_NPROC` resource limit, which specifies the maximum number of simultaneous processes a user is permitted to run, for process 0; this value is inherited by its child processes. It defaults to *CHILD_MAX*, which is currently defined to be 160. Setting *MAXUPRC* to a value less than *CHILD_MAX* is not permitted, as this would result in a violation of the semantics of ISO/IEC 9945-1:1990 ("POSIX.1").

options NOFILE=integer
Sets the soft RLIMIT_NOFILE resource limit, which specifies the maximum number of open file descriptors for each process; this value is inherited by its child processes. It defaults to *OPEN_MAX*, which is currently defined to be 64.

options MAXFILES=integer
Sets the default value of the *kern.maxfiles* sysctl variable, which indicates the maximum number of files that may be open in the system.

options DEFCORENAME=string
Sets the default value of the *kern.defcorename* sysctl variable, otherwise it is set to **%n.core**. See sysctl(8) and sysctl(3) for details.

options RASOPS_CLIPPING
Enables clipping within the **rasops** raster-console output system. *NOTE*: only available on architectures that use **rasops** for console output.

options RASOPS_SMALL
Removes optimized character writing code from the **rasops** raster-console output system. *NOTE*: only available on architectures that use **rasops** for console output.

options INCLUDE_CONFIG_FILE
Embeds the kernel config file used to define the kernel in the kernel binary itself. The embedded data also includes any files directly included by the config file itself, e.g. GENERIC.local or std.$MACHINE. The embedded config file can be extracted from the resulting kernel with config(1) **-x**, or by the following command:

```
strings netbsd | sed -n 's/^_CFG_//p' | unvis
```

options INCLUDE_JUST_CONFIG
Similar to the above option, but includes just the actual config file, not any included files.

options PIPE_SOCKETPAIR
Use slower, but smaller socketpair(2)-based pipe implementation instead of default faster, but bigger one. Primarily useful for installation kernels.

options USERCONF
Compiles in the in-kernel device configuration manager. See userconf(4) for details.

options PERFCTRS
Compiles in kernel support for CPU performance-monitoring counters. See pmc(1) for details. *NOTE*: not available on all architectures.

options SYSCALL_STATS
Count the number of times each system call number is called. The values can be read through the sysctl interface and displayed using systat(1). *NOTE*: not yet available on all architectures.

options SYSCALL_TIMES
Count the time spent (using **cpu_counter32**()) in each system call. *NOTE*: Using this option will also enable **options SYSCALL_STATS**.

options SYSCALL_TIMES_HASCOUNTER
Force use of **cpu_counter32**() even if **cpu_hascounter**() reports false. Useful for systems where the cycle counter doesn't run at a constant rate (e.g. Soekris boxes).

options XSERVER
Compiles in kernel support for X11 on architectures that still use (or can use) the legacy *pccons* console drivers rather than wscons(4). These include bebox, i386, shark.

options XSERVER_DDB
A supplement to XSERVER that adds support for entering ddb(4) while in X11.

options FILEASSOC
Support for fileassoc(9).

options FILEASSOC_NHOOKS=integer
Number of storage slots per file for fileassoc(9). Default is 4.

Networking Options

options GATEWAY
Enables *IPFORWARDING* (which see) and (on most ports) increases the size of *NMBCLUSTERS* (which see). In general, *GATEWAY* is used to indicate that a system should act as a router, and *IPFORWARDING* is not invoked directly. (Note that *GATEWAY* has no impact on protocols other than IP, such as CLNP or XNS). *GATEWAY* option also compiles IPv4 and IPv6 fast forwarding code into the kernel.

options ICMPPRINTFS
The *ICMPPRINTFS* option will enable debugging information to be printed about the icmp(4) protocol.

options IPFORWARDING=value
If *value* is 1 this enables IP routing behavior. If *value* is 0 (the default), it disables it. The *GATEWAY* option sets this to 1 automatically. With this option enabled, the machine will forward IP datagrams destined for other machines between its interfaces. Note that even without this option, the kernel will still forward some packets (such as source routed packets) -- removing *GATEWAY* and *IPFORWARDING* is insufficient to stop all routing through a bastion host on a firewall -- source routing is controlled independently. To turn off source routing, use *options IPFORWSRCRT=0* (which see). Note that IP forwarding may be turned on and off independently of the setting of the *IPFORWARDING* option through the use of the *net.inet.ip.forwarding* sysctl variable. If *net.inet.ip.forwarding* is 1, IP forwarding is on. See sysctl(8) and sysctl(3) for details.

options IPFORWSRCRT=value
If *value* is set to zero, source routing of IP datagrams is turned off. If *value* is set to one (the default) or the option is absent, source routed IP datagrams are forwarded by the machine. Note that source routing of IP packets may be turned on and off independently of the setting of the *IPFORWSRCRT* option through the use of the *net.inet.ip.forwsrcrt* sysctl variable. If *net.inet.ip.forwsrcrt* is 1, forwarding of source routed IP datagrams is on. See sysctl(8) and sysctl(3) for details.

options IFA_STATS
Tells the kernel to maintain per-address statistics on bytes sent and received over (currently) Internet and AppleTalk addresses. The option is not recommended as it degrades system stability.

options IFQ_MAXLEN=value
Increases the allowed size of the network interface packet queues. The default queue size is 50 packets, and you do not normally need to increase it.

options IPSELSRC
Includes support for source-address selection policies. See in_getifa(9).

options MROUTING
Includes support for IP multicast routers. You certainly want *INET* with this. Multicast routing is controlled by the mrouted(8) daemon. See also option **PIM**.

options PIM
Includes support for Protocol Independent Multicast (PIM) routing. You need *MROUTING* and *INET* with this. Software using this can be found e.g. in pkgsrc/net/xorp.

options INET
Includes support for the TCP/IP protocol stack. You almost certainly want this. See `inet`(4) for details.

options INET6
Includes support for the IPv6 protocol stack. See `inet6`(4) for details. Unlike *INET*, *INET6* enables multicast routing code as well. This option requires *INET* at this moment, but it should not.

options ND6_DEBUG
The option sets the default value of net.inet6.icmp6.nd6_debug to 1, for debugging IPv6 neighbor discovery protocol handling. See `sysctl`(3) for details.

options IPSEC
Includes support for the IPsec protocol. See `ipsec`(4) for details.

options IPSEC_DEBUG
Enables debugging code in IPsec stack. See `ipsec`(4) for details.

options IPSEC_ESP
Includes support for IPsec ESP protocol. See `ipsec`(4) for details.

options IPSEC_NAT_T
Includes support for IPsec Network Address Translator traversal (NAT-T), as described in RFCs 3947 and 3948. This feature might be patent-encumbered in some countries.

options ALTQ
Enabled ALTQ (Alternate Queueing). For simple rate-limiting, use `tbrconfig`(8) to set up the interface transmission rate. To use queueing disciplines, their appropriate kernel options should also be defined (documented below). Queueing disciplines are managed by `altqd`(8). See `altq`(9) for details.

options ALTQ_HFSC
Include support for ALTQ-implemented HFSC (Hierarchical Fair Service Curve) module. HFSC supports both link-sharing and guaranteed real-time services. HFSC employs a service curve based QoS model, and its unique feature is an ability to decouple delay and bandwidth allocation. Requires *ALTQ_RED* to use the RED queueing discipline on HFSC classes, or *ALTQ_RIO* to use the RIO queueing discipline on HFSC classes. This option assumes *ALTQ*.

options ALTQ_PRIQ
Include support for ALTQ-implemented PRIQ (Priority Queueing). PRIQ implements a simple priority-based queueing discipline. A higher priority class is always served first. Requires *ALTQ_RED* to use the RED queueing discipline on HFSC classes, or *ALTQ_RIO* to use the RIO queueing discipline on HFSC classes. This option assumes *ALTQ*.

options ALTQ_WFQ
Include support for ALTQ-implemented WFQ (Weighted Fair Queueing). WFQ implements a weighted-round robin scheduler for a set of queues. A weight can be assigned to each queue to give a different proportion of the link capacity. A hash function is used to map a flow to one of a set of queues. This option assumes *ALTQ*.

options ALTQ_FIFOQ
Include support for ALTQ-implemented FIFO queueing. FIFOQ is a simple drop-tail FIFO (First In, First Out) queueing discipline. This option assumes *ALTQ*.

options ALTQ_RIO
Include support for ALTQ-implemented RIO (RED with In/Out). The original RIO has 2 sets of RED parameters; one for in-profile packets and the other for out-of-profile packets. At the ingress of the network, profile meters tag packets as IN or OUT based on contracted profiles for customers. Inside the network, IN packets receive preferential treatment by the RIO dropper. ALTQ/RIO has 3 drop precedence levels defined for the Assured Forwarding PHB of DiffServ (RFC 2597). This option assumes *ALTQ*.

options ALTQ_BLUE
Include support for ALTQ-implemented Blue buffer management. Blue is another active buffer management mechanism. This option assumes *ALTQ*.

options ALTQ_FLOWVALVE
Include support for ALTQ-implemented Flowvalve. Flowvalve is a simple implementation of a RED penalty box that identifies and punishes misbehaving flows. This option requires *ALTQ_RED* and assumes *ALTQ*.

options ALTQ_CDNR
Include support for ALTQ-implemented CDNR (diffserv traffic conditioner) packet marking/manipulation. Traffic conditioners are components to meter, mark, or drop incoming packets according to some rules. As opposed to queueing disciplines, traffic conditioners handle incoming packets at an input interface. This option assumes *ALTQ*.

options ALTQ_NOPCC
Disables use of processor cycle counter to measure time in ALTQ. This option should be defined for a non-Pentium i386 CPU which does not have TSC, SMP (per-CPU counters are not in sync), or power management which affects processor cycle counter. This option assumes *ALTQ*.

options ALTQ_IPSEC
Include support for IPsec in IPv4 ALTQ. This option assumes *ALTQ*.

options ALTQ_JOBS
Include support for ALTQ-implemented JoBS (Joint Buffer Management and Scheduling). This option assumes *ALTQ*.

options ALTQ_AFMAP
Include support for an undocumented ALTQ feature that is used to map an IP flow to an ATM VC (Virtual Circuit). This option assumes *ALTQ*.

options ALTQ_LOCALQ
Include support for ALTQ-implemented local queues. Its practical use is undefined. Assumes *ALTQ*.

options SUBNETSARELOCAL
Sets default value for net.inet.ip.subnetsarelocal variable, which controls whether non-directly-connected subnets of connected networks are considered "local" for purposes of choosing the MSS for a TCP connection. This is mostly present for historic reasons and completely irrelevant if you enable Path MTU discovery.

options HOSTZEROBROADCAST
Sets default value for net.inet.ip.hostzerobroadcast variable, which controls whether the zeroth host address of each connected subnet is also considered a broadcast address. Default value is "1", for compatibility with old systems; if this is set to zero on all hosts on a subnet, you should be able to fit an extra host per subnet on the ".0" address.

options MCLSHIFT=value
This option is the base-2 logarithm of the size of mbuf clusters. The BSD networking stack keeps network packets in a linked list, or chain, of kernel buffer objects called mbufs. The system provides larger mbuf clusters as an optimization for large packets, instead of using long chains for large packets. The mbuf cluster size, or *MCLBYTES*, must be a power of two, and is computed as two raised to the power *MCLSHIFT*. On systems with Ethernet network adapters, *MCLSHIFT* is often set to 11, giving 2048-byte mbuf clusters, large enough to hold a 1500-byte Ethernet frame in a single cluster. Systems with network interfaces supporting larger frame sizes like ATM, FDDI, or HIPPI may perform better with *MCLSHIFT* set to 12 or 13, giving mbuf cluster sizes of 4096 and 8192 bytes, respectively.

options NS
Include support for the Xerox XNS protocol stack. See ns(4) for details.

options ISO,TPIP
Include support for the ubiquitous OSI protocol stack. See `iso`(4) for details. This option assumes *INET*.

options EON
Include support for tunneling OSI protocols over IP. Known to be broken, or at least very fragile, and undocumented.

options NETATALK
Include support for the AppleTalk protocol stack. The kernel provides provision for the *Datagram Delivery Protocol* (DDP), providing SOCK_DGRAM support and AppleTalk routing. This stack is used by the *NETATALK* package, which adds support for AppleTalk server services via user libraries and applications.

options BLUETOOTH
Include support for the Bluetooth protocol stack. See `bluetooth`(4) for details.

options IPNOPRIVPORTS
Normally, only root can bind a socket descriptor to a so-called "privileged" TCP port, that is, a port number in the range 0-1023. This option eliminates those checks from the kernel. This can be useful if there is a desire to allow daemons without privileges to bind those ports, e.g., on firewalls. The security tradeoffs in doing this are subtle. This option should only be used by experts.

options TCP_COMPAT_42
TCP bug compatibility with 4.2BSD. In 4.2BSD, TCP sequence numbers were 32-bit signed values. Modern implementations of TCP use unsigned values. This option clamps the initial sequence number to start in the range 2^31 rather than the full unsigned range of 2^32. Also, under 4.2BSD, keepalive packets must contain at least one byte or else the remote end would not respond.

options TCP_DEBUG
Record the last *TCP_NDEBUG* TCP packets with SO_DEBUG set, and decode to the console if *tcpconsdebug* is set.

options TCP_NDEBUG
Number of packets to record for *TCP_DEBUG*. Defaults to 100.

options TCP_SENDSPACE=value

options TCP_RECVSPACE=value
These options set the max TCP window size to other sizes than the default. The TCP window sizes can be altered via `sysctl`(8) as well.

options TCP_INIT_WIN=value
This option sets the initial TCP window size for non-local connections, which is used when the transmission starts. The default size is 1, but if the machine should act more aggressively, the initial size can be set to some other value. The initial TCP window size can be set via `sysctl`(8) as well.

options PFIL_HOOKS
This option turns on the packet filter interface hooks. See `pfil`(9) for details. This option assumes *INET*.

options IPFILTER_LOG
This option, in conjunction with *pseudo-device ipfilter*, enables logging of IP packets using ip-filter.

options IPFILTER_DEFAULT_BLOCK
This option sets the default policy of ip-filter. If it is set, ip-filter will block packets by default.

options BRIDGE_IPF
This option causes *bridge* devices to use the IP and/or IPv6 filtering hooks, forming a link-layer filter that uses protocol-layer rules. This option assumes the presence of *pseudo-device ipfilter*.

options MBUFTRACE
This option can help track down mbuf leaks. When enabled, mbufs are tagged with the devices and protocols using them, which slightly decreases network performance. This additional information can be viewed with `netstat`(1):

```
netstat -mssv
```

Not all devices or protocols support this option.

Sysctl Related Options

options SYSCTL_DISALLOW_CREATE
Disallows the creation or deletion of nodes from the sysctl tree, as well as the assigning of descriptions to nodes that lack them, by any process. These operations are still available to kernel sub-systems, including loadable kernel modules.

options SYSCTL_DISALLOW_KWRITE
Prevents processes from adding nodes to the sysctl tree that make existing kernel memory areas writable. Sections of kernel memory can still be read and new nodes that own their own data may still be writable.

options SYSCTL_DEBUG_SETUP
Causes the SYSCTL_SETUP routines to print a brief message when they are invoked. This is merely meant as an aid in determining the order in which sections of the tree are created.

options SYSCTL_DEBUG_CREATE
Prints a message each time **sysctl_create**(), the function that adds nodes to the tree, is called.

options SYSCTL_INCLUDE_DESCR
Causes the kernel to include short, human readable descriptions for nodes in the sysctl tree. The descriptions can be retrieved programmatically (see `sysctl`(3)), or by the sysctl binary itself (see `sysctl`(8)). The descriptions are meant to give an indication of the purpose and/or effects of a given node's value, not replace the documentation for the given subsystem as a whole.

System V IPC Options

options SYSVMSG
Includes support for AT&T System V UNIX style message queues. See `msgctl`(2), `msgget`(2), `msgrcv`(2), `msgsnd`(2).

options SYSVSEM
Includes support for AT&T System V UNIX style semaphores. See `semctl`(2), `semget`(2), `semop`(2).

options SEMMNI=value
Sets the number of AT&T System V UNIX style semaphore identifiers. The GENERIC config file for your port will have the default.

options SEMMNS=value
Sets the number of AT&T System V UNIX style semaphores in the system. The GENERIC config file for your port will have the default.

options SEMUME=value
Sets the maximum number of undo entries per process for AT&T System V UNIX style semaphores. The GENERIC config file for your port will have the default.

options SEMMNU=value
Sets the number of undo structures in the system for AT&T System V UNIX style semaphores. The GENERIC config file for your port will have the default.

options SYSVSHM
Includes support for AT&T System V UNIX style shared memory. See `shmat`(2), `shmctl`(2), `shmdt`(2), `shmget`(2).

options SHMMAXPGS=value
Sets the maximum number of AT&T System V UNIX style shared memory pages that are available through the shmget(2) system call. Default value is 1024 on most ports. See /usr/include/machine/vmparam.h for the default.

VM Related Options

options NMBCLUSTERS=value
The number of mbuf clusters the kernel supports. Mbuf clusters are MCLBYTES in size (usually 2k). This is used to compute the size of the kernel VM map *mb_map*, which maps mbuf clusters. Default on most ports is 1024 (2048 with "options GATEWAY"). See /usr/include/machine/param.h for exact default information. Increase this value if you get "mclpool limit reached" messages.

options NKMEMPAGES=value

options NKMEMPAGES_MIN=value

options NKMEMPAGES_MAX=value
Size of kernel VM map *kmem_map*, in PAGE_SIZE-sized chunks (the VM page size; this value may be read from the sysctl(8) variable *hw.pagesize*). This VM map is used to map the kernel malloc arena. The kernel attempts to auto-size this map based on the amount of physical memory in the system. Platform-specific code may place bounds on this computed size, which may be viewed with the sysctl(8) variable *vm.nkmempages*. See /usr/include/machine/param.h for the default upper and lower bounds. The related options 'NKMEMPAGES_MIN' and 'NKMEMPAGES_MAX' allow the bounds to be overridden in the kernel configuration file. These options are provided in the event the computed value is insufficient resulting in an "out of space in kmem_map" panic.

options SB_MAX=value
Sets the max size in bytes that a socket buffer is allowed to occupy. The default is 256k, but sometimes it needs to be increased, for example when using large TCP windows. This option can be changed via sysctl(8) as well.

options SOMAXKVA=value
Sets the maximum size of kernel virtual memory that the socket buffers are allowed to use. The default is 16MB, but in situations where for example large TCP windows are used this value must also be increased. This option can be changed via sysctl(8) as well.

options BUFCACHE=value
Size of the buffer cache as a percentage of total available RAM. Ignored if BUFPAGES is also specified.

options NBUF=value
Sets the number of buffer headers available, i.e., the number of open files that may have a buffer cache entry. Each buffer header requires MAXBSIZE (machine dependent, but usually 65536) bytes. The default value is machine dependent, but is usually equal to the value of BUFPAGES. If an architecture dependent VM_MAX_KERNEL_BUF constant is defined then NBUF may be reduced at run time so that the storage allocated for buffer headers doesn't exceed that limit.

options BUFPAGES=value
These options set the number of pages available for the buffer cache. Their default value is a machine dependent value, often calculated as between 5% and 10% of total available RAM.

options MAXTSIZ=bytes
Sets the maximum size limit of a process' text segment. See /usr/include/machine/vmparam.h for the port-specific default.

options DFLDSIZ=bytes
Sets the default size limit of a process' data segment, the value that will be returned as the soft limit for RLIMIT_DATA (as returned by getrlimit(2)). See /usr/include/machine/vmparam.h for the

port-specific default.

options MAXDSIZ=bytes
Sets the maximum size limit of a process' data segment, the value that will be returned as the hard limit for RLIMIT_DATA (as returned by getrlimit(2)). See /usr/include/machine/vmparam.h for the port-specific default.

options DFLSSIZ=bytes
Sets the default size limit of a process' stack segment, the value that will be returned as the soft limit for RLIMIT_STACK (as returned by getrlimit(2)). See /usr/include/machine/vmparam.h for the port-specific default.

options MAXSSIZ=bytes
Sets the maximum size limit of a process' stack segment, the value that will be returned as the hard limit for RLIMIT_STACK (as returned by getrlimit(2)). See /usr/include/machine/vmparam.h for the port-specific default.

options DUMP_ON_PANIC=integer
Defaults to one. If set to zero, the kernel will not dump to the dump device when it panics, though dumps can still be forced via ddb(4) with the "sync" command. Note that this sets the value of the *kern.dump_on_panic* sysctl(3) variable which may be changed at run time -- see sysctl(8) for details.

options USE_TOPDOWN_VM
User space memory allocations (as made by mmap(2)) will be arranged in a "top down" fashion instead of the traditional "upwards from MAXDSIZ + vm_daddr" method. This includes the placement of ld.so(1). Arranging memory in this manner allows either (or both of) the heap or mmap(2) allocated space to grow larger than traditionally possible. This option is not available on all ports, but is instead expected to be offered on a port-by-port basis, after which some ports will commit to using it by default. See the files /usr/include/uvm/uvm_param.h for some implementation details, and /usr/include/machine/vmparam.h for port specific details including availability.

options VMSWAP
Enable paging device/file support. This option is on by default.

options PDPOLICY_CLOCKPRO
Use CLOCK-Pro, an alternative page replace policy.

Security Options

options INSECURE
Hardwires the kernel security level at −1. This means that the system always runs in secure level −1 mode, even when running multiuser. See the manual page for init(8) for details on the implications of this. The kernel secure level may manipulated by the superuser by altering the *kern.securelevel* sysctl(3) variable (the secure level may only be lowered by a call from process ID 1, i.e., init(8)). See also sysctl(8) and sysctl(3).

options VERIFIED_EXEC_FP_MD5
Enables support for MD5 hashes in Veriexec.

options VERIFIED_EXEC_FP_SHA1
Enables support for SHA1 hashes in Veriexec.

options VERIFIED_EXEC_FP_RMD160
Enables support for RMD160 hashes in Veriexec.

options VERIFIED_EXEC_FP_SHA256
Enables support for SHA256 hashes in Veriexec.

options VERIFIED_EXEC_FP_SHA384
Enables support for SHA384 hashes in Veriexec.

options VERIFIED_EXEC_FP_SHA512
Enables support for SHA512 hashes in Veriexec.

options PAX_MPROTECT=value
Enables PaX MPROTECT, mprotect(2) restrictions from the PaX project.

The *value* is the default value for the *global* knob, see sysctl(3). If 0, PaX MPROTECT will be enabled only if explicitly set on programs using paxctl(8). If 1, PaX MPROTECT will be enabled for all programs. Programs can be exempted using paxctl(8).

See security(8) for more details.

options PAX_SEGVGUARD=value
Enables PaX Segvguard.

The *value* is the default value for the *global* knob, see sysctl(3). If 0, PaX Segvguard will be enabled only if explicitly set on programs using paxctl(8). If 1, PaX Segvguard will be enabled to all programs, and exemption can be done using paxctl(8).

See security(8) for more details.

options PAX_ASLR=value
Enables PaX ASLR.

The *value* is the default value for the *global* knob, see sysctl(3). If 0, PaX ASLR will be enabled only if explicitly set on programs using paxctl(8). If 1, PaX ASLR will be enabled to all programs, and exemption can be done using paxctl(8).

See security(8) for more details.

amiga-specific Options

options BB060STUPIDROM
When the bootloader (which passes AmigaOS ROM information) claims we have a 68060 CPU without FPU, go look into the Processor Configuration Register (PCR) to find out. You need this with Amiga ROMs up to (at least) V40.xxx (OS3.1), when you boot via the bootblocks and don't have a DraCo.

options IOBZCLOCK=frequency
The IOBlix boards come with two different serial master clocks: older ones use 24 MHz, newer ones use 22.1184 MHz. The driver normally assumes the latter. If your board uses 24 MHz, you can recompile your kernel with options IOBZCLOCK=24000000 or patch the kernel variable iobzclock to the same value.

options LIMITMEM=value
If there, limit the part of the first memory bank used by NetBSD to value megabytes. Default is unlimited.

options NKPTADD=addvalue

options NKPTADDSHIFT=shiftvalue
The CPU specific MMU table for the kernel is pre-allocated at kernel startup time. Part of it is scaled with *maxproc*, to have enough room to hold the user program MMU tables; the second part is a fixed amount for the kernel itself.

The third part accounts for the size of the file buffer cache. Its size is either NKPTADD pages (if defined) or memory size in bytes divided by two to the power of NKPTADDSHIFT. The default is undefined NKPTADD and NKPTADDSHIFT=24, allowing for 16 buffers per megabyte of main memory (while a GENERIC kernel allocates about half of that). When you get "can't get KPT page" panics, you should increase NKPTADD (if defined), or decrease NKPTADDSHIFT by one.

options P5PPC68KBOARD
Add special support for Phase5 mixed 68k+PPC boards. Currently, this only affects rebooting from NetBSD and is only needed on 68040+PPC, not on 68060+PPC; without this, affected machines will hang after NetBSD has shut down and will only restart after a keyboard reset or a power cycle.

arm32-specific Options

options FRENCH_KBD
Include translation for French keyboards when using *pccons* on a Shark.

options FINNISH_KBD
Include translation for Finnish keyboards when using *pccons* on a Shark.

options GERMAN_KBD
Include translation for German keyboards when using *pccons* on a Shark.

options NORWEGIAN_KBD
Include translation for French keyboards when using *pccons* on a Shark.

amd64-specific Options

options ENHANCED_SPEEDSTEP
Include support for the Enhanced SpeedStep Technology present in newer CPUs.

options EST_FREQ_USERWRITE
Allow any user to change the frequency of an Enhanced SpeedStep Technology capable CPU.

options INTEL_ONDEMAND_CLOCKMOD
This enables the On Demand Clock Modulation by software on Intel CPUs supporting the Thermal Monitor feature (TM). You can select the duty cycle with `sysctl`(8) in the node *machdep.clockmod* if supported.

options POWERNOW_K8
Include support for AMD Athlon 64 PowerNow! and Cool‘n’Quiet Technology, used to change the cpu voltage and frequency on the fly.

atari-specific Options

options DISKLABEL_AHDI
Include support for AHDI (native Atari) disklabels.

options DISKLABEL_NBDA
Include support for NetBSD/atari labels. If you don’t set this option, it will be set automatically. NetBSD/atari will not work without it.

options FALCON_SCSI
Include support for the 5380-SCSI configuration as found on the Falcon.

options RELOC_KERNEL
If set, the kernel will relocate itself to TT-RAM, if possible. This will give you a slightly faster system. *Beware* that on some TT030 systems, the system will frequently dump with MMU-faults with this option enabled.

options SERCONSOLE
Allow the modem1-port to act as the system-console. A carrier should be active on modem1 during system boot to active the console functionality.

options TT_SCSI
Include support for the 5380-SCSI configuration as found on the TT030 and Hades.

i386-specific Options

options ENHANCED_SPEEDSTEP
Include support for the Enhanced SpeedStep Technology present in newer CPUs.

options EST_FREQ_USERWRITE
Allow any user to change the frequency of an Enhanced SpeedStep Technology capable CPU.

options INTEL_ONDEMAND_CLOCKMOD
This enables the On Demand Clock Modulation by software on Intel CPUs supporting the Thermal Monitor feature (TM). You can select the duty cycle with `sysctl`(8) in the node *machdep.clockmod* if supported.

options POWERNOW_K7
Include support for the AMD PowerNow! Technology present in AMD Athlon Mobile processors.

options VIA_PADLOCK
Include support for the AES encryption instructions of the VIA PadLock Security engine, which is attached as a provider to the opencrypto framework.

options CPURESET_DELAY=value
Specifies the time (in millisecond) to wait before doing a hardware reset in the last phase of a reboot. This gives the user a chance to see error messages from the shutdown operations (like NFS unmounts, buffer cache flush, etc ...). Setting this to 0 will disable the delay. Default is 2 seconds.

options VM86
Include support for virtual 8086 mode, used by DOS emulators and X servers to run BIOS code, e.g., for some VESA routines.

options USER_LDT
Include i386-specific system calls for modifying the local descriptor table, used by Windows emulators.

options REALBASEMEM=integer
Overrides the base memory size passed in from the boot block. (Value given in kilobytes.) Use this option only if the boot block reports the size incorrectly. (Note that some BIOSes put the extended BIOS data area at the top of base memory, and therefore report a smaller base memory size to prevent programs overwriting it. This is correct behavior, and you should not use the *REALBASEMEM* option to access this memory).

options REALEXTMEM=integer
Overrides the extended memory size passed in from the boot block. (Value given in kilobytes. Extended memory does not include the first megabyte.) Use this option only if the boot block reports the size incorrectly.

options FRENCH_KBD,FINNISH_KBD,GERMAN_KBD,NORWEGIAN_KBD
Select a non-US keyboard layout for the *pccons* console driver.

options CYRIX_CACHE_WORKS
Relevant only to the Cyrix 486DLC CPU. This option is used to turn on the cache in hold-flush mode. It is not turned on by default because it is known to have problems in certain motherboard implementations.

options CYRIX_CACHE_REALLY_WORKS
Relevant only to the Cyrix 486DLC CPU. This option is used to turn on the cache in write-back mode. It is not turned on by default because it is known to have problems in certain motherboard implementations. In order for this option to take effect, option *CYRIX_CACHE_WORKS* must also be specified.

options PCIBIOS
Enable support for initializing the PCI bus using information from the BIOS. See `pcibios`(4) for details.

options KSTACK_CHECK_DR0
Detect kernel stack overflow using DR0 register. This option uses DR0 register exclusively so you can't use DR0 register for other purpose (e.g., hardware breakpoint) if you turn this on.

options MTRR
Include support for accessing MTRR registers from user-space. See i386_get_mtrr(2).

options BEEP_ONHALT
Make the system speaker emit several beeps when it is completely safe to power down the computer after a halt(8) command. Requires sysbeep(4) support.

options BEEP_ONHALT_COUNT=times
Number of times to beep the speaker when **options BEEP_ONHALT** is enabled. Defaults to 3.

options BEEP_ONHALT_PITCH=hz
The tone frequency used when **options BEEP_ONHALT** option, in hertz. Defaults to 1500.

options BEEP_ONHALT_PERIOD=msecs
The duration of each beep when **options BEEP_ONHALT** is enabled, in milliseconds. Defaults to 250.

options MULTIBOOT
Makes the kernel Multiboot-compliant, allowing it to be booted through a Multiboot-compliant boot manager such as GRUB. See multiboot(8) for more information.

isa-specific Options
Options specific to isa(4) busses.

options PCIC_ISA_ALLOC_IOBASE=address, PCIC_ISA_ALLOC_IOSIZE=size
Control the section of IO bus space used for PCMCIA bus space mapping. Ideally the probed defaults are satisfactory, however in practice that is not always the case. See pcmcia(4) for details.

options PCIC_ISA_INTR_ALLOC_MASK=mask
Controls the allowable interrupts that may be used for PCMCIA devices. This mask is a logical-or of power-of-2s of allowable interrupts:

IRQ	*Val*	*IRQ*	*Val*	*IRQ*	*Val*	*IRQ*	*Val*
0	0x0001	4	0x0010	8	0x0100	12	0x1000
1	0x0002	5	0x0020	9	0x0200	13	0x2000
2	0x0004	6	0x0040	10	0x0400	14	0x4000
3	0x0008	7	0x0080	11	0x0800	15	0x8000

options PCKBC_CNATTACH_SELFTEST
Perform a self test of the keyboard controller before attaching it as a console. This might be necessary on machines where we boot on cold iron, and pckbc refuses to talk until we request a self test. Currently only the netwinder port uses it.

options PCKBD_CNATTACH_MAY_FAIL
If this option is set the PS/2 keyboard will not be used as the console if it cannot be found during boot. This allows other keyboards, like USB, to be the console keyboard.

options PCKBD_LAYOUT=layout
Sets the default keyboard layout, see pckbd(4).

m68k-specific Options
options FPU_EMULATE
Include support for MC68881/MC68882 emulator.

options FPSP
Include support for 68040 floating point.

options M68020,M68030,M68040,M68060
Include support for a specific CPU, at least one (the one you are using) should be specified.

options M060SP
Include software support for 68060. This provides emulation of unimplemented integer instructions as well as emulation of unimplemented floating point instructions and data types and software support for floating point traps.

powerpc-specific Options (OEA Only)

options PMAP_MEMLIMIT=value
Limit the amount of memory seen by the kernel to *value* bytes.

options PTEGCOUNT=value
Specify the size of the page table as *value* PTE groups. Normally, one PTEG is allocated per physical page frame.

sparc-specific Options

options AUDIO_DEBUG
Enable simple event debugging of the logging of the audio(4) device.

options BLINK
Enable blinking of LED. Blink rate is full cycle every N seconds for N < then current load average. See getloadavg(3).

options COUNT_SW_LEFTOVERS
Count how many times the sw SCSI device has left 3, 2, 1 and 0 in the sw_3_leftover, sw_2_leftover, sw_1_leftover, and sw_0_leftover variables accessible from ddb(4). See sw(4).

options DEBUG_ALIGN
Adds debugging messages calls when user-requested alignment fault handling happens.

options DEBUG_EMUL
Adds debugging messages calls for emulated floating point and alignment fixing operations.

options DEBUG_SVR4
Prints registers messages calls for emulated SVR4 getcontext and setcontext operations. See *options COMPAT_SVR4*.

options EXTREME_DEBUG
Adds debugging functions callable from ddb(4). The debug_pagetables, test_region and print_fe_map functions print information about page tables for the SUN4M platforms only.

options EXTREME_EXTREME_DEBUG
Adds extra info to *options EXTREME_DEBUG*.

options FPU_CONTEXT
Make *options COMPAT_SVR4* getcontext and setcontext include floating point registers.

options MAGMA_DEBUG
Adds debugging messages to the magma(4) device.

options RASTERCONS_FULLSCREEN
Use the entire screen for the console.

options RASTERCONS_SMALLFONT
Use the Fixed font on the console, instead of the normal font.

options SUN4
Support sun4 class machines.

options SUN4C
Support sun4c class machines.

options SUN4M
Support sun4m class machines.

options SUN4_MMU3L
Enable support for sun4 3-level MMU machines.

options V9
Enable SPARC V9 assembler in ddb(4).

sparc64-specific Options

options AUDIO_DEBUG
Enable simple event debugging of the logging of the audio(4) device.

options BLINK
Enable blinking of LED. Blink rate is full cycle every N seconds for N < then current load average. See getloadavg(3).

x68k-specific Options

options EXTENDED_MEMORY
Include support for extended memory, e.g., TS-6BE16 and 060turbo on-board.

options JUPITER
Include support for Jupiter-X MPU accelerator

options ZSCONSOLE,ZSCN_SPEED=value
Use the built-in serial port as the system-console. Speed is specified in bps, defaults to 9600.

options ITE_KERNEL_ATTR=value
Set the kernel message attribute for ITE. Value, an integer, is a logical or of the following values:

1 color inversed
2 underlined
4 bolded

SEE ALSO

config(1), gdb(1), ktrace(1), pmc(1), quota(1), vndcompress(1), gettimeofday(2), i386_get_mtrr(2), i386_iopl(2), msgctl(2), msgget(2), msgrcv(2), msgsnd(2), ntp_adjtime(2), ntp_gettime(2), semctl(2), semget(2), semop(2), shmat(2), shmctl(2), shmdt(2), shmget(2), sysctl(3), apm(4), ddb(4), inet(4), iso(4), lkm(4), md(4), ns(4), pcibios(4), pcmcia(4), ppp(4), userconf(4), vnd(4), wscons(4), config(5), edquota(8), init(8), mdsetimage(8), mount_cd9660(8), mount_fdesc(8), mount_kernfs(8), mount_lfs(8), mount_mfs(8), mount_msdos(8), mount_nfs(8), mount_ntfs(8), mount_null(8), mount_portal(8), mount_procfs(8), mount_udf(8), mount_umap(8), mount_union(8), mrouted(8), newfs_lfs(8), ntpd(8), quotaon(8), rpc.rquotad(8), sysctl(8), in_getifa(9)

HISTORY

The **options** man page first appeared in NetBSD 1.3.

BUGS

The *EON* option should be a pseudo-device, and is also very fragile.

NAME

osiop — Symbios/NCR 53C710 SCSI driver

SYNOPSIS

arc

```
osiop* at jazzio? flags 0x00000
```

ews4800mips

```
osiop* at sbdio? flags 0x00000
```

hp700

```
osiop0 at gsc? flags 0x00000
```

mvme68k

```
osiop0 at pcctwo? ipl 2
```

```
scsibus* at osiop?
```

DESCRIPTION

The **osiop** driver provides support for the Symbios/NCR 53C710 SCSI controller chip.

For the Symbios/NCR 53C700 SCSI host adapters, use the oosiop(4) driver.

For the Symbios/NCR 53C8xx PCI SCSI host adapters, use the siop(4) or esiop(4) driver.

CONFIGURATION

The **osiop** driver supports the following **flags** for use in config(1) files:

bits 0-7:	disable disconnect/reselect for the corresponding SCSI target
bits 8-15:	disable synchronous negotiation for SCSI target
bits 16:	disable DMA interrupts

"Target" is synonymous with SCSI ID number.

Note that SCSI tape drives should be allowed to perform disconnect/reselect or performance will suffer.

SEE ALSO

cd(4), ch(4), esiop(4), intro(4), oosiop(4), scsi(4), sd(4), siop(4), ss(4), st(4), uk(4), scsipi(9)

HISTORY

osiop driver first appeared in NetBSD 1.6.

The original NCR 53C710 driver appeared in NetBSD 1.0 amiga port, and Izumi Tsutsui ⟨tsutsui@NetBSD.org⟩ modified the driver and made it machine-independent.

NAME

owtemp — 1-Wire temperature family type device

SYNOPSIS

```
owtemp* at onewire?
```

DESCRIPTION

The **owtemp** driver provides support for the 1-Wire temperature sensor. The sensor possesses a single temperature value that can be accessed through the envsys(4) interface.

The following chips are supported by the driver:

- Maxim/Dallas DS18B20, DS1822, DS1920

SEE ALSO

envsys(4), intro(4), onewire(4), envstat(8)

HISTORY

The **owtemp** driver first appeared in OpenBSD 4.0 and NetBSD 4.0.

AUTHORS

The **owtemp** driver was written by Alexander Yurchenko ⟨grange@openbsd.org⟩ and was ported to NetBSD by Jeff Rizzo ⟨riz@NetBSD.org⟩.

NAME

pad — Pseudo audio device driver

SYNOPSIS

```
pseudo-device pad
audio* at audiobus?
```

DESCRIPTION

pad is a pseudo-device driver which provides support for feeding back PCM data from a consumer of the audio(4) API to userland.

EXAMPLES

The following example streams an MP3 to an Apple AirTunes compatible device:

```
$ rtunes - < /dev/pad0 &
$ mpg123 -a /dev/sound1 mozart.mp3
```

FILES

The **pad** pseudo-device driver receives data from /dev/soundN and feeds the raw PCM output to /dev/padN.

/dev/soundN

/dev/padN

SEE ALSO

audio(4)

HISTORY

The **pad** driver appeared in NetBSD 5.0.

AUTHORS

Jared D. McNeill ⟨jmcneill@invisible.ca⟩

NAME

pas — ProAudio Spectrum audio device driver

SYNOPSIS

```
pas0   at isa? port 0x220 irq 7 drq 1
audio* at audiobus?
```

DESCRIPTION

The **pas** driver provides support for ProAudio Spectrum sound cards.

SEE ALSO

audio(4), isa(4)

NAME

pbms — PowerBook (and iBook) trackpad mouse support

SYNOPSIS

```
pbms? at uhidev? reportid ?
wsmouse* at pbms?
```

DESCRIPTION

The **pbms** driver provides support for the trackpads on new (post February 2005) Apple PowerBooks and iBooks that are not standard USB HID mice. Access to the trackpad is through the wscons(4) framework.

SEE ALSO

uhidev(4), usb(4), wsmouse(4)

HISTORY

The **pbms** device driver appeared in NetBSD 4.0.

AUTHORS

The **pbms** driver was written by Johan Wallén.

NAME

pcc — MVME147 Peripheral Channel Controller device

SYNOPSIS

```
pcc0 at mainbus0
```

DESCRIPTION

The **pcc** interface serves as an abstraction used by the autoconf(4) system to help find and attach devices whose resources are accessed via the PCC chip. (e.g. the Ethernet and SCSI controllers)

DIAGNOSTICS

pcc0 at mainbus0. This is the normal autoconfiguration message indicating that the PCC chip has been found and attached to the main processor bus.

SEE ALSO

autoconf(4), mainbus(4), pcctwo(4)

NAME

pcctwo — MVME167/MVME177 Peripheral Channel Controller 2 device

SYNOPSIS

```
pcctwo0 at mainbus0
```

DESCRIPTION

The **pcctwo** interface serves as an abstraction used by the autoconf(4) system to help find and attach devices whose resources are accessed via the PCCchip2 found on Motorola MVME167 and MVME177 single board computers. (e.g. the Ethernet and SCSI controllers)

DIAGNOSTICS

pcctwo0 at mainbus0. This is the normal autoconfiguration message indicating that the PCCchip2 has been found and attached to the main processor bus.

SEE ALSO

autoconf(4), mainbus(4), pcc(4)

NAME

pcdisplay — PC display adapter driver for wscons

SYNOPSIS

```
options PCDISPLAY_SOFTCURSOR

pcdisplay0 at isa?
wsdisplay* at pcdisplay? console ?
```

DESCRIPTION

This driver supports PC display adapter hardware within the wscons(4) console framework. It doesn't provide direct device driver entry points but makes its functions available via the internal wsdisplay(4) interface.

The **pcdisplay** driver is intended as a minimal "catch-all" driver for the different kinds of MDA or CGA compatible adapters. It doesn't support multiple screens, nor colors or font loading.

Supported kernel option(s):

options PCDISPLAY_SOFTCURSOR
: Use a large, non-blinking cursor generated by software. The default is to use the cursor provided by the underlying display hardware.

SEE ALSO

isa(4), vga(4), wscons(4)

NAME

pci — introduction to machine-independent PCI bus support and drivers

SYNOPSIS

```
pci* at mainbus? bus ?
pci* at pchb? bus ?
pci* at ppb? bus ?

options PCIVERBOSE
options PCI_CONFIG_DUMP
options PCI_ADDR_FIXUP
options PCI_BUS_FIXUP
options PCI_INTR_FIXUP
```

DESCRIPTION

Other **pci** attachments are machine-dependent and depend on the bus topology and PCI bus interface of your system. See intro(4) for your system for details.

NetBSD includes a machine-independent PCI bus subsystem and several machine-independent PCI device drivers.

Your system may support additional PCI devices. Drivers for PCI devices not listed here are machine-dependent. Consult your system's intro(4) for additional information.

OPTIONS

PCI_ADDR_FIXUP Fixup PCI I/O and memory addresses.

Some i386 and amd64 BIOS implementations don't allocate I/O space and memory space for some PCI devices -- primarily BIOS in PnP mode, or laptops that expect devices to be configured via ACPI. Since necessary space isn't allocated, those devices will not work without special handling.

This option allocates I/O space and memory space instead of relying upon the BIOS to do so.

If necessary space is already correctly assigned to the devices, this option leaves the space as is.

PCI_BUS_FIXUP Fixup PCI bus numbering; needed for many cardbus(4) bridges.

Each PCI bus and CardBus should have a unique bus number. But some BIOS implementations don't assign a bus number for subordinate PCI buses. And many BIOS implementations don't assign a bus number for CardBuses.

A typical symptom of this is the following boot message:

cardbus0 at cardslot0: bus 0 device 0...

Please note that this cardbus0 has a bus number '0', but normally the bus number 0 is used by the machine's primary PCI bus. Thus, this bus number for cardbus is incorrect (not assigned). In this situation, a device located in cardbus0 doesn't show correct device ID, because its bus number 0 incorrectly refers to the primary PCI bus, and a device ID in the primary PCI bus is shown in the boot message instead of the device's ID in the cardbus0.

This option assigns bus numbers for all subordinate PCI buses and CardBuses.

Since this option renumbers all PCI buses and CardBuses, all bus numbers of subordinate buses become different when this option is enabled.

`PCI_INTR_FIXUP` Fixup PCI interrupt routing via PCIBIOS or ACPI.

Some i386 and amd64 BIOS implementations don't assign an interrupt for some devices.

This option assigns an interrupt for such devices instead of relying upon the BIOS to do so.

If a valid interrupt has already been assigned to a device, this option leaves the interrupt as is.

HARDWARE

NetBSD includes machine-independent PCI drivers, sorted by device type and driver name:

SCSI interfaces

ahc	Adaptec 29xx, 39xx, and other AIC-7xxx-based SCSI interfaces.
adv	Advansys SCSI interfaces.
adw	Advansys Ultra Wide SCSI interfaces.
bha	Buslogic BT-9xx SCSI interfaces.
dpt	DPT SmartCache/SmartRAID III and IV SCSI interfaces.
iha	Initio INIC-940/950 SCSI interfaces.
isp	QLogic ISP-1020, ISP-1040, and ISP-2100 SCSI and FibreChannel interfaces.
mfi	LSI Logic & Dell MegaRAID SAS RAID controllers.
mly	Mylex AcceleRAID and eXtremeRAID controllers with v6 firmware.
pcscp	Advanced Micro Devices Am53c974 PCscsi-PCI SCSI interfaces.
siop	Symbios Logic/NCR 53c8xx-family SCSI interfaces.
trm	Tekram TRM-S1040 ASIC based SCSI interfaces.

Disk and tape controllers

aac	The Adaptec AAC family of RAID controllers.
ahcisata	AHCI 1.0 and 1.1 compliant SATA controllers.
amr	The AMI and LSI Logic MegaRAID family of RAID controllers.
cac	Compaq array controllers.
icp	ICP Vortex GDT and Intel Storage RAID controllers.
mlx	Mylex DAC960 and DEC SWXCR RAID controllers.
pciide	IDE disk controllers.
twe	3Ware Escalade RAID controllers.

Network interfaces

an	Aironet 4500/4800 and Cisco 340 series 802.11 interfaces.
bnx	Broadcom NetXtreme II 10/100/1000 Ethernet interfaces.

de	DEC DC21x4x (Tulip) based Ethernet interfaces, including the DE435, DE450, and DE500, and Znyx, SMC, Cogent/Adaptec, and Asante single- and multi-port Ethernet interfaces.
en	Midway-based Efficient Networks Inc. and Adaptec ATM interfaces.
ep	3Com 3c590, 3c595, 3c900, and 3c905 Ethernet interfaces.
epic	SMC83C170 (EPIC/100) Ethernet interfaces.
esh	RoadRunner-based HIPPI interfaces.
ex	3Com 3c900, 3c905, and 3c980 Ethernet interfaces.
fpa	DEC DEFPA FDDI interfaces.
fxp	Intel EtherExpress PRO 10+/100B Ethernet interfaces.
gsip	National Semiconductor DP83820 based Gigabit Ethernet interfaces.
hme	Sun Microelectronics STP2002-STQ Ethernet interfaces.
le	PCNet-PCI Ethernet interfaces. Note, the pcn(4) driver supersedes this driver.
lmc	LAN Media Corp WAN interfaces.
msk	Marvell Yukon 2 based Gigabit Ethernet interfaces.
ne	NE2000-compatible Ethernet interfaces.
nfe	NVIDIA nForce Ethernet interfaces.
ntwoc	SDL Communications N2pci and WAN/ic 400 synchronous serial interfaces.
pcn	AMD PCnet-PCI family of Ethernet interfaces.
ral	Ralink Technology RT2500/RT2600-based 802.11a/b/g wireless network interfaces.
rtk	Realtek 8129/8139 based Ethernet interfaces.
sf	Adaptec AIC-6915 10/100 Ethernet interfaces.
sip	Silicon Integrated Systems SiS 900, SiS 7016, and National Semiconductor DP83815 based Ethernet interfaces.
sk	SysKonnect SK-98xx based Gigabit Ethernet interfaces.
ste	Sundance ST-201 10/100 based Ethernet interfaces.
stge	Sundance/Tamarack TC9021 based Gigabit Ethernet interfaces.
ti	Alteon Networks Tigon I and Tigon II Gigabit Ethernet driver.
tl	Texas Instruments ThunderLAN-based Ethernet interfaces.
tlp	DECchip 21x4x and clone Ethernet interfaces.
vge	VIA Networking Technologies VT6122 PCI Gigabit Ethernet adapter driver.
vr	VIA VT3043 (Rhine) and VT86C100A (Rhine-II) Ethernet interfaces.
wi	WaveLAN/IEEE and PRISM-II 802.11 wireless interfaces.
wm	Intel i8254x Gigabit Ethernet driver.

Serial interfaces

cy	Cyclades Cyclom-4Y, -8Y, and -16Y multi-port serial interfaces.
cz	Cyclades-Z series multi-port serial interfaces.

Audio devices

auacer	Acer Labs M5455 I/O Controller Hub integrated AC'97 audio device.
auich	Intel I/O Controller Hub integrated AC'97 audio device.
auvia	VIA VT82C686A integrated AC'97 audio device.
autri	Trident 4DWAVE-DX/NX, SiS 7018, ALi M5451 AC'97 audio device.
clcs	Cirrus Logic CS4280 audio device.
clct	Cirrus Logic CS4281 audio device.
cmpci	C-Media CMI8x38 audio device.
eap	Ensoniq AudioPCI audio device.
emuxki	Creative Labs SBLive! and PCI 512 audio device.
esa	ESS Technology Allegro-1 / Maestro-3 audio device.
esm	ESS Maestro-1/2/2e PCI AC'97 Audio Accelerator audio device.
eso	ESS Solo-1 PCI AudioDrive audio device.
fms	Forte Media FM801 audio device.
neo	NeoMagic MagicMedia 256 audio device.
sv	S3 SonicVibes audio device.
yds	Yamaha YMF724/740/744/754-based audio device.

Bridges

cbb	PCI Yenta compatible CardBus bridges.
ppb	Generic PCI−PCI bridges, including PCI expansion backplanes.

Miscellaneous devices

bktr	Brooktree 848 compatible TV cards.
ehci	USB EHCI host controllers.
iop	I2O I/O processors.
mr	Guillemot Maxi Radio FM 2000 FM radio device.
oboe	Toshiba OBOE IrDA SIR/FIR controller.
ohci	USB OHCI host controllers.
pcic	PCI PCMCIA controllers, including the Cirrus Logic GD6729.
puc	PCI "universal" communications cards, containing `com`(4) and `lpt`(4) communications ports.
uhci	USB UHCI host controllers.

viapm VIA VT82C686A hardware monitors.

vga VGA graphics boards.

SEE ALSO

aac(4), adv(4), adw(4), agp(4), ahc(4), ahcisata(4), amr(4), an(4), auich(4), autri(4), auvia(4), bha(4), bktr(4), bnx(4), cac(4), cbb(4), clcs(4), cmpci(4), cy(4), cz(4), de(4), dpt(4), eap(4), ehci(4), emuxki(4), en(4), ep(4), epic(4), esa(4), esh(4), esm(4), eso(4), ex(4), fms(4), fpa(4), fxp(4), gsip(4), hme(4), icp(4), iha(4), intro(4), iop(4), isp(4), le(4), lmc(4), mfi(4), mlx(4), mly(4), mpt(4), msk(4), ne(4), neo(4), nfe(4), ntwoc(4), oboe(4), ohci(4), pcic(4), pciide(4), pcn(4), pcscp(4), ppb(4), puc(4), ral(4), rtk(4), sf(4), siop(4), sip(4), sk(4), ste(4), stge(4), sv(4), ti(4), tl(4), tlp(4), trm(4), twe(4), uhci(4), vga(4), vge(4), viapm(4), vr(4), wi(4), wm(4), wscons(4), yds(4)

HISTORY

The machine-independent PCI subsystem appeared in NetBSD 1.2.

NAME

pcibios — introduction to PCI BIOS support

SYNOPSIS

```
options   PCIBIOS
options   PCIBIOSVERBOSE
#options  PCIBIOS_IRQS_HINT=0x0a00 #IRQ 9,11
#options  PCIBIOS_INTR_FIXUP_FORCE
options   PCIBIOS_INTR_GUESS
#options  PCIINTR_DEBUG
```

INTRODUCTION

pcibios provides support for setting up PCI controllers, bridges, and devices using information extracted from the BIOS.

Ideally, the boot firmware of a machine (a.k.a. BIOS) should set up all PCI devices; assigning them I/O and memory addresses and interrupts. Alas, this does not always happen, so there is some PC specific code that can do the initialization when NetBSD boots.

Options:

PCIBIOS turn on the PCI BIOS support.

PCIBIOSVERBOSE make the setup procedure verbose.

PCIBIOS_IRQS_HINT hint for IRQ use. When *PCI_INTR_FIXUP* cannot guess an adequate IRQ for a device, the hint is used.

The value is a logical or of power-of-2s of allowable interrupts:

IRQ Value	*IRQ Value*	*IRQ Value*	*IRQ Value*
0 0x0001	4 0x0010	8 0x0100	12 0x1000
1 0x0002	5 0x0020	9 0x0200	13 0x2000
2 0x0004	6 0x0040	10 0x0400	14 0x4000
3 0x0008	7 0x0080	11 0x0800	15 0x8000

For example, "**options PCIBIOS_IRQS_HINT=0x0a00**" allows IRQ 9 and IRQ 11.

The kernel global variable *pcibios_irqs_hint* holds this value, so a user can override this value without kernel recompilation. For example:

- To specify this value on the fly, type the following command at the boot prompt to drop into DDB (the in-kernel debugger; you have to specify "**options DDB**" to make kernel with DDB):

  ```
  boot -d
  ```

  And type the following command on "**db>**" prompt:

  ```
  write pcibios_irqs_hint 0x0a00
  ```

  Then type the following to continue to boot:

  ```
  c
  ```

- To modify kernel image without kernel recompilation, run the following command on shell:

  ```
  gdb --write /netbsd
  ```

  And type the following commands at the "**(gdb)**" prompt:

  ```
  set pcibios_irqs_hint=0xa00
  quit
  ```

PCIBIOS_INTR_FIXUP_FORCE
Some buggy BIOS implementations provide inconsistent information between the PCI Interrupt Configuration Register and the PCI Interrupt Routing table. In such case, the PCI Interrupt Configuration Register takes precedence by default. If this happens, a kernel with *PCIBIOSVERBOSE* shows "**WARNING: preserving irq XX**" in the PCI routing table.

If *PCIBIOS_INTR_FIXUP_FORCE* is specified in addition to the *PCI_INTR_FIXUP*, the PCI Interrupt Routing table takes precedence. In this case, a kernel with *PCIBIOSVERBOSE* shows "**WARNING: overriding irq XX**" in the PCI routing table.

PCIBIOS_INTR_GUESS
make *PCI_INTR_FIXUP* work with unknown interrupt router.

If a PCI interrupt router is not known, normally interrupt configuration will not be touched.

But if *PCIBIOS_INTR_GUESS* is specified in addition to the *PCI_INTR_FIXUP*, and if a PCI interrupt routing table entry indicates that only one IRQ is available for the entry, the IRQ is assumed to be already connected to the device, and corresponding PCI Interrupt Configuration Register will be configured accordingly.

PCIINTR_DEBUG make the *PCI_INTR_FIXUP* procedure verbose.

SEE ALSO
`cardbus`(4), `pci`(4)

HISTORY
The **pcibios** code appeared in NetBSD 1.5.

BUGS
The *PCIBIOS_ADDR_FIXUP* option may conflict with the PCI CardBus driver's own address fixup.

NAME

pcic — Intel and Cirrus Logic PCMCIA controller driver

SYNOPSIS

```
pcic0   at isa? port 0x3e0 iomem 0xd0000 iosiz 0x4000 flags N
pcic1   at isa? port 0x3e2 iomem 0xd4000 iosiz 0x4000 flags N
pcic*   at isapnp?
pcic*   at pci? dev? function ?
pcmcia* at pcic? controller ? socket ?
```

DESCRIPTION

NetBSD provides support for the Intel 82365SL, Cirrus Logic PD6710 and PD672x PCMCIA controllers.

For the isa(4) attachment a *flags* value of 1 can be used to force the use of polling instead of interrupts for card events.

The default configuration of the pcic gives each controller 16 kilobytes of memory, to be shared between slots. Some PC Card devices require somewhat more memory than this; it may therefore be necessary to adjust the *iomem* and *iosiz* parameters of the **pcic** devices in the kernel config file to accommodate these cards.

SEE ALSO

isa(4), isapnp(4), pci(4), pcmcia(4), tcic(4)

http://www.intel.com/
http://www.cirrus.com/

HISTORY

The **pcic** driver appeared in NetBSD 1.3.

NAME

pciide — PCI IDE disk controllers driver

SYNOPSIS

```
pciide* at pci? dev ? function ? flags 0x0000
pciide* at pnpbios? index ?
```

DESCRIPTION

The **pciide** driver supports the PCI IDE controllers as specified in the "PCI IDE controller specification, revision 1.0" draft, and provides the interface with the hardware for the **ata** driver. Please use the chip-specific drivers for the following controllers for enhanced and DMA support:

- Acard ATP850 (Ultra/33) and ATP860 (Ultra/66) IDE Controllers: acardide(4)
- Acer labs M5229 IDE Controller: aceride(4)
- Advanced Micro Devices AMD-756, 766, and 768 IDE Controllers: viaide(4)
- Advanced Micro Devices Geode IDE Controllers: geodeide(4)
- CMD Tech PCI0643, PCI0646, PCI0648, and PCI0649 IDE Controllers: cmdide(4)
- Contaq Microsystems/Cypress CY82C693 IDE Controller: cypide(4)
- HighPoint HPT366 Ultra/66, HPT370 Ultra/100, HPT372, and HPT374 Ultra/133 IDE controller: hptide(4)
- Intel PIIX, PIIX3, and PIIX4 IDE Controllers: piixide(4)
- Intel i31244 Serial ATA controller: artsata(4)
- Intel 82801 (ICH/ICH0/ICH2/ICH3/ICH4/ICH5/ICH6) IDE Controllers: piixide(4)
- NVIDIA nForce/nForce2 IDE Controllers: viaide(4)
- OPTi 82c621 (plus a few of its derivatives) IDE Controllers: optiide(4)
- Promise PDC20246 (Ultra/33), PDC20262 (Ultra/66), PDC20265/PDC20267 (Ultra100), PDC20268 (Ultra/100TX2 and Ultra/100TX2v2), Ultra/133, Ultra/133TX2 and Ultra/133TX2v2 PCI IDE controllers: pdcide(4)
- Serverworks K2 SATA controllers: svwsata(4)
- Silicon Image 0680 IDE controller: cmdide(4)
- Silicon Image SATALink 3112 Serial ATA controller: satalink(4)
- Silicon Image SteelVine 3124/3132/3531 Serial ATA II controller: siisata(4)
- Silicon Integrated System 5597/5598 IDE controller: siside(4)
- Symphony Labs 82C105 and Winbond W83C553F IDE controller: slide(4)
- VIA Technologies VT82C586, VT82C586A, VT82C596A, VT82C686A, VT8233A, and VT8235 IDE Controllers: viaide(4)

The 0x0001 flag forces the **pciide** driver to use DMA when there is no explicit DMA mode setting support for the controller but DMA is present. If the BIOS didn't set up the controller properly, this can cause a machine hang.

The 0x0002 flag forces the **pciide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

acardide(4), aceride(4), artsata(4), ata(4), atapi(4), cmdide(4), cypide(4), geodeide(4), hptide(4), intro(4), optiide(4), pci(4), pdcide(4), piixide(4), pnpbios(4), satalink(4), siisata(4), siside(4), slide(4), viaide(4), wd(4), wdc(4)

NAME

pckbc — PC (ISA) keyboard controller driver

SYNOPSIS

```
pckbc* at acpi?
pckbc* at isa?
pckbc* at pnpbios? index ?
pckbd* at pckbc? slot ?
pms*   at pckbc? slot ?
```

DESCRIPTION

The **pckbc** driver handles resource allocation and device attachment for the traditional PC/AT keyboard controller. It provides two logical connections for child devices, the “keyboard” slot for a keyboard and the “auxiliary” slot for mice (the latter might be missing in older keyboard controllers).

The optional “slot” locator argument can be used to force unusual connections of devices to logical slots. This feature is for experimentation only, it will not be useful in normal operation.

SEE ALSO

acpi(4), isa(4), pckbd(4), pms(4), pnpbios(4)

NAME

pckbd — PC keyboard driver for wscons

SYNOPSIS

```
pckbc* at isa?
pckbd* at pckbc?
wskbd* at pckbd? console ?
options PCKBD_LAYOUT=XXX
```

DESCRIPTION

This driver supports PC/AT keyboards within the wscons(4) console framework. It doesn't provide direct device driver entry points but makes its functions available via the internal wskbd(4) interface.

The **pckbd** driver supports a number of different key mappings which can be chosen from with the kernel option PCKBD_LAYOUT at compile time or with the utility wsconsctl(8) (variable: "encoding") at run-time. Other mappings can be used if the whole keymap is replaced by means of wsconsctl(8). The builtin mappings are at this time:

option	wsconsctl	language
KB_US	us	English/US keyboard mapping (default)
KB_DE	de	German with "dead accents"
KB_FR	fr	French
KB_DK	dk	Danish with "dead accents"
KB_IT	it	Italian
KB_UK	uk	British
KB_JP	jp	Japanese
KB_SV	sv	Swedish with "dead accents"
KB_US|KB_DECLK	us.declk	English/US mapping for DEC LK400-style keyboards with PC keyboard interface (e.g., LK461)
KB_US|KB_DVORAK	us.dvorak	English/US keyboard with "Dvorak" layout
KB_US|KB_COLEMAK	us.colemak	English/US keyboard with "Colemak" layout
KB_NO	no	Norwegian with "dead accents"
KB_PT	pt	Portuguese
KB_ES	es	Spanish

The KB_DE, KB_DK, KB_NO and KB_SV mappings can be used in the KB_NODEAD (".nodead") variant. This switches off the "dead accents".

The KB_US, KB_JP and KB_US|KB_DVORAK mappings can be modified to swap the left CTRL and the CAPS LOCK keys by the KB_SWAPCTRLCAPS variant bit or the ".swapctrlcaps" suffix.

The KB_METAESC (".metaesc") option can be applied to any layout. If set, keys pressed together with the ALT modifier are prefixed by an ESC character. (Standard behaviour is to add 128 to the ASCII value.)

Because PC keyboard hardware doesn't contain a beeper, requests for "keyboard beeps" cannot be handled directly. On alpha and i386 a helper device attached to the pcppi(4) driver allows the use of the standard ISA speaker for this purpose. On acorn32, vidcaudio(4) performs this function.

EXAMPLES

To set a German keyboard layout without "dead accents" and sending an ESC character before the key symbol if the ALT key is pressed simultaneously, use **wsconsctl -w** *encoding=de.nodead.metaesc*. To set it at kernel build time, add **options PCKBD_LAYOUT="(KB_DE | KB_NODEAD |**

KB_METAESC)"
to the kernel configuration file.

SEE ALSO

`isa`(4), `pcppi`(4), `wskbd`(4), `wsconsctl`(8)

BUGS

The list of builtin mappings doesn't follow any logic. It grew as people submitted what they needed.

NAME

pcl — DEC CSS PCL-11 B Network Interface

SYNOPSIS

```
pcl0 at uba? csr 164200 vector pclxint pclrint
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **pcl** device provides an IP-only interface to the DEC CSS PCL-11 time division multiplexed network bus. The controller itself is not accessible to users.

The host's address is specified with the `SIOCSIFADDR` `ioctl`(2). The interface will not transmit or receive any data before its address is defined.

As the PCL-11 hardware is only capable of having 15 interfaces per network, a single-byte host-on-network number is used, with range [1..15] to match the TDM bus addresses of the interfaces.

The interface currently only supports the Internet protocol family and only provides "natural" (header) encapsulation.

DIAGNOSTICS

pcl%d: can't init. Insufficient UNIBUS resources existed to initialize the device. This is likely to occur when the device is run on a buffered data path on an 11/750 and other network interfaces are also configured to use buffered data paths, or when it is configured to use buffered data paths on an 11/730 (which has none).

pcl%d: can't handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

pcl%d: stray xmit interrupt. An interrupt occurred when no output had previously been started.

pcl%d: master. The TDM bus had no station providing ''bus master'' timing signals, so this interface has assumed the ''master'' role. This message should only appear at most once per UNIBUS INIT on a single system. Unless there is a hardware failure, only one station may be master at a time.

pcl%d: send error, tcr=%b, tsr=%b. The device indicated a problem sending data on output. If a ''receiver offline'' error is detected, it is not normally logged unless the option `PCL_TESTING` has been selected, as this causes a lot of console chatter when sending to a down machine. However, this option is quite useful when debugging problems with the PCL interfaces.

pcl%d: rcv error, rcr=%b rsr=%b. The device indicated a problem receiving data on input.

pcl%d: bad len=%d. An input operation resulted in a data transfer of less than 0 or more than 1008 bytes of data into memory (according to the word count register). This should never happen as the maximum size of a PCL message has been agreed upon to be 1008 bytes (same as ARPANET message).

SEE ALSO

`inet`(4), `intro`(4)

HISTORY

The **pcl** interface appeared in 4.2BSD.

NAME

pcmcia — introduction to PCMCIA (PC Card) support

SYNOPSIS

```
pcmcia* at pcic? controller ? socket ?
pcmcia* at tcic? controller ? socket ?
pcmcia* at cardslot?

options PCMCIAVERBOSE
```

amiga

```
pcmcia* at pccard0
```

hpcmips

```
pcmcia* at it8368e? controller ? socket ?
pcmcia* at plumpcmcia? controller ? socket ?
```

hpcsh

```
pcmcia* at hd64461pcmcia? controller ? socket ?
```

sh3

```
pcmcia* at shpcic? controller ? socket ?
```

sparc

```
pcmcia* at nell?
```

DESCRIPTION

NetBSD provides machine-independent bus support and drivers for PCMCIA (Personal Computer Memory Card International Association) a.k.a. PC Card, CardBus devices.

HARDWARE

NetBSD includes the following machine-independent PCMCIA drivers, sorted by function and driver name:

Serial interfaces and modems

com	8250/16450/16550-compatible PCMCIA serial cards and modems.

Network interfaces

an	Aironet 4500/4800 and Cisco 340 series 802.11 controller.
awi	802.11 controller based on the AMD PCnetMobile chipset.
cnw	Netwave AirSurfer Wireless LAN interface.
ep	3Com 3c589 EtherLink III Ethernet card.
mbe	Ethernet card based on the Fujitsu MB86960A/MB86965A chipset.
mhzc	Megahertz Ethernet/Modem combo cards
ne	NE2000 compatible cards.
ray	Raytheon Raylink and WebGear Aviator2.4 802.11 controller.
sm	Megahertz Ethernet card.

wi Lucent WaveLAN/IEEE and PRISM-II based 802.11 controller.

xi Xircom CreditCard Ethernet

SCSI controllers

aic Adaptec APA-1460 SCSI controller card.

esp NCR 53C9x, Emulex ESP406, and Qlogic FAS408 SCSI controllers.

spc Fujitsu MB87030/MB89352 SCSI controllers.

IDE controllers

wdc Digital Hinote Ultra Mobile Media Adapter

Audio devices

esl Eiger Labs ESS1688 AudioDrive.

Bluetooth devices

bt3c 3Com 3CRWB6096 Bluetooth PC Card driver.

btbc AnyCom Bluetooth BlueCard driver.

USB Controller

slhci Cypress/ScanLogic SL811HS USB Host Controller driver.

SEE ALSO

aic(4), an(4), awi(4), bt3c(4), btbc(4), cardbus(4), cnw(4), com(4), ep(4), esl(4), esp(4), intro(4), isa(4), mbe(4), mhzc(4), ne(4), options(4), pcic(4), pcmcom(4), ray(4), slhci(4), sm(4), spc(4), tcic(4), wi(4), xi(4)

http://www.pcmcia.org/

HISTORY

The **pcmcia** driver appeared in NetBSD 1.3.

BUGS

IO space conflicts

NetBSD probes the PCMCIA IO bus width and uses that information to decide where to map PCMCIA IO space. For 10-bit wide cards, 0x300-0x3ff is used. For 12-bit wide cards, 0x400-0x4ff is used.

Neither choice is perfect. In the 12-bit case, 0x400 appears to work on substantially more devices than 0x300. In the event that PCMCIA devices are mapped in 0x400-0x4ff and appear to be nonfunctional, remapping to 0x300-0x3ff may be appropriate; consult **options PCIC_ISA_ALLOC_IOBASE** and **options PCIC_ISA_ALLOC_IOSIZE** in options(4). Example:

```
# Avoid PCMCIA bus space conflicts with the default IO space
# allocation on 12-bit wide busses (base 0x300 size 0xff).
options PCIC_ISA_ALLOC_IOBASE=0x300
options PCIC_ISA_ALLOC_IOSIZE=0x0ff
```

Interrupt conflicts

NetBSD attempts to probe for available interrupts to assign to PCMCIA devices. In some cases, it is not possible to detect all interrupts in use; in such cases, use of **options PCIC_ISA_INTR_ALLOC_MASK** may be necessary. See options(4).

Unconfigured devices

During autoconfiguration, if a message is displayed saying that your card is "not configured" it indicates that there isn't support for your card compiled into the kernel. To fix this problem, it may simply be a matter of adding the manufacturer and product IDs to the PCMCIA database or adding a front-end attachment to an existing driver. In the latter case, it is normally always necessary to get a dump of the CIS table from the card. You can do this by adding **options PCMCIACISDEBUG** and **options PCMCIADEBUG** into your kernel config file. Additionally, you will have to patch the kernel to enable run-time debugging. This can be done in the source by changing the variables *pcmcia_debug* and *pcmciacis_debug* to 0xff. Alternatively, you can patch the same variables at run-time using ddb(4). For most drivers you should also consider enabling any driver-specific debugging options.

NAME

pcmcom — PCMCIA multi-port serial card driver

SYNOPSIS

```
pcmcom* at pcmcia? function ?
com*    at pcmcom? slave ?
```

DESCRIPTION

The **pcmcom** driver provides support for Megahertz XJ2288 modem.

SEE ALSO

com(4), pcmcia(4)

HISTORY

The **pcmcom** driver appeared in NetBSD 1.3.

NAME

pcn — AMD PCnet-PCI Ethernet family driver

SYNOPSIS

pcn* **at pci? dev ? function ?**

Configuration of PHYs may also be necessary. See mii(4).

DESCRIPTION

The **pcn** device driver supports Ethernet interfaces based on the AMD PCnet-PCI family of Ethernet chips. The chips supported by the **pcn** driver include:

- Am79c970 PCnet-PCI Single-Chip Ethernet Controller for PCI Local Bus
- Am79c970A PCnet-PCI II Single-Chip Full-Duplex Ethernet Controller for PCI Local Bus
- Am79c971 PCnet-FAST Single-Chip Full-Duplex 10/100Mbps Ethernet Controller for PCI Local Bus
- Am79c972 PCnet-FAST+ Enhanced 10/100Mbps PCI Ethernet Controller with OnNow Support
- Am79c973/Am79c975 PCnet-FAST III Single-Chip 10/100Mbps PCI Ethernet Controller with Integrated PHY

PCnet-PCI chips are found on some Hewlett-Packard PCI Ethernet boards, and on the Allied Telesyn AT-2700TX PCI Ethernet board. They are also found on some processor evaluation boards as an example peripheral.

The **pcn** driver also supports the emulated PCnet-PCI interface provided by VMware.

SEE ALSO

arp(4), ifmedia(4), mii(4), netintro(4), pci(4), ifconfig(8)

HISTORY

The **pcn** driver first appeared in NetBSD 1.6.

AUTHORS

The **pcn** driver was written by Jason R. Thorpe ⟨thorpej@wasabisystems.com⟩.

NAME

pcppi — PC (ISA) control port driver

SYNOPSIS

```
pcppi*   at acpi?
pcppi*   at isa?
isabeep* at pcppi?
```
(alpha only)
```
sysbeep* at pcppi?
```
(i386 only)
```
spkr0    at pcppi?
midi*    at pcppi?
```

DESCRIPTION

The **pcppi** driver handles resource allocation and device attachment for the ports related to the ISA speaker in the traditional PC/AT “design”. These are the “system control port” (which was implemented by the 8255 “PPI” in the XT, hence the name of this driver) at IO address 0x61.

When associated with an attimer(4) device, it is possible to change the pitch of the sounds emitted through **pcppi**.

The **pcppi** driver provides its child devices with the ability to output simple tones through the PC speaker. The speaker(4) and midi(4) devices use this to synthesize sounds. The isabeep(4) and sysbeep(4) devices are helpers which the pckbd(4) driver uses as a substitute for a “keyboard beep”, because the PC keyboard hardware doesn’t provide this.

SEE ALSO

acpi(4), attimer(4), isa(4), midi(4), pckbd(4), speaker(4)

NAME

pcscp — Advanced Micro Devices Am53c974 PCscsi-PCI SCSI driver

SYNOPSIS

```
pcscp* at pci? dev ? function ?
scsibus* at pcscp?
```

DESCRIPTION

The **pcscp** driver provides support for the Advanced Micro Devices Am53c974 PCscsi-PCI SCSI controller and boards using this chip, including the Tekram DC-390 PCI SCSI host adapter.

For Tekram DC-390U/UW/F PCI SCSI host adapters, use the siop(4) driver.

For Tekram DC-395U/UW/F PCI SCSI host adapters, use the trm(4) driver.

SEE ALSO

cd(4), ch(4), esp(4), intro(4), pci(4), scsi(4), sd(4), ss(4), st(4), uk(4), scsipi(9)

http://www.amd.com/

BUGS

The driver currently ignores EEPROM settings, which establish per-target parameters etc.

NAME

pcweasel — Support for the PC-Weasel serial console board

SYNOPSIS

```
pseudo-device pcweasel
weasel* at pci? dev ? function ?
```

Note that the appropriate display device must also be enabled. See pcdisplay(4) for more information.

DESCRIPTION

The PC-Weasel is a serial console board for use primarily on Intel-based PC-class systems. It addresses a problem that nearly everyone who has deployed a PC-class server has experienced: the total lack of remote management capability on PC-class hardware.

In addition to serial console support, the PC-Weasel provides the ability to remotely reset the system (by means of a hardware reset signal), and provides a watchdog timer function.

The PC-Weasel works by emulating the original IBM Monochrome Display Adapter (MDA). Writes to the display's character cells are translated into ANSI terminal sequences which are then sent out the PC-Weasel's serial port. Incoming characters are translated into PC keyboard scan codes and then fed (by means of a cable) into the system's keyboard controller. The system believes it is using a display console. This is particularly important in the event that one needs access to BIOS configuration menus.

The PC-Weasel also includes a ST16550 serial port, which may be configured as any one of the system's serial ports. Typical usage is to configure the port as *com0* at ISA I/O address 0x3f8. When the PC-Weasel detects activity on the ST16550, the serial port is automatically connected to the ST16550 so that the serial port may be used as normal. When the PC-Weasel detects activity on the internal UART used for MDA emulation, the serial port is automatically reconnected to the emulation UART. This allows the boot program and kernel to be configured to use the serial port directly (which is more efficient than using the MDA emulation mode), yet allows the MDA emulation to be reestablished as soon as the kernel loses control of the system.

The **pcweasel** driver provides support for the additional features present on the PC-Weasel. At the moment, this includes support for the watchdog timer function. Use of the **pcweasel** driver is not required in order for the system to function with a PC-Weasel installed so long as only the MDA emulation and ST16550 serial port functionality is required.

SEE ALSO

pcdisplay(4), wdogctl(8)

HISTORY

The **pcweasel** driver first appeared in NetBSD 1.5.1.

AUTHORS

The PC-Weasel was invented by Herb Peyerl and Jonathan Levine at Canada Connect Corporation. It is now produced by Middle Digital, Inc., http://www.realweasel.com/

The **pcweasel** driver was written by Jason R. Thorpe ⟨thorpej@zembu.com⟩, and contributed by Zembu Labs, Inc. Herb Peyerl of Middle Digital, Inc. provided several firmware updates during the development of the driver.

NAME

pdcide — Promise IDE disk controllers driver

SYNOPSIS

```
pdcide* at pci? dev ? function ? flags 0x0000
```

DESCRIPTION

The **pdcide** driver supports the Promise Ultra33, Ultra66, Ultra100, Ultra100TX2, Ultra100TX2v2, Ultra133, Ultra133TX2, Ultra133TX2v2, Fasttrak133 and Serial ATA/150 IDE controllers, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **pdcide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

The timings used for the PIO and DMA modes for controllers listed above are for a PCI bus running at 30 or 33 MHz. This driver may not work properly on overclocked systems.

The **pdcide** driver does NOT function correctly on NetBSD/sparc64.

NAME

pdcsata — Promise Serial-ATA disk controllers driver

SYNOPSIS

```
pdcsata* at pci? dev ? function ? flags 0x0000
```

DESCRIPTION

The **pdcsata** driver supports the Promise SATA150 (PDC20571, PDC20575, PDC20579, PDC20318, PDC20319, PDC20371, PDC20375, PDC20376, PDC20377, PDC20378, and PDC20379) and SATA300 (PDC20775, PDC40518, PDC40519, PDC40718 , PDC40719 and PDC40779) families of serial-ATA controllers, and provides the interface with the hardware for the ata(4) driver.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

NAME

pf — packet filter

SYNOPSIS

```
pseudo-device pf
```

DESCRIPTION

Packet filtering takes place in the kernel. A pseudo-device, `/dev/pf`, allows userland processes to control the behavior of the packet filter through an `ioctl`(2) interface. There are commands to enable and disable the filter, load rulesets, add and remove individual rules or state table entries, and retrieve statistics. The most commonly used functions are covered by `pfctl`(8).

Manipulations like loading a ruleset that involve more than a single `ioctl`(2) call require a so-called *ticket*, which prevents the occurrence of multiple concurrent manipulations.

Fields of `ioctl`(2) parameter structures that refer to packet data (like addresses and ports) are generally expected in network byte-order.

Rules and address tables are contained in so-called *anchors*. When servicing an `ioctl`(2) request, if the anchor field of the argument structure is empty, the kernel will use the default anchor (i.e., the main ruleset) in operations. Anchors are specified by name and may be nested, with components separated by '/' characters, similar to how file system hierarchies are laid out. The final component of the anchor path is the anchor under which operations will be performed.

IOCTL INTERFACE

pf supports the following `ioctl`(2) commands, available through ⟨`net/pfvar.h`⟩:

`DIOCSTART`
: Start the packet filter.

`DIOCSTOP`
: Stop the packet filter.

`DIOCSTARTALTQ`
: Start the ALTQ bandwidth control system (see `altq`(9)).

`DIOCSTOPALTQ`
: Stop the ALTQ bandwidth control system.

`DIOCBEGINADDRS` *struct pfioc_pooladdr *pp*

```
struct pfioc_pooladdr {
        u_int32_t               action;
        u_int32_t               ticket;
        u_int32_t               nr;
        u_int32_t               r_num;
        u_int8_t                r_action;
        u_int8_t                r_last;
        u_int8_t                af;
        char                    anchor[MAXPATHLEN];
        struct pf_pooladdr      addr;
};
```

Clear the buffer address pool and get a *ticket* for subsequent `DIOCADDADDR`, `DIOCADDRULE`, and `DIOCCHANGERULE` calls.

`DIOCADDADDR` *`struct pfioc_pooladdr *pp`*

Add the pool address *addr* to the buffer address pool to be used in the following `DIOCADDRULE` or `DIOCCHANGERULE` call. All other members of the structure are ignored.

`DIOCADDRULE` *`struct pfioc_rule *pr`*

```
struct pfioc_rule {
        u_int32_t      action;
        u_int32_t      ticket;
        u_int32_t      pool_ticket;
        u_int32_t      nr;
        char           anchor[MAXPATHLEN];
        char           anchor_call[MAXPATHLEN];
        struct pf_rule rule;
};
```

Add *rule* at the end of the inactive ruleset. This call requires a *ticket* obtained through a preceding `DIOCXBEGIN` call and a *pool_ticket* obtained through a `DIOCBEGINADDRS` call. `DIOCADDADDR` must also be called if any pool addresses are required. The optional *anchor* name indicates the anchor in which to append the rule. *nr* and *action* are ignored.

`DIOCADDALTQ` *`struct pfioc_altq *pa`*

Add an ALTQ discipline or queue.

```
struct pfioc_altq {
        u_int32_t      action;
        u_int32_t      ticket;
        u_int32_t      nr;
        struct pf_altq  altq;
};
```

`DIOCGETRULES` *`struct pfioc_rule *pr`*

Get a *ticket* for subsequent `DIOCGETRULE` calls and the number *nr* of rules in the active ruleset.

`DIOCGETRULE` *`struct pfioc_rule *pr`*

Get a *rule* by its number *nr* using the *ticket* obtained through a preceding `DIOCGETRULES` call.

`DIOCGETADDRS` *`struct pfioc_pooladdr *pp`*

Get a *ticket* for subsequent `DIOCGETADDR` calls and the number *nr* of pool addresses in the rule specified with *r_action*, *r_num*, and *anchor*.

`DIOCGETADDR` *`struct pfioc_pooladdr *pp`*

Get the pool address *addr* by its number *nr* from the rule specified with *r_action*, *r_num*, and *anchor* using the *ticket* obtained through a preceding `DIOCGETADDRS` call.

`DIOCGETALTQS` *`struct pfioc_altq *pa`*

Get a *ticket* for subsequent `DIOCGETALTQ` calls and the number *nr* of queues in the active list.

`DIOCGETALTQ` *`struct pfioc_altq *pa`*

Get the queueing discipline *altq* by its number *nr* using the *ticket* obtained through a preceding `DIOCGETALTQS` call.

`DIOCGETQSTATS` *`struct pfioc_qstats *pq`*

Get the statistics on a queue.

```
struct pfioc_qstats {
        u_int32_t       ticket;
        u_int32_t       nr;
```

```
        void           *buf;
        int             nbytes;
        u_int8_t        scheduler;
};
```

This call fills in a pointer to the buffer of statistics *buf*, of length *nbytes*, for the queue specified by *nr*.

`DIOCGETRULESETS` *`struct pfioc_ruleset *pr`*

```
struct pfioc_ruleset {
        u_int32_t       nr;
        char            path[MAXPATHLEN];
        char            name[PF_ANCHOR_NAME_SIZE];
};
```

Get the number *nr* of rulesets (i.e., anchors) directly attached to the anchor named by *path* for use in subsequent `DIOCGETRULESET` calls. Nested anchors, since they are not directly attached to the given anchor, will not be included. This ioctl returns `EINVAL` if the given anchor does not exist.

`DIOCGETRULESET` *`struct pfioc_ruleset *pr`*

Get a ruleset (i.e., an anchor) *name* by its number *nr* from the given anchor *path*, the maximum number of which can be obtained from a preceding `DIOCGETRULESETS` call. This ioctl returns `EINVAL` if the given anchor does not exist or `EBUSY` if another process is concurrently updating a ruleset.

`DIOCADDSTATE` *`struct pfioc_state *ps`*

Add a state entry.

```
struct pfioc_state {
        u_int32_t       nr;
        struct pf_state         state;
};
```

`DIOCGETSTATE` *`struct pfioc_state *ps`*

Extract the entry with the specified number *nr* from the state table.

`DIOCKILLSTATES` *`struct pfioc_state_kill *psk`*

Remove matching entries from the state table. This ioctl returns the number of killed states in *psk_af*.

```
struct pfioc_state_kill {
        sa_family_t             psk_af;
        int                     psk_proto;
        struct pf_rule_addr     psk_src;
        struct pf_rule_addr     psk_dst;
        char                    psk_ifname[IFNAMSIZ];
};
```

`DIOCCLRSTATES` *`struct pfioc_state_kill *psk`*

Clear all states. It works like `DIOCKILLSTATES`, but ignores the *psk_af*, *psk_proto*, *psk_src*, and *psk_dst* fields of the *`pfioc_state_kill`* structure.

`DIOCSETSTATUSIF` *`struct pfioc_if *pi`*

Specify the interface for which statistics are accumulated.

```
struct pfioc_if {
        char            ifname[IFNAMSIZ];
};
```

DIOCGETSTATUS *struct pf_status *s*
Get the internal packet filter statistics.

```
struct pf_status {
        u_int64_t      counters[PFRES_MAX];
        u_int64_t      lcounters[LCNT_MAX];
        u_int64_t      fcounters[FCNT_MAX];
        u_int64_t      scounters[SCNT_MAX];
        u_int64_t      pcounters[2][2][3];
        u_int64_t      bcounters[2][2];
        u_int64_t      stateid;
        u_int32_t      running;
        u_int32_t      states;
        u_int32_t      src_nodes;
        u_int32_t      since;
        u_int32_t      debug;
        u_int32_t      hostid;
        char           ifname[IFNAMSIZ];
};
```

DIOCCLRSTATUS
Clear the internal packet filter statistics.

DIOCNATLOOK *struct pfioc_natlook *pnl*
Look up a state table entry by source and destination addresses and ports.

```
struct pfioc_natlook {
        struct pf_addr  saddr;
        struct pf_addr  daddr;
        struct pf_addr  rsaddr;
        struct pf_addr  rdaddr;
        u_int16_t       sport;
        u_int16_t       dport;
        u_int16_t       rsport;
        u_int16_t       rdport;
        sa_family_t     af;
        u_int8_t        proto;
        u_int8_t        direction;
};
```

DIOCSETDEBUG *u_int32_t *level*
Set the debug level.

```
enum    { PF_DEBUG_NONE, PF_DEBUG_URGENT, PF_DEBUG_MISC,
          PF_DEBUG_NOISY };
```

DIOCGETSTATES *struct pfioc_states *ps*
Get state table entries.

```
struct pfioc_states {
        int     ps_len;
        union {
```

```
                caddr_t         psu_buf;
                struct pf_state *psu_states;
        } ps_u;
#define ps_buf          ps_u.psu_buf
#define ps_states       ps_u.psu_states
};
```

If *ps_len* is zero, all states will be gathered into *pf_states* and *ps_len* will be set to the size they take in memory (i.e., `sizeof(struct pf_state) * nr`). If *ps_len* is non-zero, as many states that can fit into *ps_len* as possible will be gathered, and *ps_len* will be updated to the size those rules take in memory.

`DIOCCHANGERULE` *`struct pfioc_rule *pcr`*

Add or remove the *rule* in the ruleset specified by *rule.action*.

The type of operation to be performed is indicated by *action*, which can be any of the following:

```
enum    { PF_CHANGE_NONE, PF_CHANGE_ADD_HEAD, PF_CHANGE_ADD_TAIL,
          PF_CHANGE_ADD_BEFORE, PF_CHANGE_ADD_AFTER,
          PF_CHANGE_REMOVE, PF_CHANGE_GET_TICKET };
```

ticket must be set to the value obtained with `PF_CHANGE_GET_TICKET` for all actions except `PF_CHANGE_GET_TICKET`. *pool_ticket* must be set to the value obtained with the `DIOCBEGINADDRS` call for all actions except `PF_CHANGE_REMOVE` and `PF_CHANGE_GET_TICKET`. *anchor* indicates to which anchor the operation applies. *nr* indicates the rule number against which `PF_CHANGE_ADD_BEFORE`, `PF_CHANGE_ADD_AFTER`, or `PF_CHANGE_REMOVE` actions are applied.

`DIOCCHANGEADDR` *`struct pfioc_pooladdr *pca`*

Add or remove the pool address *addr* from the rule specified by *r_action*, *r_num*, and *anchor*.

`DIOCSETTIMEOUT` *`struct pfioc_tm *pt`*

```
struct pfioc_tm {
        int             timeout;
        int             seconds;
};
```

Set the state timeout of *timeout* to *seconds*. The old value will be placed into *seconds*. For possible values of *timeout*, consult the `PFTM_*` values in ⟨`net/pfvar.h`⟩.

`DIOCGETTIMEOUT` *`struct pfioc_tm *pt`*

Get the state timeout of *timeout*. The value will be placed into the *seconds* field.

`DIOCCLRRULECTRS`

Clear per-rule statistics.

`DIOCSETLIMIT` *`struct pfioc_limit *pl`*

Set the hard limits on the memory pools used by the packet filter.

```
struct pfioc_limit {
        int             index;
        unsigned        limit;
};

enum { PF_LIMIT_STATES, PF_LIMIT_SRC_NODES, PF_LIMIT_FRAGS };
```

`DIOCGETLIMIT` *`struct pfioc_limit *pl`*
Get the hard *limit* for the memory pool indicated by *index*.

`DIOCRCLRTABLES` *`struct pfioc_table *io`*
Clear all tables. All the ioctls that manipulate radix tables use the same structure described below. For `DIOCRCLRTABLES`, *pfrio_ndel* contains on exit the number of tables deleted.

```
struct pfioc_table {
        struct pfr_table        pfrio_table;
        void                   *pfrio_buffer;
        int                     pfrio_esize;
        int                     pfrio_size;
        int                     pfrio_size2;
        int                     pfrio_nadd;
        int                     pfrio_ndel;
        int                     pfrio_nchange;
        int                     pfrio_flags;
        u_int32_t               pfrio_ticket;
};
#define pfrio_exists    pfrio_nadd
#define pfrio_nzero     pfrio_nadd
#define pfrio_nmatch    pfrio_nadd
#define pfrio_naddr     pfrio_size2
#define pfrio_setflag   pfrio_size2
#define pfrio_clrflag   pfrio_nadd
```

`DIOCRADDTABLES` *`struct pfioc_table *io`*
Create one or more tables. On entry, *pfrio_buffer[pfrio_size]* contains a table of *`pfr_table`* structures. On exit, *pfrio_nadd* contains the number of tables effectively created.

```
struct pfr_table {
        char            pfrt_anchor[MAXPATHLEN];
        char            pfrt_name[PF_TABLE_NAME_SIZE];
        u_int32_t       pfrt_flags;
        u_int8_t        pfrt_fback;
};
```

`DIOCRDELTABLES` *`struct pfioc_table *io`*
Delete one or more tables. On entry, *pfrio_buffer[pfrio_size]* contains a table of *`pfr_table`* structures. On exit, *pfrio_nadd* contains the number of tables effectively deleted.

`DIOCRGETTABLES` *`struct pfioc_table *io`*
Get the list of all tables. On entry, *pfrio_buffer[pfrio_size]* contains a valid writeable buffer for *`pfr_table`* structures. On exit, *pfrio_size* contains the number of tables written into the buffer. If the buffer is too small, the kernel does not store anything but just returns the required buffer size, without error.

`DIOCRGETTSTATS` *`struct pfioc_table *io`*
This call is like `DIOCRGETTABLES` but is used to get an array of *`pfr_tstats`* structures.

```
struct pfr_tstats {
        struct pfr_table pfrts_t;
        u_int64_t       pfrts_packets
                            [PFR_DIR_MAX][PFR_OP_TABLE_MAX];
        u_int64_t       pfrts_bytes
                            [PFR_DIR_MAX][PFR_OP_TABLE_MAX];
```

```
        u_int64_t       pfrts_match;
        u_int64_t       pfrts_nomatch;
        long            pfrts_tzero;
        int             pfrts_cnt;
        int             pfrts_refcnt[PFR_REFCNT_MAX];
};
#define pfrts_name      pfrts_t.pfrt_name
#define pfrts_flags     pfrts_t.pfrt_flags
```

`DIOCRCLRTSTATS` *`struct pfioc_table *io`*

Clear the statistics of one or more tables. On entry, *pfrio_buffer[pfrio_size]* contains a table of *`pfr_table`* structures. On exit, *pfrio_nzero* contains the number of tables effectively cleared.

`DIOCRCLRADDRS` *`struct pfioc_table *io`*

Clear all addresses in a table. On entry, *pfrio_table* contains the table to clear. On exit, *pfrio_ndel* contains the number of addresses removed.

`DIOCRADDADDRS` *`struct pfioc_table *io`*

Add one or more addresses to a table. On entry, *pfrio_table* contains the table ID and *pfrio_buffer[pfrio_size]* contains the list of *`pfr_addr`* structures to add. On exit, *pfrio_nadd* contains the number of addresses effectively added.

```
struct pfr_addr {
        union {
                struct in_addr  _pfra_ip4addr;
                struct in6_addr         _pfra_ip6addr;
        }               pfra_u;
        u_int8_t        pfra_af;
        u_int8_t        pfra_net;
        u_int8_t        pfra_not;
        u_int8_t        pfra_fback;
};
#define pfra_ip4addr    pfra_u._pfra_ip4addr
#define pfra_ip6addr    pfra_u._pfra_ip6addr
```

`DIOCRDELADDRS` *`struct pfioc_table *io`*

Delete one or more addresses from a table. On entry, *pfrio_table* contains the table ID and *pfrio_buffer[pfrio_size]* contains the list of *`pfr_addr`* structures to delete. On exit, *pfrio_ndel* contains the number of addresses effectively deleted.

`DIOCRSETADDRS` *`struct pfioc_table *io`*

Replace the content of a table by a new address list. This is the most complicated command, which uses all the structure members.

On entry, *pfrio_table* contains the table ID and *pfrio_buffer[pfrio_size]* contains the new list of *`pfr_addr`* structures. Additionally, if *pfrio_size2* is non-zero, *pfrio_buffer[pfrio_size..pfrio_size2]* must be a writeable buffer, into which the kernel can copy the addresses that have been deleted during the replace operation. On exit, *pfrio_ndel*, *pfrio_nadd*, and *pfrio_nchange* contain the number of addresses deleted, added, and changed by the kernel. If *pfrio_size2* was set on entry, *pfrio_size2* will point to the size of the buffer used, exactly like `DIOCRGETADDRS`.

`DIOCRGETADDRS` *`struct pfioc_table *io`*

Get all the addresses of a table. On entry, *pfrio_table* contains the table ID and *pfrio_buffer[pfrio_size]* contains a valid writeable buffer for *`pfr_addr`* structures. On exit, *pfrio_size* contains the number of addresses written into the buffer. If the buffer was too small, the

kernel does not store anything but just returns the required buffer size, without returning an error.

`DIOCRGETASTATS` *`struct pfioc_table *io`*

This call is like `DIOCRGETADDRS` but is used to get an array of *`pfr_astats`* structures.

```
struct pfr_astats {
        struct pfr_addr         pfras_a;
        u_int64_t       pfras_packets
                            [PFR_DIR_MAX][PFR_OP_ADDR_MAX];
        u_int64_t       pfras_bytes
                            [PFR_DIR_MAX][PFR_OP_ADDR_MAX];
        long            pfras_tzero;
};
```

`DIOCRCLRASTATS` *`struct pfioc_table *io`*

Clear the statistics of one or more addresses. On entry, *pfrio_table* contains the table ID and *pfrio_buffer[pfrio_size]* contains a table of *`pfr_addr`* structures to clear. On exit, *pfrio_nzero* contains the number of addresses effectively cleared.

`DIOCRTSTADDRS` *`struct pfioc_table *io`*

Test if the given addresses match a table. On entry, *pfrio_table* contains the table ID and *pfrio_buffer[pfrio_size]* contains a table of *`pfr_addr`* structures to test. On exit, the kernel updates the *`pfr_addr`* table by setting the *pfra_fback* member appropriately.

`DIOCRSETTFLAGS` *`struct pfioc_table *io`*

Change the `PFR_TFLAG_CONST` or `PFR_TFLAG_PERSIST` flags of a table. On entry, *pfrio_buffer[pfrio_size]* contains a table of *`pfr_table`* structures, and *pfrio_setflag* contains the flags to add, while *pfrio_clrflag* contains the flags to remove. On exit, *pfrio_nchange* and *pfrio_ndel* contain the number of tables altered or deleted by the kernel. Yes, tables can be deleted if one removes the `PFR_TFLAG_PERSIST` flag of an unreferenced table.

`DIOCRINADEFINE` *`struct pfioc_table *io`*

Defines a table in the inactive set. On entry, *pfrio_table* contains the table ID and *pfrio_buffer[pfrio_size]* contains the list of *`pfr_addr`* structures to put in the table. A valid ticket must also be supplied to *pfrio_ticket*. On exit, *pfrio_nadd* contains 0 if the table was already defined in the inactive list or 1 if a new table has been created. *pfrio_naddr* contains the number of addresses effectively put in the table.

`DIOCXBEGIN` *`struct pfioc_trans *io`*

```
struct pfioc_trans {
        int             size;  /* number of elements */
        int             esize; /* size of each element in bytes */
        struct pfioc_trans_e {
                int             rs_num;
                char            anchor[MAXPATHLEN];
                u_int32_t       ticket;
        }              *array;
};
```

Clear all the inactive rulesets specified in the *`pfioc_trans_e`* array. For each ruleset, a ticket is returned for subsequent "add rule" ioctls, as well as for the `DIOCXCOMMIT` and `DIOCXROLLBACK` calls.

Ruleset types, identified by *rs_num*, include the following:

PF_RULESET_SCRUB	Scrub (packet normalization) rules.
PF_RULESET_FILTER	Filter rules.
PF_RULESET_NAT	NAT (Network Address Translation) rules.
PF_RULESET_BINAT	Bidirectional NAT rules.
PF_RULESET_RDR	Redirect rules.
PF_RULESET_ALTQ	ALTQ disciplines.
PF_RULESET_TABLE	Address tables.

DIOCXCOMMIT *struct pfioc_trans *io*

Atomically switch a vector of inactive rulesets to the active rulesets. This call is implemented as a standard two-phase commit, which will either fail for all rulesets or completely succeed. All tickets need to be valid. This ioctl returns EBUSY if another process is concurrently updating some of the same rulesets.

DIOCXROLLBACK *struct pfioc_trans *io*

Clean up the kernel by undoing all changes that have taken place on the inactive rulesets since the last DIOCXBEGIN. DIOCXROLLBACK will silently ignore rulesets for which the ticket is invalid.

DIOCSETHOSTID *u_int32_t *hostid*

Set the host ID, which is used by pfsync(4) to identify which host created state table entries.

DIOCOSFPFLUSH

Flush the passive OS fingerprint table.

DIOCOSFPADD *struct pf_osfp_ioctl *io*

```
struct pf_osfp_ioctl {
        struct pf_osfp_entry {
                SLIST_ENTRY(pf_osfp_entry) fp_entry;
                pf_osfp_t               fp_os;
                char                    fp_class_nm[PF_OSFP_LEN];
                char                    fp_version_nm[PF_OSFP_LEN];
                char                    fp_subtype_nm[PF_OSFP_LEN];
        }                       fp_os;
        pf_tcpopts_t            fp_tcpopts;
        u_int16_t               fp_wsize;
        u_int16_t               fp_psize;
        u_int16_t               fp_mss;
        u_int16_t               fp_flags;
        u_int8_t                fp_optcnt;
        u_int8_t                fp_wscale;
        u_int8_t                fp_ttl;
        int                     fp_getnum;
};
```

Add a passive OS fingerprint to the table. Set *fp_os.fp_os* to the packed fingerprint, *fp_os.fp_class_nm* to the name of the class (Linux, Windows, etc), *fp_os.fp_version_nm* to the name of the version (NT, 95, 98), and *fp_os.fp_subtype_nm* to the name of the subtype or patchlevel. The members *fp_mss*, *fp_wsize*, *fp_psize*, *fp_ttl*, *fp_optcnt*, and *fp_wscale* are set to the TCP MSS, the TCP window size, the IP length, the IP TTL, the number of TCP options, and the TCP window scaling constant of the TCP SYN packet, respectively.

The *fp_flags* member is filled according to the ⟨net/pfvar.h⟩ include file PF_OSFP_* defines. The *fp_tcpopts* member contains packed TCP options. Each option uses PF_OSFP_TCPOPT_BITS bits in the packed value. Options include any of PF_OSFP_TCPOPT_NOP, PF_OSFP_TCPOPT_SACK, PF_OSFP_TCPOPT_WSCALE,

PF_OSFP_TCPOPT_MSS, or PF_OSFP_TCPOPT_TS.

The *fp_getnum* member is not used with this ioctl.

The structure's slack space must be zeroed for correct operation; memset(3) the whole structure to zero before filling and sending to the kernel.

DIOCOSFPGET *struct pf_osfp_ioctl *io*

Get the passive OS fingerprint number *fp_getnum* from the kernel's fingerprint list. The rest of the structure members will come back filled. Get the whole list by repeatedly incrementing the *fp_getnum* number until the ioctl returns EBUSY.

DIOCGETSRCNODES *struct pfioc_src_nodes *psn*

```
struct pfioc_src_nodes {
        int     psn_len;
        union {
                caddr_t         psu_buf;
                struct pf_src_node     *psu_src_nodes;
        } psn_u;
#define psn_buf              psn_u.psu_buf
#define psn_src_nodes  psn_u.psu_src_nodes
};
```

Get the list of source nodes kept by sticky addresses and source tracking. The ioctl must be called once with *psn_len* set to 0. If the ioctl returns without error, *psn_len* will be set to the size of the buffer required to hold all the *pf_src_node* structures held in the table. A buffer of this size should then be allocated, and a pointer to this buffer placed in *psn_buf*. The ioctl must then be called again to fill this buffer with the actual source node data. After that call, *psn_len* will be set to the length of the buffer actually used.

DIOCCLRSRCNODES

Clear the tree of source tracking nodes.

DIOCIGETIFACES *struct pfioc_iface *io*

Get the list of interfaces and interface drivers known to **pf**. All the ioctls that manipulate interfaces use the same structure described below:

```
struct pfioc_iface {
        char                    pfiio_name[IFNAMSIZ];
        void                   *pfiio_buffer;
        int                     pfiio_esize;
        int                     pfiio_size;
        int                     pfiio_nzero;
        int                     pfiio_flags;
};

#define PFI_FLAG_GROUP     0x0001  /* gets groups of interfaces */
#define PFI_FLAG_INSTANCE  0x0002  /* gets single interfaces */
#define PFI_FLAG_ALLMASK   0x0003
```

If not empty, *pfiio_name* can be used to restrict the search to a specific interface or driver. *pfiio_buffer[pfiio_size]* is the user-supplied buffer for returning the data. On entry, *pfiio_size* represents the number of *pfi_if* entries that can fit into the buffer. The kernel will replace this value by the real number of entries it wants to return. *pfiio_esize* should be set to sizeof(struct pfi_if). *pfiio_flags* should be set to PFI_FLAG_GROUP, PFI_FLAG_INSTANCE, or both, to tell the kernel to return a group of interfaces (drivers, like "fxp"), real interface instances (like

"fxp1") or both. The data is returned in the *pfi_if* structure described below:

```
struct pfi_if {
        char                            pfif_name[IFNAMSIZ];
        u_int64_t                       pfif_packets[2][2][2];
        u_int64_t                       pfif_bytes[2][2][2];
        u_int64_t                       pfif_addcnt;
        u_int64_t                       pfif_delcnt;
        long                            pfif_tzero;
        int                             pfif_states;
        int                             pfif_rules;
        int                             pfif_flags;
};

#define PFI_IFLAG_GROUP             0x0001  /* group of interfaces */
#define PFI_IFLAG_INSTANCE          0x0002  /* single instance */
#define PFI_IFLAG_CLONABLE          0x0010  /* clonable group */
#define PFI_IFLAG_DYNAMIC           0x0020  /* dynamic group */
#define PFI_IFLAG_ATTACHED          0x0040  /* interface attached */
```

DIOCICLRISTATS *struct pfioc_iface *io*

Clear the statistics counters of one or more interfaces. *pfiio_name* and *pfiio_flags* can be used to select which interfaces need to be cleared. The filtering process is the same as for DIOCIGETIFACES. *pfiio_nzero* will be set by the kernel to the number of interfaces and drivers that have been cleared.

DIOCSETIFFLAG *struct pfioc_iface *io*

Set the user setable flags (described below) of the pf internal interface description. The filtering process is the same as for DIOCIGETIFACES.

```
#define PFI_IFLAG_SKIP          0x0100  /* skip interface */
#define PFI_IFLAG_SETABLE_MASK          0x0100  /* mask */
```

DIOCCLRIFFLAG *struct pfioc_iface *io*

Works as DIOCSETIFFLAG above but clears the flags.

FILES

/dev/pf packet filtering device.

EXAMPLES

The following example demonstrates how to use the DIOCNATLOOK command to find the internal host/port of a NATed connection:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <sys/fcntl.h>
#include <net/if.h>
#include <netinet/in.h>
#include <net/pfvar.h>
#include <err.h>
#include <stdio.h>
#include <stdlib.h>

u_int32_t
```

```
read_address(const char *s)
{
        int a, b, c, d;

        sscanf(s, "%i.%i.%i.%i", &a, &b, &c, &d);
        return htonl(a << 24 | b << 16 | c << 8 | d);
}

void
print_address(u_int32_t a)
{
        a = ntohl(a);
        printf("%d.%d.%d.%d", a >> 24 & 255, a >> 16 & 255,
            a >> 8 & 255, a & 255);
}

int
main(int argc, char *argv[])
{
        struct pfioc_natlook nl;
        int dev;

        if (argc != 5) {
                printf("%s <gwy addr> <gwy port> <ext addr> <ext port>\n",
                    argv[0]);
                return 1;
        }

        dev = open("/dev/pf", O_RDWR);
        if (dev == -1)
                err(1, "open(\"/dev/pf\") failed");

        memset(&nl, 0, sizeof(struct pfioc_natlook));
        nl.saddr.v4.s_addr    = read_address(argv[1]);
        nl.sport              = htons(atoi(argv[2]));
        nl.daddr.v4.s_addr    = read_address(argv[3]);
        nl.dport              = htons(atoi(argv[4]));
        nl.af                 = AF_INET;
        nl.proto              = IPPROTO_TCP;
        nl.direction          = PF_IN;

        if (ioctl(dev, DIOCNATLOOK, &nl))
                err(1, "DIOCNATLOOK");

        printf("internal host ");
        print_address(nl.rsaddr.v4.s_addr);
        printf(":%u\n", ntohs(nl.rsport));
        return 0;
}
```

SEE ALSO

`ioctl`(2), `bridge`(4), `pflog`(4), `pfctl`(8), `altq`(9)

HISTORY

The **`pf`** packet filtering mechanism first appeared in OpenBSD 3.0.

CAVEATS

The following functionality is missing from **`pf`** in this version of NetBSD:

- The pfsync protocol is not supported.
- The *group* keyword is not supported.

NAME

pflog — packet filter logging interface

SYNOPSIS

```
pseudo-device pflog
```

DESCRIPTION

The **pflog** interface is a pseudo-device which makes visible all packets logged by the packet filter, pf(4). Logged packets can easily be monitored in real time by invoking tcpdump(8) on the **pflog** interface, or stored to disk using pflogd(8).

Each packet retrieved on this interface has a header associated with it of length PFLOG_HDRLEN. This header documents the address family, interface name, rule number, reason, action, and direction of the packet that was logged. This structure, defined in ⟨net/if_pflog.h⟩ looks like

```
struct pfloghdr {
        u_int8_t        length;
        sa_family_t     af;
        u_int8_t        action;
        u_int8_t        reason;
        char            ifname[IFNAMSIZ];
        char            ruleset[PF_RULESET_NAME_SIZE];
        u_int32_t       rulenr;
        u_int32_t       subrulenr;
        u_int8_t        dir;
        u_int8_t        pad[3];
};
```

EXAMPLES

```
# ifconfig pflog0 up
# tcpdump -n -e -ttt -i pflog0
```

SEE ALSO

inet(4), inet6(4), netintro(4), pf(4), ifconfig(8), pflogd(8), tcpdump(8)

HISTORY

The **pflog** device first appeared in OpenBSD 3.0.

NAME

phantomas — Phantom PseudoBC GSC+ Port

SYNOPSIS

```
phantomas* at mainbus0
sti*       at phantomas?
lasi*      at phantomas?
dino*      at phantomas?
```

DESCRIPTION

Phantom bus convertor used to connect a sti(4) graphics, a lasi(4) bus host adapter for gsc(4) and a PCI bridge dino(4) to the system bus, where cpu(4) and memory are located.

MACHINES

The Phantom bus convertor is used in conjunction with the PA7300LC CPU in this machines:

- 744/*
- 748/*
- A180[C]
- B132L[+], B160L, B180L+
- C132L, C160L
- D220, D230, D320, D330
- RDI PrecisionBook

SEE ALSO

cpu(4), dino(4), gsc(4), intro(4), lasi(4), sti(4)

HISTORY

The **phantomas** driver appeared in OpenBSD 3.3. It was ported to NetBSD 2.0 by Jochen Kunz.

NAME

pic — Processor Interface Controller

SYNOPSIS

```
pic0 at mainbus0 addr 0x1fa00000
```

DESCRIPTION

The Processor Interface Controller interfaces the gio(4) bus (VME bus optional) to main memory and CPU. The **pic** is found in the Personal Iris 4D/3x and Indigo R3k machines.

SEE ALSO

gio(4)

HISTORY

The **pic** driver first appeared in NetBSD 2.0.

NAME

piixide — Intel IDE/SATA disk controllers driver

SYNOPSIS

```
piixide* at pci? dev ? function ? flags 0x0000
```

DESCRIPTION

The **piixide** driver supports the Intel PIIX, PIIX3, PIIX4, and 82801 (ICH/ICH0/ICH2/ICH3/ICH4/ICH5/ICH6/ICH7/ICH8/ICH9) IDE/SATA controllers and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **piixide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

The timings used for the PIO and DMA modes for controllers listed above are for a PCI bus running at 30 or 33 MHz. This driver may not work properly on overclocked systems.

NAME

piixpcib — Intel PIIX4 PCI-ISA bridge with SpeedStep

SYNOPSIS

```
piixpcib* at pci? dev ? function ?
isa* at piixpcib?
```

DESCRIPTION

The **piixpcib** driver provides support for the Intel PIIX and compatible PCI-ISA Bridges with Intel's first generation SpeedStep.

Frequency scaling is supported on Pentium III with two voltage modes, used by SpeedStep as power states low and high. The driver will switch into low power state by reducing voltage and frequency of the CPU. The factor depends on the processor itself, but will always reduce power consumption about 1/2.

The user can manually control the CPU frequency with the `sysctl`(8) program using the following node:

machdep.speedstep_state = [0/1]

SEE ALSO

`isa`(4), `pci`(4), `apmd`(8), `sysctl`(8)

HISTORY

The **piixpcib** driver first appeared in FreeBSD 5.5 and then in NetBSD 4.0.

AUTHORS

The current **piixpcib** driver was written by Bruno Ducrot. It was ported to NetBSD by Jared D. McNeill ⟨jmcneill@NetBSD.org⟩.

NAME

piixpm — Intel PIIX and compatible Power Management controller

SYNOPSIS

```
piixpm* at pci? dev ? function ?
iic* at piixpm?
```

DESCRIPTION

The **piixpm** driver provides support for the Intel PIIX and compatible Power Management controller. Only the SMBus host interface is supported and can be used with the iic(4) framework.

Supported chipsets:

- ATI SB200, SB300, SB400, SB600, SB700, SB800
- Intel 82371AB (PIIX4), 82440MX
- Serverworks OSB4, OSB5, OSB6, HT1000SB

SEE ALSO

iic(4), intro(4), pci(4)

HISTORY

The **piixpm** driver first appeared in OpenBSD 3.9 and then in NetBSD 4.0.

AUTHORS

The current **piixpm** driver was written by Alexander Yurchenko ⟨grange@openbsd.org⟩. It was ported to NetBSD by Jared D. McNeill ⟨jmcneill@netbsd.org⟩.

BUGS

The driver doesn't support I2C commands with a data buffer size of more than 2 bytes.

NAME

pim — Protocol Independent Multicast

SYNOPSIS

```
options MROUTING
options PIM

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip_mroute.h>
#include <netinet/pim.h>

int
getsockopt(int s, IPPROTO_IP, MRT_PIM, void *optval, socklen_t *optlen);

int
setsockopt(int s, IPPROTO_IP, MRT_PIM, const void *optval, socklen_t optlen);

int
getsockopt(int s, IPPROTO_IPV6, MRT6_PIM, void *optval, socklen_t *optlen);

int
setsockopt(int s, IPPROTO_IPV6, MRT6_PIM, const void *optval,
     socklen_t optlen);
```

DESCRIPTION

PIM is the common name for two multicast routing protocols: Protocol Independent Multicast - Sparse Mode (PIM-SM) and Protocol Independent Multicast - Dense Mode (PIM-DM).

PIM-SM is a multicast routing protocol that can use the underlying unicast routing information base or a separate multicast-capable routing information base. It builds unidirectional shared trees rooted at a Rendezvous Point (RP) per group, and optionally creates shortest-path trees per source.

PIM-DM is a multicast routing protocol that uses the underlying unicast routing information base to flood multicast datagrams to all multicast routers. Prune messages are used to prevent future datagrams from propagating to routers with no group membership information.

Both PIM-SM and PIM-DM are fairly complex protocols, though PIM-SM is much more complex. To enable PIM-SM or PIM-DM multicast routing in a router, the user must enable multicast routing and PIM processing in the kernel (see **SYNOPSIS** about the kernel configuration options), and must run a PIM-SM or PIM-DM capable user-level process. From developer's point of view, the programming guide described in the **Programming Guide** section should be used to control the PIM processing in the kernel.

Programming Guide

After a multicast routing socket is open and multicast forwarding is enabled in the kernel (see `multicast`(4)), one of the following socket options should be used to enable or disable PIM processing in the kernel. Note that those options require certain privilege (i.e., root privilege):

```
/* IPv4 */
int v = 1;        /* 1 to enable, or 0 to disable */
setsockopt(mrouter_s4, IPPROTO_IP, MRT_PIM, (void *)&v, sizeof(v));

/* IPv6 */
int v = 1;        /* 1 to enable, or 0 to disable */
setsockopt(mrouter_s6, IPPROTO_IPV6, MRT6_PIM, (void *)&v, sizeof(v));
```

After PIM processing is enabled, the multicast-capable interfaces should be added (see multicast(4)). In case of PIM-SM, the PIM-Register virtual interface must be added as well. This can be accomplished by using the following options:

```
/* IPv4 */
struct vifctl vc;
memset(&vc, 0, sizeof(vc));
/* Assign all vifctl fields as appropriate */
...
if (is_pim_register_vif)
    vc.vifc_flags |= VIFF_REGISTER;
setsockopt(mrouter_s4, IPPROTO_IP, MRT_ADD_VIF, (void *)&vc,
           sizeof(vc));
```

```
/* IPv6 */
struct mif6ctl mc;
memset(&mc, 0, sizeof(mc));
/* Assign all mif6ctl fields as appropriate */
...
if (is_pim_register_vif)
    mc.mif6c_flags |= MIFF_REGISTER;
setsockopt(mrouter_s6, IPPROTO_IPV6, MRT6_ADD_MIF, (void *)&mc,
           sizeof(mc));
```

Sending or receiving of PIM packets can be accomplished by opening first a "raw socket" (see socket(2)), with protocol value of IPPROTO_PIM:

```
/* IPv4 */
int pim_s4;
pim_s4 = socket(AF_INET, SOCK_RAW, IPPROTO_PIM);
```

```
/* IPv6 */
int pim_s6;
pim_s6 = socket(AF_INET6, SOCK_RAW, IPPROTO_PIM);
```

Then, the following system calls can be used to send or receive PIM packets: sendto(2), sendmsg(2), recvfrom(2), recvmsg(2).

SEE ALSO

getsockopt(2), recvfrom(2), recvmsg(2), sendmsg(2), sendto(2), setsockopt(2), socket(2), inet(4), intro(4), ip(4), multicast(4)

STANDARDS

The PIM-SM protocol is specified in RFC 2362 (to be replaced by *draft-ietf-pim-sm-v2-new-*). The PIM-DM protocol is specified in *draft-ietf-pim-dm-new-v2-*).

AUTHORS

The original IPv4 PIM kernel support for IRIX and SunOS-4.x was implemented by Ahmed Helmy (USC and SGI). Later the code was ported to various BSD flavors and modified by George Edmond Eddy (Rusty) (ISI), Hitoshi Asaeda (WIDE Project), and Pavlin Radoslavov (USC/ISI and ICSI). The IPv6 PIM kernel support was implemented by the KAME project (http://www.kame.net), and was based on the IPv4 PIM kernel support.

This manual page was written by Pavlin Radoslavov (ICSI).

NAME

plip — printer port Internet Protocol driver

SYNOPSIS

```
plip* at ppbus?
options PLIP_DEBUG
```

DESCRIPTION

The **plip** driver allows a PC parallel printer port to be used as a point-to-point network interface between two similarly configured systems. Data is transferred 4 bits at a time, using the printer status lines for input: hence there is no requirement for special bidirectional hardware and any standard AT-compatible printer port with working interrupts may be used.

During the boot process, for each ppbus(4) device which is attached and has an interrupt capability, a corresponding **plip** device is attached. The **plip** device is configured using ifconfig(8) using the options for a point-to-point network interface:

ifconfig *plip0 hostaddress destaddress* [**-link0|link0**] [up|down] [. . .]

Configuring a **plip** device "up" with ifconfig(8) causes the corresponding ppbus(4) to be reserved for PLIP until the network interface is configured "down".

The communication protocol is selected by the **link0** flag:

-link0 (default) Use FreeBSD mode (LPIP). This is the simpler of the two modes and therefore slightly more efficient.

link0 Use Crynwr/Linux compatible mode (CLPIP). This mode has a simulated ethernet packet header, and is easier to interface to other types of equipment.

The interface MTU defaults to 1500, but may be set to any value. Both ends of the link must be configured with the same MTU. See ifconfig(8) for details on configuring network interfaces.

Cable Connections

The cable connecting the two parallel ports should be wired as follows:

Pin	Pin	Description
2	15	Data0 -> ERROR*
3	13	Data1 -> SLCT
4	12	Data2 -> PE
5	10	Data3 -> ACK*
6	11	Data4 -> BUSY
15	2	ERROR* -> Data0
13	3	SLCT -> Data1
12	4	PE -> Data2
10	5	ACK* -> Data3
11	6	BUSY -> Data4
18-25	18-25	Ground

Cables with this wiring are widely available as "Laplink" cables, and are often colored yellow.

The connections are symmetric, and provide 5 lines in each direction (four data plus one handshake). The two modes use the same wiring, but make a different choice of which line to use as handshake.

FreeBSD LPIP mode

The signal lines are used as follows:

Data0 (Pin 2)	Data out, bit 0.
Data1 (Pin 3)	Data out, bit 1.
Data2 (Pin 4)	Data out, bit 2.
Data3 (Pin 5)	Handshake out.
Data4 (Pin 6)	Data out, bit 3.
ERROR∗ (pin 15)	Data in, bit 0.
SLCT (pin 13)	Data in, bit 1.
PE (pin 12)	Data in, bit 2.
BUSY (pin 11)	Data in, bit 3.
ACK∗ (pin 10)	Handshake in.

When idle, all data lines are at zero. Each byte is signaled in four steps: sender writes the 4 most significant bits and raises the handshake line; receiver reads the 4 bits and raises its handshake to acknowledge; sender places the 4 least significant bits on the data lines and lowers the handshake; receiver reads the data and lowers its handshake.

The packet format has a two-byte header, comprising the fixed values 0x08, 0x00, immediately followed by the IP header and data.

The start of a packet is indicated by simply signaling the first byte of the header. The end of the packet is indicated by inverting the data lines (i.e. writing the ones-complement of the previous nibble to be transmitted) without changing the state of the handshake.

Note that the end-of-packet marker assumes that the handshake signal and the data-out bits can be written in a single instruction - otherwise certain byte values in the packet data would falsely be interpreted as end-of-packet. This is not a problem for the PC printer port, but requires care when implementing this protocol on other equipment.

Crynwr/Linux CLPIP mode

The signal lines are used as follows:

Data0 (Pin 2)	Data out, bit 0.
Data1 (Pin 3)	Data out, bit 1.
Data2 (Pin 4)	Data out, bit 2.
Data3 (Pin 5)	Data out, bit 3.
Data4 (Pin 6)	Handshake out.
ERROR∗ (pin 15)	Data in, bit 0.
SLCT (pin 13)	Data in, bit 1.
PE (pin 12)	Data in, bit 2.
ACK∗ (pin 10)	Data in, bit 3.
BUSY (pin 11)	Handshake in.

When idle, all data lines are at zero. Each byte is signaled in four steps: sender writes the 4 least significant bits and raises the handshake line; receiver reads the 4 bits and raises its handshake to acknowledge; sender places the 4 most significant bits on the data lines and lowers the handshake; receiver reads the data and lowers its handshake. [Note that this is the opposite nibble order to LPIP mode].

Packet format is:

```
Length (least significant byte)
Length (most significant byte)
12 bytes of supposed MAC addresses (ignored by FreeBSD).
Fixed byte 0x08
Fixed byte 0x00
<IP datagram>
Checksum byte.
```

The length includes the 14 header bytes, but not the length bytes themselves nor the checksum byte.

The checksum is a simple arithmetic sum of all the bytes (again, including the header but not checksum or length bytes). FreeBSD calculates outgoing checksums, but does not validate incoming ones.

The start of packet has to be signaled specially, since the line chosen for handshake-in cannot be used to generate an interrupt. The sender writes the value 0x08 to the data lines, and waits for the receiver to respond by writing 0x01 to its data lines. The sender then starts signaling the first byte of the packet (the length byte).

End of packet is deduced from the packet length and is not signaled specially (although the data lines are restored to the zero, idle state to avoid spuriously indicating the start of the next packet).

SEE ALSO

`atppc`(4), `ppbus`(4), `ifconfig`(8)

HISTORY

The **`plip`** driver was implemented for `ppbus`(4) in FreeBSD and imported into NetBSD. Crynwr packet drivers implemented PLIP for MS-DOS. Linux also has a PLIP driver. The protocols are know as LPIP (FreeBSD) and CLPIP (Crynwr/Linux) in the documentation and code of this port. LPIP originally appeared in FreeBSD.

AUTHORS

This manual page is based on the FreeBSD **`lp`** manual page. The information has been updated for the NetBSD port by Gary Thorpe.

BUGS

Busy-waiting loops are used while handshaking bytes (and worse still when waiting for the receiving system to respond to an interrupt for the start of a packet). Hence a fast system talking to a slow one will consume excessive amounts of CPU. This is unavoidable in the case of CLPIP mode due to the choice of handshake lines; it could theoretically be improved in the case of LPIP mode.

Regardless of the speed difference between hosts, PLIP is CPU-intensive and its made worse by having to send nibbles (4 bits) at a time.

Polling timeouts are controlled by counting loop iterations rather than timers, and so are dependent on CPU speed. This is somewhat stabilized by the need to perform (slow) ISA bus cycles to actually read the port.

In the FreeBSD implementation, the idle state was not properly being restored on errors or when finishing transmitting/receiving. This implementation attempts to fix this problem which would result in an unresponsive interface that could no longer be used (the port bits get stuck in a state and nothing can progress) by zeroing the data register when necessary.

For unknown reasons, the more complex protocol (CLPIP) yields higher data transfer rates during testing so far. This could possibly be because the other side can reliably detect when the host is transmitting in this implementation of CLPIP (this may not necessarily be true in Linux or MS-DOS packet drivers). CLPIP gets about 70 KB/sec (the best expected is about 75 KB/sec) and LPIP get about 55 KB/sec. This is despite LPIP being able to send more packets over the interface (tested with "**ping -f**") compared to CLPIP.

NAME

pm — DECstation 2100/3100 baseboard framebuffer

SYNOPSIS

```
pm* at ibus? addr ?
wsdisplay* at pm?
```

DESCRIPTION

The **pm** driver provides support for the 2100/3100 baseboard framebuffer. It can operate as either a monochrome framebuffer (with memory part number VFB01) or an 8 bpp colour framebuffer (with memory part number VFB02). Both provide 16x16 hardware sprite cursor.

SEE ALSO

cfb(4), mfb(4), px(4), pxg(4), sfb(4), tc(4), tfb(4), wscons(4), xcfb(4)

NAME

pms — PS/2 auxiliary port mouse driver

SYNOPSIS

```
pckbc* at isa?
pms* at pckbc?
wsmouse* at pms?

options PMS_DISABLE_POWERHOOK
options PMS_SYNAPTICS_TOUCHPAD
```

DESCRIPTION

The **pms** driver provides an interface to PS/2 auxiliary port mice within the wscons(4) framework. Parent device in terms of the autoconfiguration framework is pckbc(4), the PC keyboard controller. "pms" is a generic driver which supports mice using common variants of the PS/2 protocol, including wheel mice of the "IntelliMouse" breed. Wheel movements are mapped to a third (z-) axis. The driver is believed to work with both 3-button and 5-button mice with scroll wheels. Mice which use other protocol extensions are not currently supported, but might be if protocol documentation could be found. Mouse related data are accessed by wsmouse(4) devices.

The **pms** driver has been updated to attempt to renegotiate mouse protocol after seeing suspicious or defective mouse protocol packets, or unusual delays in the middle of a packet; this should improve the chances that a mouse will recover after being switched away or reset (for instance, by a console switch).

The *PMS_DISABLE_POWERHOOK* kernel option disables PS/2 reset on resume.

In addition, the **pms** driver supports the "Synaptics" touchpad in native mode, enabled with the *PMS_SYNAPTICS_TOUCHPAD* kernel option. This allows the driver to take advantage of extra features available on Synaptics Touchpads. The following sysctl(8) variables control the touchpad's behavior:

`hw.synaptics.up_down_emulation`
: If the touchpad reports the existence of Up/Down buttons, this value determines if they should be reported as button 4 and 5 events or if they should be used to emulate some other event. When set to 0, report Up/Down events as buttons 4 and 5. When set to 1, the Up and Down buttons are both mapped to the middle button. When set to 2 (default), the Up and Down buttons are used for Z-axis emulation, which more closely resembles how mouse wheels operate.

`hw.synaptics.up_down_motion_delta`
: When the Up/Down buttons are used for Z-axis emulation, this value specifies the emulated delta-Z value per click.

`hw.synaptics.gesture_move`
: Gestures will not be recognised if the finger moves by more than this amount between taps.

`hw.synaptics.gesture_length`
: Gestures will not be recognised if the number of packets (at 80 packets per second) between taps exceeds this value.

`hw.synaptics.edge_left`

`hw.synaptics.edge_right`

`hw.synaptics.edge_top`

`hw.synaptics.edge_bottom`
: These values define a border around the touchpad which will be used for edge motion emulation during a drag gesture. If a drag gesture is in progress and the finger moves into this bor-

der, the driver will behave as if the finger continues to move in the same direction beyond the edge of the touchpad.

`hw.synaptics.edge_motion_delta`
This specifies the pointer speed when edge motion is in effect.

`hw.synaptics.finger_high`
The driver will ignore new finger events until the reported pressure exceeds this value.

`hw.synaptics.finger_low`
The driver will assume a finger remains on the touchpad until the reported pressure drops below this value.

`hw.synaptics.two_fingers_emulation`
More recent touchpads can report the presence of more than one finger on the pad. This value determines how such events are used. If set to 0 (default), two-finger events are ignored. If set to 1, two-finger events generate a right button click. If set to 2, two-finger events generate a middle button click.

`hw.synaptics.scale_x`

`hw.synaptics.scale_y`
Scale factor used to divide movement deltas derived from Synaptics coordinates (0-6143) to yield more reasonable values (default 16).

`hw.synaptics.max_speed_x`

`hw.synaptics.max_speed_y`
Limits pointer rate of change (after scaling) per reported movement event (default 32).

`hw.synaptics.movement_threshold`
Movements of less than this value (in Synaptics coordinates) are ignored (default 4).

SEE ALSO

`pckbc`(4), `ums`(4), `wsmouse`(4)

AUTHORS

The **`pms`** driver was originally written by Christopher G. Demetriou. The changes to merge the "IntelliMouse" protocol in, and reset the mouse in the event of protocol problems, were contributed by Peter Seebach. Special thanks to Ray Trent, at Synaptics, who contributed valuable insight into how to identify bogus mouse data. The changes to add "Synaptics" pad support were by Ales Krenek, Kentaro A. Kurahone, and Steve C. Woodford.

BUGS

It is possible for the driver to mistakenly negotiate the non-scroll-wheel protocol, after which it is unlikely to recover until the device is closed and reopened.

NAME

pmu — support for Power Management Units found in all Apple laptops and some desktop Power Macintosh computers

SYNOPSIS

```
pmu* at obio?
nadb* at pmu?
battery* at pmu?
smartbat* at pmu?
```

DESCRIPTION

The **pmu** driver provides support for the Power Management Unit found in Apple laptops and some desktop Power Macintosh computers. Functions controlled by the PMU include the real time clock, ADB, power, batteries, on some laptops like the PowerBook 3400c and similar machines it also controls hotkeys and display brightness, on others it provides an `iic`(9) bus and on some it controls CPU speed. On many older machines it also provides access to some non-volatile memory and thermal sensors. Not all those features are present on all machines, for instance Power Macintosh G4 and later machines don't have ADB, many more recent laptops have display brightness and backlight control built into the graphics controller instead of the PMU, only a few older PowerBooks use the PMU for CPU speed control and newer machines use a different way to access non-volatile memory. However, all known PMUs so far provide a real time clock and power control.

Notes by model

Real time clock and power control are present and supported on all machines that can run NetBSD/macppc, ADB is supported when present.

`PowerBook 2400, 3400c, and 3500`
: Battery status and thermal sensors found on the mainboard and in the battery pack are supported by the `battery`(4) driver, values can be read via `envsys`(4). Hotkeys for brightness control are supported, CPU speed control and parameter RAM are present but unsupported.

`Power Macintosh G4`
: ADB is not present, `iic`(9) is present but unsupported.

SEE ALSO

`battery`(4), `cuda`(4), `nadb`(4), `nvram`(4), `obio`(4), `iic`(9)

BUGS

Some features are currently unsupported, like the `iic`(9) bus, access to parameter RAM and CPU speed control.

NAME

pnaphy — Driver for generic HomePNA PHYs

SYNOPSIS

pnaphy* **at mii? phy ?**

DESCRIPTION

The **pnaphy** is a generic driver for HomePNA “home networking” PHYs which provide Ethernet-like connectivity over standard home telephone lines without interrupting POTS service.

HomePNA 1.0 runs at a speed of 1Mb/s.

The **pnaphy** driver currently supports the following devices:

- AMD Am79c901 HomePNA 1.0 PHY

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

pnozz — Weitek Power9100 accelerated frame buffer

SYNOPSIS

```
pnozz0 at sbus? slot ? offset ?
```

DESCRIPTION

The **pnozz** is a color frame buffer with graphics acceleration, embedded in the Tadpole SPARCbook 3GS, 3GX, 3TX, and 3XP laptops. It is based on the Weitek Power9100 video processor and an IBM RGB525 RAMDAC.

If the `tctrl`(4) device is also configured, the **pnozz** will be powered down when the lid of the laptop is closed or the screen is blanked.

SEE ALSO

`intro`(4), `sbus`(4), `tctrl`(4)

HISTORY

Support for the **pnozz** first appeared in NetBSD 1.6.

BUGS

There is currently no way to switch back and forth from the onboard display to the external connector. It is not possible to change resolutions or color depth.

NAME

pnpbios — introduction to PnP BIOS support

SYNOPSIS

```
pnpbios0 at mainbus?
XX*      at pnpbios0 index ?

options PNPBIOSVERBOSE
options PNPBIOSDEBUG
```

INTRODUCTION

pnpbios provides support for finding and attaching devices by extracting information from the PnP BIOS of a machine.

SUPPORTED DEVICES

NetBSD includes the following **pnpbios** drivers, sorted by driver name:

com	serial communications interface
ess	ESS Technology AudioDrive family audio device driver
fdc	floppy controller
joy	game controller
lpt	parallel port driver
npx	math coprocessor
pciide	PCI IDE controllers driver
pckbc	PC keyboard/mouse controller
sb	SoundBlaster family audio device driver
wss	Windows Sound System hardware driver
ym	Yamaha OPL3-SA3 sound driver

SEE ALSO

com(4), ess(4), fdc(4), intro(4), isa(4), joy(4), lpt(4), npx(4), pci(4), pciide(4), pckbc(4), sb(4), wss(4), ym(4)

HISTORY

The **pnpbios** driver appeared in NetBSD 1.5.

BUGS

The **pnpbios** bus provides a different way to attach devices than the real buses (e.g., isa(4)). The reason to use **pnpbios** instead of the real bus is that some machines have weird routings of, e.g., interrupts. The exact information about these things can be found in the BIOS, which is why the **pnpbios** bus has a better chance of succeeding with getting it right.

Many older BIOS implementations do not support PnP, and some newer ones do not follow the standards.

NAME

podulebus — Acorn Expansion Card bus driver

SYNOPSIS

```
podulebus0 at ioc0 bank 4
```
(NetBSD/acorn26)
```
podulebus0 at root
```
(NetBSD/acorn32)

DESCRIPTION

The **podulebus** driver handles the expansion-card interface in Archimedes machines and their successors. This includes conventional expansion cards, mini expansion cards (as introduced in the A3000), network expansion cards (as introduced in the A3020), DEBI expansion cards (as introduced in the Risc PC), and Mk II network cards (also introduced in the Risc PC). Drivers for individual cards attach as children of the **podulebus** device.

NetBSD includes several machine-independent expansion card device drivers. There are also some device drivers which are specific to NetBSD/acorn26 or NetBSD/acorn32.

HARDWARE

The following devices are supported by NetBSD.

SCSI interfaces

asc Acorn AKA30, AKA31, and AKA32 SCSI expansion cards (NetBSD/acorn32).

cosc MCS Connect32 SCSI interface (NetBSD/acorn32).

csa Cumana 8-bit SCSI interface (NetBSD/acorn32).

csc Cumana 16-bit SCSI interface (NetBSD/acorn32).

hcsc HCCS 8-bit SCSI interface.

oak Oak SCSI interface.

ptsc Powertec SCSI interface (NetBSD/acorn32).

sec Acorn AKA30, AKA31, and AKA32 SCSI expansion cards.

Disk controllers

dtide D.T. Software IDE controller.

hcide HCCS IDE controller.

icside ICS IDE controller (NetBSD/acorn32).

rapide Yellowstone Educational Solutions RapIDE IDE controller (NetBSD/acorn32).

simide Simtec IDE controller (NetBSD/acorn32).

Network interfaces

ea Atomwide A-10*xx* and Acorn AEH54 Ethernet cards (Ether3).

eb Atomwide and ANT network-slot and Acorn AEH61 Ethernet cards (EtherB).

eh i-cubed EtherLan 100-, 200- and 500-series, and Acorn AEH75, AEH77, and AEH79 Ethernet cards (EtherH) (NetBSD/acorn26).

ei Acorn AKA25 Ethernet card (Ether1).

ie Acorn AKA25 Ethernet card (Ether1) (NetBSD/acorn32).

ne Various vaguely NE2000-compatible Ethernet cards (NetBSD/acorn32).

Serial interfaces

amps Atomwide multi-port serial interface (NetBSD/acorn32).

SEE ALSO

acorn26/eh(4), acorn32/asc(4), acorn32/cosc(4), acorn32/csc(4), acorn32/ie(4), acorn32/ptsc(4), dtide(4), ea(4), eb(4), ei(4), hcide(4), ne(4), oak(4), sec(4)

BUGS

Too few drivers are shared between NetBSD/acorn26 and NetBSD/acorn32.

NAME

pow — X68k power switch device

DESCRIPTION

Files /dev/pow and /dev/pow1 are devices used to access the x68k power switch / bootstrap information and the RTC alarm timer. The following operations are allowed using ioctl(2) system call:

POWIOGPOWERINFO
: Getting power switch status

POWIOGALARMINFO
: Getting RTC alarm timer status

POWIOSALARMINFO
: Setting RTC alarm timer

POWIOSSIGNAL
: Setting signal number which is sent at changing the power status

GETTING POWER SWITCH STATUS

```
<sys/ioctl.h>
<machine/powioctl.h>

ioctl (fd, POWIOCGPOWERINFO, &powerinfo);
struct x68k_powerinfo powerinfo;
```

Returns the power switch status etc. in the following structure.

```
struct x68k_powerinfo {
        int pow_switch_boottime;
        int pow_switch_current;
        time_t pow_boottime;
        unsigned int pow_bootcount;
        time_t pow_usedtotal;
};
```

Each member means:

pow_switch_boottime: The power switch status at the system start time. The status is known by applying logical and (&) with:

POW_ALARMSW
: True if the system started by the RTC alarm timer

POW_EXTERNALSW
: True if the EXPWON signal of the I/O slot is on

POW_FRONTSW
: True if the front power switch is on

EXPWON-related information is not tested.

pow_switch_current: Current power switch status. See above.

pow_boottime: The time when NetBSD started.

pow_bootcount, pow_usedtotal: Host's boot information from the system memory switch. Total usage in count (pow_bootcount) and second (pow_usedtotal).

ACCESSING RTC ALARM TIMER

```
<sys/ioctl.h>
<machine/powioctl.h>

ioctl (fd, POWIOCGALARMINFO, &alarminfo);
struct x68k_alarminfo alarminfo;
```

Returns x68k's alarm timer information in the following structure.

```
struct x68k_alarminfo {
        int al_enable;
        unsigned int al_ontime;
        int al_dowhat;
        time_t al_offtime;
};
```

Each of the members means:

al_enable: True if the alarm timer is enable.

al_ontime: Internal expression of the alarm timer. See 'C Compiler PRO-68K Programmers' Manual' Chapter 3 Section ALARMSET.

al_dowhat: What to do when started by the alarm timer. See 'Programmers' manual'.

al_offtime: Display the time in seconds since the alarm is activated until the power is turned off (though the resolution is resolved to minutes). Be cautious when using IOCS and the like as these require minute time units. 0 indicates an infinity.

ioctl (fd, POWIOCSALARMINFO, &alarminfo);
struct x68k_alarminfo alarminfo;

Sets the alarm timer according to the argument *alarminfo* (see above).

SIGNAL

```
<sys/ioctl.h>
<machine/powioctl.h>
```

ioctl (fd, POWIOCSSIGNAL, &signum);
int signum;

Signal *signum* is sent when the state of the power switch (front or external) changes. It becomes invalid when close(2) is called. Note that this function is not provided with /dev/pow1.

SEE ALSO

ioctl(2), poffd(8), rtcalarm(8)

'C Compiler PRO-68K Programmers' Manual', Chapter 3 IOCS Calls, Section ALARMSET, Sharp

AUTHORS

Minoura Makoto ⟨minoura@flab.fujitsu.co.jp⟩.

SPECIAL THANKS

Liam Hahne Minn <hahne@sail.t.u-tokyo.ac.jp>.

BUGS

The file /dev/pow can only be opened by one process at a time.

NAME

ppbus — Parallel Port Bus system

SYNOPSIS

```
ppbus* at atppc?
options PPBUS_VERBOSE
options PPBUS_DEBUG
options DEBUG_1284

gpio* at ppbus?
lpt* at ppbus?
plip* at ppbus?
pps* at ppbus?
```

DESCRIPTION

The **ppbus** system provides a uniform, modular, and architecture-independent system for the implementation of drivers to control various parallel devices, and to use different parallel port chip sets.

DEVICE DRIVERS

In order to write new drivers or port existing drivers, the **ppbus** system provides the following facilities:

- architecture-independent macros or functions to access parallel ports
- mechanism to allow various devices to share the same parallel port
- a gpio(4) interface to access the individual pins
- a user interface named ppi(4) that allows parallel port access from outside the kernel without conflicting with kernel-in drivers.

Developing new drivers

The **ppbus** system has been designed to support the development of standard and non-standard software:

Driver	*Description* It uses standard and non-standard parallel port accesses.
ppi	Parallel port interface for general I/O
pps	Pulse per second Timing Interface

Porting existing drivers

Another approach to the **ppbus** system is to port existing drivers. Various drivers have already been ported:

Driver	*Description*
lpt	lpt printer driver
lp	plip network interface driver

ppbus should let you port any other software even from other operating systems that provide similar services.

PARALLEL PORT CHIP SETS

Parallel port chip set support is provided by atppc(4).

The **ppbus** system provides functions and macros to request service from the **ppbus** including reads, writes, setting of parameters, and bus requests/releases.

atppc(4) detects chip set and capabilities and sets up interrupt handling. It makes methods available for use to the **ppbus** system.

PARALLEL PORT MODEL

The logical parallel port model chosen for the **ppbus** system is the AT parallel port model. Consequently, for the *atppc* implementation of **ppbus**, most of the services provided by **ppbus** will translate into I/O instructions on actual registers. However, other parallel port implementations may require more than one I/O instruction to do a single logical register operation on data, status and control virtual registers.

Description

The parallel port may operate in the following modes:

- Compatible (Centronics -- the standard parallel port mode) mode, output byte
- Nibble mode, input 4-bits
- Byte (PS/2) mode, input byte
- Extended Capability Port (ECP) mode, bidirectional byte
- Enhanced Parallel Port (EPP) mode, bidirectional byte

Compatible mode

This mode defines the protocol used by most PCs to transfer data to a printer. In this mode, data is placed on the port's data lines, the printer status is checked for no errors and that it is not busy, and then a data Strobe is generated by the software to clock the data to the printer.

Many I/O controllers have implemented a mode that uses a FIFO buffer to transfer data with the Compatibility mode protocol. This mode is referred to as "Fast Centronics" or "Parallel Port FIFO mode".

Nibble mode

The Nibble mode is the most common way to get reverse channel data from a printer or peripheral. When combined with the standard host to printer mode, a bidirectional data channel is created. Inputs are accomplished by reading 4 of the 8 bits of the status register.

Byte mode

In this mode, the data register is used either for outputs and inputs. All transfers are 8-bits long. Channel direction must be negotiated when doing IEEE 1248 compliant operations.

Extended Capability Port mode

The ECP protocol was proposed as an advanced mode for communication with printer and scanner type peripherals. Like the EPP protocol, ECP mode provides for a high performance bidirectional communication path between the host adapter and the peripheral.

ECP protocol features include:

Run_Length_Encoding (RLE) data compression for host adapters (not supported in these drivers)

FIFO's for both the forward and reverse channels

DMA or programmed I/O for the host register interface.

Enhanced Parallel Port mode

The EPP protocol was originally developed as a means to provide a high performance parallel port link that would still be compatible with the standard parallel port.

The EPP mode has two types of cycle: address and data. What makes the difference at hardware level is the strobe of the byte placed on the data lines. Data are strobed with nAutofeed, addresses are strobed with nSelectin signals.

A particularity of the ISA implementation of the EPP protocol is that an EPP cycle fits in an ISA cycle. In this fashion, parallel port peripherals can operate at close to the same performance levels as an equivalent ISA plug-in card.

At software level, you may implement the protocol you wish, using data and address cycles as you want. This is for the IEEE 1284 compatible part. Peripheral vendors may implement protocol handshake with the following status lines: PError, nFault and Select. Try to know how these lines toggle with your peripheral, allowing the peripheral to request more data, stop the transfer and so on.

At any time, the peripheral may interrupt the host with the nAck signal without disturbing the current transfer.

Mixed modes

Some manufacturers, like SMC, have implemented chip sets that support mixed modes. With such chip sets, mode switching is available at any time by accessing the extended control register. All ECP-capable chip sets can switch between standard, byte, fast centronics, and ECP modes. Some ECP chip sets also support switching to EPP mode.

IEEE 1284 1994 Standard

Background

This standard is also named "IEEE Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers". It defines a signaling method for asynchronous, fully interlocked, bidirectional parallel communications between hosts and printers or other peripherals. It also specifies a format for a peripheral identification string and a method of returning this string to the host.

This standard is architecture independent and only specifies dialog handshake at signal level. One should refer to architecture specific documentation in order to manipulate machine dependent registers, mapped memory or other methods to control these signals.

The IEEE 1284 protocol is fully oriented with all supported parallel port modes. The computer acts as master and the peripheral as slave.

Any transfer is defined as a finite state automate. It allows software to properly manage the fully interlocked scheme of the signaling method. The compatible mode is supported "as is" without any negotiation because it is the default, backward-compatible transfer mode. Any other mode must be firstly negotiated by the host to check it is supported by the peripheral, then to enter one of the forward idle states.

At any time, the slave may want to send data to the host. The host must negotiate to permit the peripheral to complete the transfer. Interrupt lines may be dedicated to the requesting signals to prevent time consuming polling methods.

If the host accepts the transfer, it must firstly negotiate the reverse mode and then start the transfer. At any time during reverse transfer, the host may terminate the transfer or the slave may drive wires to signal that no more data is available.

Implementation

IEEE 1284 Standard support has been implemented at the top of the **ppbus** system as a set of procedures that perform high level functions like negotiation, termination, transfer in any mode without bothering you with low level characteristics of the standard.

IEEE 1284 interacts with the **ppbus** system as little as possible. That means you still have to request the **ppbus** when you want to access it, and of course, release it when finished.

ARCHITECTURE

Chip set, ppbus and device layers

First, there is the *chip set* layer, the lowest of the **ppbus** system. It provides chip set abstraction through a set of low level functions that maps the logical model to the underlying hardware.

Secondly, there is the *ppbus* layer that provides functions to:

1. share the parallel port bus among the daisy-chain like connected devices
2. manage devices linked to **ppbus**
3. propose an arch-independent interface to access the hardware layer.

Finally, the *device* layer represents the traditional device drivers such as `lpt`(4) which now use an abstraction instead of real hardware.

Parallel port mode management

Operating modes are differentiated at various **ppbus** system layers. There is a difference between a *capability* and a *mode*. A chip set may have a combination of capabilities, but at any one time the **ppbus** system operates in a single mode.

Nibble mode is a *virtual* mode: the actual chip set would be in standard mode and the driver would change its behavior to drive the right lines on the parallel port.

Each child device of **ppbus** must set its operating mode and other parameters whenever it requests and gets access to its parent **ppbus**.

FEATURES

The boot process

ppbus attachment tries to detect any PnP parallel peripheral (according to *Plug and Play Parallel Port Devices draft from (c)1993-4* Microsoft Corporation) then probes and attaches known device drivers.

During probe, device drivers should request the **ppbus** and try to determine if the right capabilities are present in the system.

Bus request and interrupts

ppbus reservation via a bus request is mandatory not to corrupt I/O of other devices. For example, when the `lpt`(4) device is opened, the bus will be "allocated" to the device driver and attempts to reserve the bus for another device will fail until the `lpt`(4) driver releases the bus.

Child devices can also register interrupt handlers to be called when a hardware interrupt occurs. In order to attach a handler, drivers must own the bus. Drivers should have interrupt handlers that check to see if the device still owns the bus when they are called and/or ensure that these handlers are removed whenever the device does not own the bus.

Micro-sequences

Micro-sequences are a general purpose mechanism to allow fast low-level manipulation of the parallel port. Micro-sequences may be used to do either standard (in IEEE 1284 modes) or non-standard transfers. The philosophy of micro-sequences is to avoid the overhead of the **ppbus** layer for a sequence of operations and do most of the job at the chip set level.

A micro-sequence is an array of opcodes and parameters. Each opcode codes an operation (opcodes are described in `microseq`(9)). Standard I/O operations are implemented at ppbus level whereas basic I/O operations and microseq language are coded at adapter level for efficiency.

GPIO interface

Pins 1 through 17 of the parallel port can be controlled through the `gpio`(4) interface, pins 18 through 25 are hardwired to ground. Pins 10 through 13 and pin 15 are input pins, the others are output pins. Some of the pins are inverted by the hardware, the values read or written are adjusted accordingly. Note that the `gpio`(4) interface starts at 0 when numbering pins.

SEE ALSO

`atppc`(4), `gpio`(4,) `lpt`(4), `plip`(4), `ppi`(4), `microseq`(9)

HISTORY

The **`ppbus`** system first appeared in FreeBSD 3.0.

AUTHORS

This manual page is based on the FreeBSD **`ppbus`** manual page. The information has been updated for the NetBSD port by Gary Thorpe.

BUGS

The **`ppbus`** framework is still experimental and not enabled by default yet.

NAME

ppi — HP-IB printer/plotter interface

SYNOPSIS

```
ppi0 at hpibbus0 slave 5
```

DESCRIPTION

The **ppi** interface provides a means of communication with HP-IB printers and plotters.

Special files `ppi0` through `ppi7` are used to access the devices, with the digit at the end of the filename referring to the bus address of the device. Current versions of the autoconf code can not probe for these devices, so the device entry in the configuration file must be fully qualified.

The device files appear as follows:

```
"crw-rw-rw-  1 root      11,   0 Dec 21 11:22 /dev/ppi"
```

DIAGNOSTICS

None.

SEE ALSO

`hpib`(4)

BUGS

This driver is very primitive, it handshakes data out byte by byte. It should use DMA if possible.

NAME

ppi — user-space interface to ppbus parallel port

SYNOPSIS

ppi* **at ppbus?**

Minor numbering: Unit numbers correspond directly to ppbus(4) numbers.

DESCRIPTION

The **ppi** driver provides a convenient means for user applications to manipulate the state of the parallel port, enabling easy low-speed I/O operations without the security problems inherent with the use of the /dev/io interface.

The programming interface is described in ppi(9).

SEE ALSO

atppc(4), io(4), ppbus(4), ppi(9)

HISTORY

ppi originally appeared in FreeBSD.

AUTHORS

This manual page is based on the FreeBSD **ppi** manual page and was updated for the NetBSD port by Gary Thorpe.

BUGS

The inverse sense of signals is confusing.

The **ioctl**() interface is slow, and there is no way (yet) to chain multiple operations together.

The headers required for user applications are not installed as part of the standard system.

NAME

ppp — point to point protocol network interface

SYNOPSIS

```
options PPP_BSDCOMP
options PPP_DEFLATE
options PPP_FILTER

pseudo-device ppp
```

DESCRIPTION

The **ppp** interface allows serial lines to be used as network interfaces using the *Point-to-Point Protocol* (PPP). The **ppp** interface can use various types of compression and has many features over the SLIP protocol used by the sl(4) interface.

Supported options are:

options PPP_BSDCOMP
: Enable support for BSD-compress ('bsdcomp') compression in ppp.

options PPP_DEFLATE
: Enable support for deflate compression in ppp.

options PPP_FILTER
: This option turns on pcap(3) based filtering for ppp connections. This option is used by pppd(8) which needs to be compiled with *PPP_FILTER* defined (the current default).

DIAGNOSTICS

ppp%d: af%d not supported . The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

SEE ALSO

inet(4), intro(4), sl(4), pppd(8), pppstats(8)

The Point-to-Point Protocol (PPP), RFC 1661, July 1994.

HISTORY

The **ppp** device appeared in NetBSD 1.0.

BUGS

Currently, only the ip(4) and ip6(4) protocols are supported.

NAME

pppoe — PPP over Ethernet protocol network interface

SYNOPSIS

```
pseudo-device pppoe
```

DESCRIPTION

The **pppoe** interface encapsulates *Point-to-Point Protocol (PPP)* packets inside Ethernet frames as defined by RFC2516.

This is often used to connect a router via a DSL modem to an access concentrator. The **pppoe** interface does not by itself transmit or receive frames, but needs an Ethernet interface to do so. This Ethernet interface is connected to the **pppoe** interface via pppoectl(8). The Ethernet interface needs to be marked UP, but does not need to have an IP address.

There are two basic modes of operation, controlled via the *link1* switch. The default mode, *link1* not being set, tries to keep the configured session open all the time. If the session is disconnected, a new connection attempt is started immediately. The "dial on demand" mode, selected by setting *link1*, only establishes a connection when data is being sent to the interface.

If the kernel is compiled with options PPPOE_SERVER, there are two modes of connection, controlled via the *link0* switch. The default mode, *link0* not being set, is client mode. The "PPPoE server" mode, selected by setting *link0*, is to wait for incoming PPPoE session.

Before a **pppoe** interface is usable, it needs to be configured. The following steps are necessary:

- Create the interface.
- Connect an Ethernet interface. This interface is used for the physical communication. As noted above it must be marked UP, but need not have an IP address.
- Configure authentication. The PPP session needs to identify the client to the peer. For more details on the available options see pppoectl(8).

This all is typically accomplished using an /etc/ifconfig.pppoe0 file.

MSS/MTU problems

If you are using a **pppoe** interface, you will have an unusually low MTU for todays Internet. Combined with a lot of misconfigured sites (host using path MTU discovery behind a router blocking all ICMP traffic) this will often cause problems. Connections to these servers will only work if your system advertises the right MSS in the TCP three way handshake. To get the right MSS, you need to set

```
# Obey interface MTUs when calculating MSS
net.inet.tcp.mss_ifmtu=1
```

in your /etc/sysctl.conf file. This causes the calculated MSS to be based on the MTU of the interface via which the packet is sent. This is always the right value if you are sure the answer to this packet will be received on the same interface (i.e., you only have one interface connected to the Internet.)

Unfortunately this sysctl does not fix the MSS advertised by hosts in the network behind a **pppoe** connected router. To fix this you need *MSS-clamping*, explained below.

Setting up NAT with MSS-clamping

Some systems behind misconfigured firewalls try to use Path-MTU-Discovery, while their firewall blocks all ICMP messages. This is an illegal, but not uncommon, setup. Typically you will have no chance to fix this (remote, outside of your control) setup. And sometimes you will have to use such remote systems (to download data from them, or to do your online banking).

Without special care systems as described above will not be able to send larger chunks of data to a system connected via **pppoe**. But there is a workaround (some may call it cheating): pretend to not be able to handle large packets, by sending a small MSS (maximum segment size) option during initial TCP handshake.

For connections originating from your **pppoe** connected machines, this is accomplished by setting the sysctl variable `net.inet.tcp.mss_ifmtu` to 1 (see above). For connections originating from systems behind your **pppoe** router, you need to set the `mssclamp` options in your NAT rules, like in this example of `/etc/ipnat.conf`:

```
map pppoe0 192.168.1.0/24 -> 0/32 portmap tcp/udp 44000:49999 mssclamp 1440
map pppoe0 192.168.1.0/24 -> 0/32 mssclamp 1440
```

If you do not use NAT, you need to set up a 1:1 NAT rule, just to get the clamping:

```
map pppoe0 x.x.x.x/24 -> 0/0 mssclamp 1440
```

The above examples assume a MTU of 1492 bytes. If the MTU on your PPPoE connection is smaller use the MTU − 52 bytes for clamping e.g. 1408 bytes for a MTU of 1460 bytes. *Note*: The theoretically correct value for the above example would be 1452 bytes (it accounts for the smaller PPPoE MTU, the TCP header and the maximum of 0x40 bytes of TCP options) but it seems to not be sufficient in some cases. Experiments conducted by various people have shown that clamping to the MSS values suggested above works best.

EXAMPLES

A typical `/etc/ifconfig.pppoe0` file looks like this:

```
create
! /sbin/ifconfig ne0 up
! /sbin/pppoectl -e ne0 $int
! /sbin/pppoectl $int myauthproto=pap myauthname=testcaller myauthsecret=donttel
inet 0.0.0.0 0.0.0.1 netmask 0xffffffff
#! /sbin/route add default -iface 0.0.0.1
up
```

The commented out call to `route`(8) may be omitted and the route added in the ip-up script called by `ifwatchd`(8) when the real IP address is known. This is easy in the “connect always” mode (link1 not set), but hard to accomplish in the “dial on demand” mode (link1 set). In the latter case adding an iface route is an easy workaround.

The **pppoe** interfaces operate completely inside the kernel, without any userland support. Because of this, a special daemon is used to fire ip-up or down scripts to execute arbitrary code when the PPP session is established and addresses of the interface become available. To enable the usage of `/etc/ppp/ip-up` and `/etc/ppp/ip-down` for this purpose, simply add

```
ifwatchd=YES
```

to `/etc/rc.conf`. See `ifwatchd`(8) for details and parameters passed to these scripts.

Since this is a PPP interface, the addresses assigned to the interface may change during PPP negotiation. There is no fine grained control available for deciding which addresses are acceptable and which are not. For the local side and the remote address there is exactly one choice: hard coded address or wildcard. If a real address is assigned to one side of the connection, PPP negotiation will only agree to exactly this address. If one side is wildcarded, every address suggested by the peer will be accepted.

To wildcard the local address set it to 0.0.0.0, to wildcard the remote address set it to 0.0.0.1. Wildcarding is not available (nor necessary) for IPv6 operation.

OPTIONS

A **pppoe** enabled kernel will not interfere with other **PPPoE** implementations running on the same machine. Under special circumstances (details below) this is not desirable, so the **pppoe** driver can be told to kill all unknown **PPPoE** sessions received by the Ethernet interface used for a configured **pppoe** interface. To do this, add the following to your kernel config file:

```
options PPPOE_TERM_UNKNOWN_SESSIONS
```

Note that this will break all userland **PPPoE** implementations using the same Ethernet interface!

This option is only useful if you have a static IP address assigned and your ISP does not use LCP echo requests to monitor the link status. After a crash or power failure the peer device still tries to send data to the no longer active session on your computer, and might refuse to reestablish a new connection, because there already is an open session. On receipt of such packets, the **pppoe** driver with this option set will send a PADT packet (request to terminate the session). The peer will immediately disconnect the orphaned session and allow a new one to be established.

To enable **pppoe** server support in the kernel, use

```
options PPPOE_SERVER
```

As described above, this allows **pppoe** interfaces to be created and configured for incoming connections by setting the "`link0`" flag with `ifconfig`(8).

SEE ALSO

`ifwatchd`(8), `pppoectl`(8)

A Method for Transmitting PPP Over Ethernet (PPPoE), RFC, 2516, February 1999.

HISTORY

The **pppoe** device appeared in NetBSD 1.6.

DEVIATIONS FROM STANDARD

The PPPoE standard, `RFC2516`, requires a maximal MTU of 1492 octets. This value is the maximum conservative value possible, based on the PPPoE header size and the minimum frame size Ethernet interfaces are required to support.

In practice most modern Ethernet interfaces support bigger frames, and many PPPoE services allow the use of (slightly) larger MTUs, to avoid the problems described above.

This implementation allows MTU values as large as possible with the actual MTU of the used Ethernet interface.

BUGS

When using the wildcard address 0.0.0.0 (as described above) it is important to specify the proper "`netmask`" to `ifconfig`(8), in most setups "`0xffffffff`". If the netmask is unspecified, it will be set to 8 when 0.0.0.0 is configured to the interface, and it will persist after negotiation.

NAME

ps — Evans and Sutherland Picture System 2 graphics device interface

SYNOPSIS

```
ps0 at uba? csr 0172460 vector psclockintr pssystemintr
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **ps** driver provides access to an Evans and Sutherland Picture System 2 graphics device. Each minor device is a new PS2. When the device is opened, its interface registers are mapped, via virtual memory, into a user process's address space. This allows the user process very high bandwidth to the device with no system call overhead.

DMA to and from the PS2 is not supported. All read and write system calls will fail. All data is moved to and from the PS2 via programmed I/O using the device's interface registers.

Commands are fed to and from the driver using the following `ioctl`(2)s:

`PSIOGETADDR` Returns the virtual address through which the user process can access the device's interface registers.

`PSIOAUTOREFRESH` Start auto refreshing the screen. The argument is an address in user space where the following data resides. The first longword is a *count* of the number of static refresh buffers. The next *count* longwords are the addresses in refresh memory where the refresh buffers lie. The driver will cycle through these refresh buffers displaying them one by one on the screen.

`PSIOAUTOMAP` Start automatically passing the display file through the matrix processor and into the refresh buffer. The argument is an address in user memory where the following data resides. The first longword is a *count* of the number of display files to operate on. The next *count* longwords are the address of these display files. The final longword is the address in refresh buffer memory where transformed coordinates are to be placed if the driver is not in double buffer mode (see below).

`PSIODOUBLEBUFFER` Cause the driver to double buffer the output from the map that is going to the refresh buffer. The argument is again a user space address where the real arguments are stored. The first argument is the starting address of refresh memory where the two double buffers are located. The second argument is the length of each double buffer. The refresh mechanism displays the current double buffer, in addition to its static refresh lists, when in double buffer mode.

`PSIOSINGLEREFRESH` Single step the refresh process. That is, the driver does not continually refresh the screen.

`PSIOSINGLEMAP` Single step the matrix process. The driver does not automatically feed display files through the matrix unit.

`PSIOSINGLEBUFFER` Turn off double buffering.

`PSIOTIMEREFRESH` The argument is a count of the number of refresh interrupts to take before turning off the screen. This is used to do time exposures.

`PSIOWAITREFRESH` Suspend the user process until a refresh interrupt has occurred. If in `TIMEREFRESH` mode, suspend until count refreshes have occurred.

PSIOSTOPREFRESH	Wait for the next refresh, stop all refreshes, and then return to user process.
PSIOWAITMAP	Wait until a map done interrupt has occurred.
PSIOSTOPMAP	Wait for a map done interrupt, do not restart the map, and then return to the user.

FILES

/dev/ps

DIAGNOSTICS

ps device intr.

ps DMA intr. An interrupt was received from the device. This shouldn't happen, check your device configuration for overlapping interrupt vectors.

HISTORY

The **ps** driver appeared in 4.2BSD.

BUGS

An invalid access (e.g., longword) to a mapped interface register can cause the system to crash with a machine check. A user process could possibly cause infinite interrupts hence bringing things to a crawl.

NAME

psh3lcd — PERSONA SH3 LCD screen driver

SYNOPSIS

```
psh3lcd* at shb?
```

DESCRIPTION

The **psh3lcd** driver provides support for controlling brightness, contrast, and power of the LCD screen in PERSONA SH3 series machines.

In the default keymap, ⟨Ctrl⟩ + ⟨Alt⟩ + ⟨F7⟩ and ⟨F8⟩ control brightness, and ⟨Ctrl⟩ + ⟨Alt⟩ + ⟨F5⟩ and ⟨F6⟩ control contrast.

HISTORY

The **psh3lcd** driver first appeared in NetBSD 3.0.

NAME

psh3tp — driver for PERSONA SH3 touch-panel

SYNOPSIS

```
psh3tp* at adc?
wsmouse* at psh3tp? mux 0
```

DESCRIPTION

The **psh3tp** driver provides support for the PERSONA SH3 touch-panel.

Pen movements are passed to wsmouse(4) as mouse clicks and drags.

SEE ALSO

adc(4), wsmouse(4), tpctl(8)

HISTORY

The **psh3tp** driver first appeared in NetBSD 3.0.

NAME

ptm — pseudo-terminal multiplexor device

SYNOPSIS

pseudo-device pty [*count*]

DESCRIPTION

The **ptm** driver is the backend for the `/dev/ptm` device. It supports three `ioctl`(2)s. The first is `TIOCPTMGET`, which allocates a free pseudo-terminal device, sets its user ID to the calling user, `revoke`(2)s it, and returns the opened file descriptors for both the master and the slave pseudo-terminal device to the caller in a *struct ptmget*. This struct has the following content:

```
struct ptmget {
        int     cfd;
        int     sfd;
        char    cn[16];
        char    sn[16];
};
```

where *cfd* and *sfd* contain the master resp. slave device's file descriptor and *cn* and *sn* the corresponding paths in the file system.

The `/dev/ptmx` device supports two more `ioctl`(2)s, `TIOCGRANTPT`, which is used by `grantpt`(3), `TIOCPTSNAME`, which is used by `ptsname`(3).

The **ptm** device is included with the pseudo-device `pty`(4). It can be disabled by adding "**options NO_DEV_PTM**" to the kernel configuration.

FILES

`/dev/ptm`	**ptm** access device
`/dev/ptmx`	**ptm** cloning device, used to implement Unix98 ptys

SEE ALSO

`grantpt`(3), `openpty`(3), `posix_openpt`(3), `ptsname`(3), `unlockpt`(3), `pty`(4)

HISTORY

The `/dev/ptm` device appeared in OpenBSD 3.5 and was ported to NetBSD 3.0.

NAME

ptsc — Powertec SCSI II Card device interface

SYNOPSIS

```
ptsc0 at podulebus?
```

DESCRIPTION

The **ptsc** interface provides access to Powertec SCSI II interfaces.

SEE ALSO

asc(4), cosc(4), csc(4), oak(4)

NAME

pty — pseudo terminal driver

SYNOPSIS

pseudo-device pty [*count*]

DESCRIPTION

The **pty** driver provides support for a device-pair termed a *pseudo terminal*. A pseudo terminal is a pair of character devices, a *master* device and a *slave* device. The slave device provides to a process an interface identical to that described in tty(4). However, whereas all other devices which provide the interface described in tty(4) have a hardware device of some sort behind them, the slave device has, instead, another process manipulating it through the master half of the pseudo terminal. That is, anything written on the master device is given to the slave device as input and anything written on the slave device is presented as input on the master device.

In configuring, if an optional *count* is given in the specification, that number of pseudo terminal pairs is initially configured; the default count is 16. Additional pseudo terminal pairs are allocated on as-needed basis, maximum number of them is controlled via *kern.maxptys* sysctl (defaults to 992).

The following ioctl(2) calls apply only to pseudo terminals:

TIOCSTOP Stops output to a terminal (e.g. like typing '^S'). Takes no parameter.

TIOCSTART Restarts output (stopped by TIOCSTOP or by typing '^S'). Takes no parameter.

TIOCPKT Enable/disable *packet* mode. Packet mode is enabled by specifying (by reference) a nonzero parameter and disabled by specifying (by reference) a zero parameter. When applied to the master side of a pseudo terminal, each subsequent read(2) from the terminal will return data written on the slave part of the pseudo terminal preceded by a zero byte (symbolically defined as TIOCPKT_DATA), or a single byte reflecting control status information. In the latter case, the byte is an inclusive-or of zero or more of the bits:

TIOCPKT_FLUSHREAD whenever the read queue for the terminal is flushed.

TIOCPKT_FLUSHWRITE whenever the write queue for the terminal is flushed.

TIOCPKT_STOP whenever output to the terminal is stopped a la '^S'.

TIOCPKT_START whenever output to the terminal is restarted.

TIOCPKT_DOSTOP whenever *t_stopc* is '^S' and *t_startc* is '^Q'.

TIOCPKT_NOSTOP whenever the start and stop characters are not ^S/^Q.

While this mode is in use, the presence of control status information to be read from the master side may be detected by a select(2) for exceptional conditions.

This mode is used by rlogin(1) and rlogind(8) to implement a remote-echoed, locally ^S/^Q flow-controlled remote login with proper back-flushing of output; it can be used by other similar programs.

TIOCPKT_IOCTL When this bit is set, the slave has changed the termios(4) structure (TTY state), and the remainder of the data read from the master side of the **pty** is a copy of the new termios(4) structure.

This is used by `telnetd`(8) to implement TELNET "line mode" - it allows the `telnetd`(8) to detect `tty`(4) state changes by the slave, and negotiate the appropriate TELNET protocol equivalents with the remote peer.

`TIOCUCNTL` Enable/disable a mode that allows a small number of simple user `ioctl`(2) commands to be passed through the pseudo-terminal, using a protocol similar to that of `TIOCPKT`. The `TIOCUCNTL` and `TIOCPKT` modes are mutually exclusive. This mode is enabled from the master side of a pseudo terminal by specifying (by reference) a nonzero parameter and disabled by specifying (by reference) a zero parameter. Each subsequent `read`(2) from the master side will return data written on the slave part of the pseudo terminal preceded by a zero byte, or a single byte reflecting a user control operation on the slave side. A user control command consists of a special `ioctl`(2) operation with no data; the command is given as `UIOCCMD`(n), where *n* is a number in the range 1-255. The operation value *n* will be received as a single byte on the next `read`(2) from the master side. The `ioctl`(2) `UIOCCMD`(0) is a no-op that may be used to probe for the existence of this facility. As with `TIOCPKT` mode, command operations may be detected with a `select`(2) for exceptional conditions.

`TIOCREMOTE` A mode for the master half of a pseudo terminal, independent of `TIOCPKT`. This mode causes input to the pseudo terminal to be flow controlled and not input edited (regardless of the terminal mode). Each write to the control terminal produces a record boundary for the process reading the terminal. In normal usage, a write of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an end-of-file character. `TIOCREMOTE` can be used when doing remote line editing in a window manager, or whenever flow controlled input is required.

FILES

`/dev/pty[p-zP-T][0-9a-zA-Z]` master pseudo terminals
`/dev/tty[p-zP-T][0-9a-zA-Z]` slave pseudo terminals

DIAGNOSTICS

None.

SEE ALSO

`ioctl`(2), `read`(2), `select`(2), `write`(2), `openpty`(3), `tty`(4)

HISTORY

The **`pty`** driver appeared in 4.2BSD.

NAME

puc — PCI “universal” communications card driver

SYNOPSIS

```
puc* at pci? dev ? function ?
com* at puc? port ?
lpt* at puc? port ?
```

DESCRIPTION

The **puc** driver provides support for PCI communications cards containing simple communications ports, such as NS16550-family (**com**) serial ports and standard PC-like (**lpt**) parallel ports. The driver is called “universal” because the interfaces to these devices aren’t nearly as well defined and standard as they should be.

The driver currently supports the following cards:

Dolphin Peripherals 4014 (dual parallel)
Dolphin Peripherals 4035 (dual serial)
NetMos NM9835 (dual parallel and single serial)
SIIG Cyber 2P1S PCI (dual parallel and single serial)
SIIG Cyber 2S1P PCI (dual serial and single parallel)
SIIG Cyber 4S PCI (quad serial)
SIIG Cyber I/O PCI (single serial and single parallel)
SIIG Cyber Parallel Dual PCI (dual parallel)
SIIG Cyber Parallel PCI (single parallel)
SIIG Cyber Serial Dual PCI (dual serial)
SIIG Cyber Serial PCI (single serial)
VScom PCI-200 (dual serial)
VScom PCI-400 (4-port serial
VScom PCI-800 (8-port serial)
Titan PCI-800 (8-port serial)
Titan PCI-200 (dual serial)
NEC PK-UG-X001 K56flex PCI Modem
NEC PK-UG-X008
Lava Computers 2SP-PCI (single parallel)
Lava Computers dual serial
Lava Computers Octopus (8-port serial)
US Robotics (3Com) 3CP5609 PCI 16550 Modem
Actiontec 56K PCI Master
Oxford Semiconductor OX16PCI952 (dual serial and single parallel)
Oxford Semiconductor OX16PCI954 (4-port serial)
NetMos 1P PCI (single parallel)
NetMos 2S1P PCI 16C650 (dual serial and single parallel)
NetMos 4S1P PCI NM9845 (4-port serial and single parallel)
Middle Digital, Inc. Weasel serial port
Avlab Technology, Inc. Low Profile PCI 4 Serial (4-port serial)
Boca Research Turbo Serial 654 (4-port serial)
Chase Research / Perle PCI-FAST4 (4-port serial)
Boca Research Turbo Serial 658 (8-port serial)
Chase Research / Perle PCI-FAST8 (8-port serial)

ADDI-DATA APCI-7800 (8-port serial)
Moxa Technologies SmartIO CP104/PCI (4-port serial)
EXAR XR17D152 (2-port serial)
EXAR XR17D154 (4-port serial)
EXAR XR17D158 (8-port serial)
Digi International Digi Neo 4 (4-port serial)
Digi International Digi Neo 8 (8-port serial)

The driver does not support the cards:

Dolphin Peripherals 4006 (single parallel)
Dolphin Peripherals 4025 (single serial)
Dolphin Peripherals 4078 (dual serial and single parallel)

but support for them (and for similar cards) should be trivial to add.

The *port* locator is used to identify the port (starting from 0) on the communications card that a subdevice is supposed to attach to. Typically, the numbering of ports is explained in a card's hardware documentation, and the port numbers used by the driver are the same as (or one off from, e.g. the manual uses ports numbered starting from 1) those described in the documentation.

SEE ALSO

com(4), lpt(4), pci(4)

HISTORY

The **puc** driver appeared in NetBSD 1.4.

AUTHORS

The **puc** driver was written by Chris Demetriou.

BUGS

The current design of this driver keeps any **com** ports on these cards from easily being used as console. Of course, because boards with those are PCI boards, they also suffer from dynamic address assignment, which also means that they can't easily be used as console.

Some of the cards supported by this driver have jumper-selectable **com** port clock multipliers, which are unsupported by this driver. Those can be easily accommodated with driver flags, or by using a properly scaled baud rate when talking to the card.

Some of the cards supported by this driver, e.g. the VScom PCI-800, have software-selectable **com** port clock multipliers, which are unsupported by this driver. Those can be accommodated using internal driver flags, or by using a properly scaled baud rate when talking to the card.

Some ports use an **lpt** driver other than the machine-independent driver. Those ports will not be able to use **lpt** ports attached to **puc** devices.

NAME

pud — Pass-to-Userspace Device

SYNOPSIS

```
pseudo-device pud
```

DESCRIPTION

The **pud** driver enables the implementation of block and character device drivers as userspace daemons. The daemons register the device major number they wish to handle. Registering a character device is mandatory, supporting the block device interface for same major device is optional. The major number must be available, i.e. another driver must not be registered to handle the operation. After succesful registration the userspace daemon is supposed to handle the driver methods the kernel passes down to it.

SEE ALSO

`puffs`(4), `putter`(9)

BUGS

This document is in a hit-in-the-head style obviously not even near complete. But that does not matter, as neither is the code.

NAME

puffs — Pass-to-Userspace Framework File System

SYNOPSIS

```
file-system PUFFS
pseudo-device putter
```

DESCRIPTION

THIS DOCUMENT IS HOPELESSLY OUT OF DATE. While some parts are still valid, please refer to the source code for current reality.

IMPORTANT NOTE! This document describes interfaces which are not yet guaranteed to be stable. In case you update your system sources, please recompile everything and fix compilation errors. If your sources are out-of-sync, incorrect operation may result.

puffs provides a framework for creating file systems as userspace servers. The in-kernel VFS attachment is controlled through a special device node, `/dev/puffs`. This document describes the operations on the device. People looking to implement file systems should prefer using the system through the convenience library described in `puffs`(3). Users wanting to access the device node directly should include the header `sys/fs/puffs/puffs_msgif.h` for relevant definitions.

Mounting

The **puffs** device node should be opened once per file system instance (i.e. mount). The device itself is a cloning node, so the same node can be opened a practically unlimited number of times. Once the device is open, the file system can be mounted the normal way using the `mount`(2) system call and using the argument structure to control mount options:

```
struct puffs_args {
        int             pa_vers;
        int             pa_fd;
        unsigned int    pa_flags;
        size_t          pa_maxreqlen;
        char            pa_name[PUFFSNAMESIZE];
        uint8_t         pa_vnopmask[PUFFS_VN_MAX];
};
```

The member *pa_vers* is currently always 0 and ignored. The *pa_fd* member is the file descriptor number from opening the device node. *pa_flags* controls some operations specific to puffs:

`PUFFS_KFLAG_ALLOWCTL` Allow file system fcntl and ioctl operations. Allowing these has security implications as the file system can technically read anything out of a calling processes address space. This flag may additionally be enforced by the kernel security policy.

`PUFFS_KFLAG_NOCACHE` Do not store data in the page cache. This causes operations to always consult the user server instead of consulting the page cache. This makes sense in situations where there is relatively little bulk data to be transferred and the user server does not want to take part in complex cache management routines in case the file system data can be modified through routes other than the file system interface.

`PUFFS_KFLAG_ALLOPS` Transport all vnode operations to the file system server instead of just the ones specified by *pa_vnopmask*.

The *pa_maxreqlen* member signifies the length of the incoming data buffer in userspace. A good value is PUFFS_REQ_MAXSIZE, which is the maximum the kernel will use. A minimum value is also enforced, so the value of this field should be checked after the mount operation to determine the correct buffer size. During operation, in case request fetch is attempted with a buffer too short, the error E2BIG will be returned. The file system type is give in *pa_name*. It will always be prepended by "puffs:" by the kernel. Finally, the array *pa_vnopmask* specifies which operations are supported by the file system server. The array is indexed with PUFFS_VN_FOO and 0 means vnode operation FOO is unimplemented while non-zero means an implemented operation. This array is ignored if PUFFS_KFLAG_ALLOPS is given.

After a successful mount system call, the the ioctl PUFFSSTARTOP must be issued through the open device node. The parameter for this ioctl is the following structure:

```
struct puffs_startreq {
        void           *psr_cookie;
        struct statvfs psr_sb;
};
```

The member *psr_cookie* should be set before calling. This signals the cookie value of the root node of the file system (see puffs(3) for more details on cookie strategies). The value of *psr_sb* should be filled with the same results as for a regular statvfs call. After successfully executing this operation the file system is active.

Operation

Operations must be queried from the kernel using the ioctl PUFFSGETOP, processed, and the results pushed back to the kernel using PUFFSPUTOP. Normally the system will block until an event is available for PUFFSGETOP, but it is possible to set the file descriptor into non-blocking mode, in which case EWOULDBLOCK is returned if no event is available. Asynchronous I/O calls (i.e., select(2), poll(2), and kevent(2)) can be issued to be notified of events.

As the argument both get and push use the following structure:

```
struct puffs_req {
        uint64_t        preq_id;
        uint8_t         preq_opclass;
        uint8_t         preq_optype;
        void           *preq_cookie;

        int             preq_rv;

        void           *preq_aux;
        size_t          preq_auxlen;
};
```

The member *preq_id* is used as an identifier in the reply. It should not be modified during the processing of a PUFFSGETOP - PUFFSPUTOP sequence. The members *preq_opclass* and *preq_optype* identify the request; they also are used for typing the data pointed to by *preq_aux*. Currently the mapping between these two is only documented in code in src/lib/libpuffs/puff.c:puffcall(). The handling of this will very likely change in the future towards a more automatic direction. The length of the buffer given to PUFFSGETOP is described by *preq_auxlen* and will be modified by the kernel to indicate how much data actually was transmitted. This is for the benefit of calls such as write, which transmit a variable amount of data. Similarly, the user server should fill in the amount of data the kernel must copy for PUFFSPUTOP; most of the time this will be constant for a given operation, but operations such as read want to adjust it dynamically. Finally, *preq_rv* is used by the userspace server to fill in the success value of the operation in question.

In case the macro **PUFFSOP_WANTREPLY**() returns false for *preq_opclass*, a return value is not wanted and PUFFSPUTOP should not be issued.

Additionally, an operation of type PUFFSSIZEOP is supported, but it is only used by the ioctl and fcntl operations and will likely go away in the future. It is not described here.

Termination

The file system can be unmounted regularly using umount(8). It will automatically be unmounted in case the userspace server is killed or the control file descriptor closed, but in this case the userspace server will not be separately requested to unmount itself and this may result in data loss.

SEE ALSO

ioctl(2), mount(2), puffs(3), umount(8)

Antti Kantee, "puffs - Pass-to-Userspace Framework File System", *Proceedings of AsiaBSDCon 2007*, pp. 29-42, March 2007.

HISTORY

An unsupported experimental version of **puffs** first appeared in NetBSD 4.0.

AUTHORS

Antti Kantee ⟨pooka@iki.fi⟩

BUGS

puffs is currently more like a souffle than puff pastry.

NAME

pvr — PowerVR2 graphics driver

SYNOPSIS

```
pvr0       at shb?
wsdisplay* at pvr? console ?
```

DESCRIPTION

The **pvr** driver provides support for NEC PowerVR2 graphics. Access to the graphics is through the wscons(4) driver.

SEE ALSO

wsdisplay(4)

HISTORY

The **pvr** device driver appeared in NetBSD 1.6.

NAME

px — driver for TURBOchannel PX accelerated graphics boards

SYNOPSIS

```
px* at tc? slot ? offset ?
wsdisplay* at px?
```

DESCRIPTION

The **px** driver provides support for the 2D DEC PixelStamp video display option card (PMAG-C).

SEE ALSO

pxg(4), tc(4), wscons(4)

HISTORY

The **px** driver first appeared in NetBSD 1.3. It became usable as a console device in NetBSD 1.5.

BUGS

NetBSD/pmax does not currently support the machine-independent wscons(4) interface and uses a machine-dependent driver.

NAME

pxg — driver for TURBOchannel PXG accelerated graphics boards

SYNOPSIS

```
pxg* at tc? slot ? offset ?
wsdisplay* at pxg?
```

DESCRIPTION

The **pxg** driver provides support for the 3D DEC PixelStamp video display option cards (PMAG-C, PMAG-E, and PMAG-F).

SEE ALSO

px(4), tc(4), wscons(4)

HISTORY

The **pxg** driver first appeared in NetBSD 1.6. Previously, the **px** driver supported these devices in NetBSD/pmax.

BUGS

NetBSD/pmax does not currently support the machine-independent wscons(4) interface and uses a machine-dependent driver.

NAME

qe — DEC DEQNA/DELQA Q-bus 10 Mb/s Ethernet interface

SYNOPSIS

```
qe0 at uba? csr 174440
qe1 at uba? csr 174460
```

DESCRIPTION

The **qe** interface provides access to a 10 Mb/s Ethernet network through the DEC DEQNA or DELQA Q-bus controller.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **qe** interface employs the address resolution protocol described in `arp`(4) to map dynamically between Internet and Ethernet addresses on the local network.

SEE ALSO

`arp`(4), `inet`(4), `netintro`(4)

HISTORY

The **qe** driver appeared in 4.3BSD.

NAME

qe — SPARC Fast Ethernet interface

SYNOPSIS

```
qec* at sbus? slot ? offset ?
qe* at qec?
```

DESCRIPTION

The **qe** interface provides access to 10Mb/s Ethernet networks via the AMD Am79C940 (MACE) Ethernet controller. The **qe** is found on the Sun QuadEthernet boards (Sun part number SUNW,501-2062).

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **qe** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

SEE ALSO

`arp`(4), `be`(4), `hme`(4), `ie`(4), `ifmedia`(4), `inet`(4), `intro`(4), `le`(4), `netintro`(4), `qec`(4), `sbus`(4), `ifconfig`(8)

HISTORY

Support for the **qe** first appeared in NetBSD 1.4.

NAME

qec — SPARC Quad Ethernet Controller

SYNOPSIS

```
qec* at sbus? slot ? offset ?
qe* at qec?
be* at qec?
```

DESCRIPTION

The **qec** driver is an Sbus controller that can contain either one be(4) Ethernet controller (Sun part number SUNW,501-2655) or four qe(4) Ethernet controllers (Sun part number SUNW,501-2062). This driver, like other busses, is not directly available to users. In essence it is a buffering DMA controller.

SEE ALSO

be(4), intro(4), qe(4), sbus(4)

HISTORY

Support for the **qec** first appeared in NetBSD 1.4.

NAME

qms — Quadrature mouse driver

SYNOPSIS

```
qms* at iomd?
wsmouse* at qms?
```

DESCRIPTION

This driver provides an interface to an IOMD quadrature mouse, found on IOMD20-based systems such as the Acorn Risc PC. The **qms** driver interfaces to the wsmouse(4) driver to provide a generic, machine-independent mouse interface.

SEE ALSO

pms(4), wsmouse(4)

NAME

qn — Ethernet driver for Fujitsu MB86950 based boards

SYNOPSIS

qn* **at zbus0**

DESCRIPTION

The **qn** interface provides access to the 10 Mb/s Ethernet network via the Fujitsu MB86950 Ethernet chip set.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **qn** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

HARDWARE

The **qn** interface supports the following Zorro II expansion cards:

QuickNet Resource Management Force's Ethernet card, manufacturer 2011, product 2

NOTES

Multicasting not done yet. If the driver happens to lock up, you may use `ifconfig`(8) to force the driver down and up again. This usually helps.

SEE ALSO

`arp`(4), `inet`(4), `intro`(4), `ifconfig`(8)

HISTORY

The **qn** interface first appeared in NetBSD 1.1

NAME

qsphy — Driver for Quality Semiconductor QS6612 10/100 Ethernet PHYs

SYNOPSIS

```
qsphy* at mii? phy ?
```

DESCRIPTION

The **qsphy** driver supports the Quality Semiconductor QS6612 10/100 Ethernet PHYs. These PHYs are found on a variety of high-performance Ethernet interfaces.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

NAME

qt — DEC DELQA-PLUS (DELQA-YM) Q-bus 10 Mb/s Ethernet interface

SYNOPSIS

```
qt0 at uba? csr 174440
qt1 at uba? csr 174460
```

DESCRIPTION

The **qt** interface provides access to a 10 Mb/s Ethernet network through a DEC DELQA-YM Q-bus controller.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **qt** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

DIAGNOSTICS

qt%d: chained packet. A packet received from the network was longer than the allowed max length and is therefore discarded. This is usually because a host on the local network is behaving badly.

SEE ALSO

`arp`(4), `inet`(4), `netintro`(4)

HISTORY

The **qt** driver first appeared in 2.11BSD. It was later ported to NetBSD 2.0.

NAME

radio — device-independent radio driver layer

SYNOPSIS

```
radio* at az?
radio* at bktr?
radio* at mr?
radio* at rt?
radio* at rtii?
radio* at sf2r?
radio* at udsbr?

#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/radioio.h>
```

DESCRIPTION

The **radio** driver provides support for various FM radio cards. It provides an uniform programming interface layer above different underlying radio hardware drivers.

For radio tuner controlling there is a single device file available: `/dev/radio`.

The following `ioctl`(2) commands are supported:

`RIOCSSRCH (int)`

This command assumes that a signal search is required and gives direction of search to the driver − 0 to search down and any non-zero value to search up.

`RIOCGINFO (struct radio_info)`
`RIOCSINFO (struct radio_info)`

Get or set the current hardware device information into the struct radio_info structure.

```
struct radio_info {
        int     mute;
        int     volume;
        int     stereo;
        int     rfreq;  /* reference frequency */
        int     lock;   /* locking field strength */
        uint32_t        freq;   /* in kHz */
        uint32_t        caps;   /* card capabilities */
#define RADIO_CAPS_DETECT_STEREO      (1<<0)
#define RADIO_CAPS_DETECT_SIGNAL      (1<<1)
#define RADIO_CAPS_SET_MONO           (1<<2)
#define RADIO_CAPS_HW_SEARCH          (1<<3)
#define RADIO_CAPS_HW_AFC             (1<<4)
#define RADIO_CAPS_REFERENCE_FREQ     (1<<5)
#define RADIO_CAPS_LOCK_SENSITIVITY   (1<<6)
#define RADIO_CARD_TYPE                       (0xFF<<16)
        uint32_t        info;
#define RADIO_INFO_STEREO             (1<<0)
#define RADIO_INFO_SIGNAL             (1<<1)
};
```

The *mute* field is a boolean.

The *volume* field holds the card volume information and can be at most 255.

The *stereo* field is a boolean.

The *rfreq* holds information about the card reference frequency (not all cards support this feature).

The *lock* field holds information about the card locking field strength during an automatic search for cards that support this feature.

The *freq* field is the frequency in kHz the card is tuned to.

The *caps* field is read-only and describes the card capabilities. The capabilities can have following values:

`RADIO_CAPS_DETECT_STEREO`
: The device can determine is it tuned to a stereo signal.

`RADIO_CAPS_DETECT_SIGNAL`
: The device can determine is it tuned or not.

`RADIO_CAPS_SET_MONO`
: The device capable to forcible set its output to mono.

`RADIO_CAPS_HW_SEARCH`
: The device can do hardware search.

`RADIO_CAPS_HW_AFC`
: The device has an internal hardware automatic frequency control.

`RADIO_CAPS_REFERENCE_FREQ`
: The device allow to change the reference frequency of a received signal.

`RADIO_CAPS_LOCK_SENSITIVITY`
: The device allow to change the station lock sensitivity used during search operation.

`RADIO_CARD_TYPE`
: Some cards have several different incarnations. This allow to determine the variant of the card. Currently not used.

The *info* field is read-only and describes the current state of the card − tuned/not tuned, stereo signal/mono signal.

`RADIO_INFO_STEREO`
: Informs whether the device receives a stereo or mono signal.

`RADIO_INFO_SIGNAL`
: Informs whether the device receives a valid signal or noise.

FILES

`/dev/radio`

SEE ALSO

`radioctl`(1), `ioctl`(2), `az`(4), `bktr`(4), `mr`(4), `rt`(4), `rtii`(4), `sf2r`(4), `udsbr`(4)

HISTORY

The **`radio`** device driver appeared in OpenBSD 3.0 and NetBSD 1.6.

AUTHORS

The **`radio`** driver was written by Vladimir Popov and Maxim Tsyplakov. The man page was written by Vladimir Popov.

NAME

raid — RAIDframe disk driver

SYNOPSIS

```
options RAID_AUTOCONFIG
options RAID_DIAGNOSTIC
options RF_ACC_TRACE=n
options RF_DEBUG_MAP=n
options RF_DEBUG_PSS=n
options RF_DEBUG_QUEUE=n
options RF_DEBUG_QUIESCE=n
options RF_DEBUG_RECON=n
options RF_DEBUG_STRIPELOCK=n
options RF_DEBUG_VALIDATE_DAG=n
options RF_DEBUG_VERIFYPARITY=n
options RF_INCLUDE_CHAINDECLUSTER=n
options RF_INCLUDE_EVENODD=n
options RF_INCLUDE_INTERDECLUSTER=n
options RF_INCLUDE_PARITY_DECLUSTERING=n
options RF_INCLUDE_PARITY_DECLUSTERING_DS=n
options RF_INCLUDE_PARITYLOGGING=n
options RF_INCLUDE_RAID5_RS=n

pseudo-device raid [count]
```

DESCRIPTION

The **raid** driver provides RAID 0, 1, 4, and 5 (and more!) capabilities to NetBSD. This document assumes that the reader has at least some familiarity with RAID and RAID concepts. The reader is also assumed to know how to configure disks and pseudo-devices into kernels, how to generate kernels, and how to partition disks.

RAIDframe provides a number of different RAID levels including:

RAID 0 provides simple data striping across the components.

RAID 1 provides mirroring.

RAID 4 provides data striping across the components, with parity stored on a dedicated drive (in this case, the last component).

RAID 5 provides data striping across the components, with parity distributed across all the components.

There are a wide variety of other RAID levels supported by RAIDframe. The configuration file options to enable them are briefly outlined at the end of this section.

Depending on the parity level configured, the device driver can support the failure of component drives. The number of failures allowed depends on the parity level selected. If the driver is able to handle drive failures, and a drive does fail, then the system is operating in "degraded mode". In this mode, all missing data must be reconstructed from the data and parity present on the other components. This results in much slower data accesses, but does mean that a failure need not bring the system to a complete halt.

The RAID driver supports and enforces the use of ‘component labels’. A ‘component label’ contains important information about the component, including a user-specified serial number, the row and column of that component in the RAID set, and whether the data (and parity) on the component is ‘clean’. The component label currently lives at the half-way point of the ‘reserved section’ located at the beginning of each component. This ‘reserved section’ is RF_PROTECTED_SECTORS in length (64 blocks or 32Kbytes) and the

component label is currently 1Kbyte in size.

If the driver determines that the component labels are very inconsistent with respect to each other (e.g. two or more serial numbers do not match) or that the component label is not consistent with its assigned place in the set (e.g. the component label claims the component should be the 3rd one in a 6-disk set, but the RAID set has it as the 3rd component in a 5-disk set) then the device will fail to configure. If the driver determines that exactly one component label seems to be incorrect, and the RAID set is being configured as a set that supports a single failure, then the RAID set will be allowed to configure, but the incorrectly labeled component will be marked as 'failed', and the RAID set will begin operation in degraded mode. If all of the components are consistent among themselves, the RAID set will configure normally.

Component labels are also used to support the auto-detection and autoconfiguration of RAID sets. A RAID set can be flagged as autoconfigurable, in which case it will be configured automatically during the kernel boot process. RAID file systems which are automatically configured are also eligible to be the root file system. There is currently only limited support (alpha, amd64, i386, pmax, sparc, sparc64, and vax architectures) for booting a kernel directly from a RAID 1 set, and no support for booting from any other RAID sets. To use a RAID set as the root file system, a kernel is usually obtained from a small non-RAID partition, after which any autoconfiguring RAID set can be used for the root file system. See `raidctl`(8) for more information on autoconfiguration of RAID sets. Note that with autoconfiguration of RAID sets, it is no longer necessary to hard-code SCSI IDs of drives. The autoconfiguration code will correctly configure a device even after any number of the components have had their device IDs changed or device names changed.

The driver supports 'hot spares', disks which are on-line, but are not actively used in an existing file system. Should a disk fail, the driver is capable of reconstructing the failed disk onto a hot spare or back onto a replacement drive. If the components are hot swappable, the failed disk can then be removed, a new disk put in its place, and a copyback operation performed. The copyback operation, as its name indicates, will copy the reconstructed data from the hot spare to the previously failed (and now replaced) disk. Hot spares can also be hot-added using `raidctl`(8).

If a component cannot be detected when the RAID device is configured, that component will be simply marked as 'failed'.

The user-land utility for doing all **`raid`** configuration and other operations is `raidctl`(8). Most importantly, `raidctl`(8) must be used with the **`-i`** option to initialize all RAID sets. In particular, this initialization includes re-building the parity data. This rebuilding of parity data is also required when either a) a new RAID device is brought up for the first time or b) after an un-clean shutdown of a RAID device. By using the **`-P`** option to `raidctl`(8), and performing this on-demand recomputation of all parity before doing a `fsck`(8) or a `newfs`(8), file system integrity and parity integrity can be ensured. It bears repeating again that parity recomputation is *`required`* before any file systems are created or used on the RAID device. If the parity is not correct, then missing data cannot be correctly recovered.

RAID levels may be combined in a hierarchical fashion. For example, a RAID 0 device can be constructed out of a number of RAID 5 devices (which, in turn, may be constructed out of the physical disks, or of other RAID devices).

The first step to using the **`raid`** driver is to ensure that it is suitably configured in the kernel. This is done by adding a line similar to:

```
pseudo-device   raid   4      # RAIDframe disk device
```

to the kernel configuration file. The 'count' argument ('4', in this case), specifies the number of RAIDframe drivers to configure. To turn on component auto-detection and autoconfiguration of RAID sets, simply add:

```
options RAID_AUTOCONFIG
```

to the kernel configuration file.

All component partitions must be of the type FS_BSDFFS (e.g. 4.2BSD) or FS_RAID. The use of the latter is strongly encouraged, and is required if autoconfiguration of the RAID set is desired. Since RAIDframe leaves room for disklabels, RAID components can be simply raw disks, or partitions which use an entire disk.

A more detailed treatment of actually using a **raid** device is found in raidctl(8). It is highly recommended that the steps to reconstruct, copyback, and re-compute parity are well understood by the system administrator(s) *before* a component failure. Doing the wrong thing when a component fails may result in data loss.

Additional internal consistency checking can be enabled by specifying:

options RAID_DIAGNOSTIC

These assertions are disabled by default in order to improve performance.

RAIDframe supports an access tracing facility for tracking both requests made and performance of various parts of the RAID systems as the request is processed. To enable this tracing the following option may be specified:

options RF_ACC_TRACE=1

For extensive debugging there are a number of kernel options which will aid in performing extra diagnosis of various parts of the RAIDframe sub-systems. Note that in order to make full use of these options it is often necessary to enable one or more debugging options as listed in src/sys/dev/raidframe/rf_options.h. As well, these options are also only typically useful for people who wish to debug various parts of RAIDframe. The options include:

For debugging the code which maps RAID addresses to physical addresses:

options RF_DEBUG_MAP=1

Parity stripe status debugging is enabled with:

options RF_DEBUG_PSS=1

Additional debugging for queuing is enabled with:

options RF_DEBUG_QUEUE=1

Problems with non-quiescent file systems should be easier to debug if the following is enabled:

options RF_DEBUG_QUIESCE=1

Stripelock debugging is enabled with:

options RF_DEBUG_STRIPELOCK=1

Additional diagnostic checks during reconstruction are enabled with:

options RF_DEBUG_RECON=1

Validation of the DAGs (Directed Acyclic Graphs) used to describe an I/O access can be performed when the following is enabled:

options RF_DEBUG_VALIDATE_DAG=1

Additional diagnostics during parity verification are enabled with:

options RF_DEBUG_VERIFYPARITY=1

There are a number of less commonly used RAID levels supported by RAIDframe. These additional RAID types should be considered experimental, and may not be ready for production use. The various types and the options to enable them are shown here:

For Even-Odd parity:

options RF_INCLUDE_EVENODD=1

For RAID level 5 with rotated sparing:

options RF_INCLUDE_RAID5_RS=1

For Parity Logging (highly experimental):

options RF_INCLUDE_PARITYLOGGING=1

For Chain Declustering:

options RF_INCLUDE_CHAINDECLUSTER=1

For Interleaved Declustering:

options RF_INCLUDE_INTERDECLUSTER=1

For Parity Declustering:

options RF_INCLUDE_PARITY_DECLUSTERING=1

For Parity Declustering with Distributed Spares:

options RF_INCLUDE_PARITY_DECLUSTERING_DS=1

The reader is referred to the RAIDframe documentation mentioned in the **HISTORY** section for more detail on these various RAID configurations.

WARNINGS

Certain RAID levels (1, 4, 5, 6, and others) can protect against some data loss due to component failure. However the loss of two components of a RAID 4 or 5 system, or the loss of a single component of a RAID 0 system, will result in the entire file systems on that RAID device being lost. RAID is *NOT* a substitute for good backup practices.

Recomputation of parity *MUST* be performed whenever there is a chance that it may have been compromised. This includes after system crashes, or before a RAID device has been used for the first time. Failure to keep parity correct will be catastrophic should a component ever fail -- it is better to use RAID 0 and get the additional space and speed, than it is to use parity, but not keep the parity correct. At least with RAID 0 there is no perception of increased data security.

FILES

`/dev/{,r}raid*` **raid** device special files.

SEE ALSO

`config`(1), `sd`(4), `MAKEDEV`(8), `fsck`(8), `mount`(8), `newfs`(8), `raidctl`(8)

HISTORY

The **raid** driver in NetBSD is a port of RAIDframe, a framework for rapid prototyping of RAID structures developed by the folks at the Parallel Data Laboratory at Carnegie Mellon University (CMU). RAIDframe, as originally distributed by CMU, provides a RAID simulator for a number of different architectures, and a user-level device driver and a kernel device driver for Digital Unix. The **raid** driver is a kernelized version of RAIDframe v1.1.

A more complete description of the internals and functionality of RAIDframe is found in the paper "RAIDframe: A Rapid Prototyping Tool for RAID Systems", by William V. Courtright II, Garth Gibson, Mark Holland, LeAnn Neal Reilly, and Jim Zelenka, and published by the Parallel Data Laboratory of Carnegie Mellon University. The **raid** driver first appeared in NetBSD 1.4.

COPYRIGHT

The RAIDframe Copyright is as follows:

NAME

ral — Ralink Technology IEEE 802.11a/b/g wireless network driver

SYNOPSIS

```
ral* at cardbus?
ral* at pci?
ral* at uhub? port ?
```

DESCRIPTION

The **ral** driver supports PCI/CardBus wireless adapters based on the Ralink RT2500, RT2501, and RT2600 chipsets. The **ral** driver supports USB 2.0 wireless adapters based on the Ralink RT2500USB chipset.

The RT2500 chipset is the first generation of 802.11b/g adapters from Ralink. It consists of two integrated chips, an RT2560 or RT2570(USB) MAC/BBP and an RT2525 or RT2526(USB) radio transceiver.

The RT2501 chipset is the second generation of 802.11b/g adapters from Ralink. It consists of two integrated chips, an RT2561 MAC/BBP and an RT2527 radio transceiver. This chipset provides support for the IEEE 802.11e standard with multiple hardware transmission queues and allows scatter/gather for efficient DMA operations.

The RT2600 chipset consists of two integrated chips, an RT2661 MAC/BBP and an RT2529 radio transceiver. This chipset uses the MIMO (multiple-input multiple-output) technology with multiple antennas to extend the operating range of the adapter and to achieve higher throughput. MIMO will be the basis of the future IEEE 802.11n standard.

These are the modes the **ral** driver can operate in:

BSS mode — Also known as *infrastructure* mode, this is used when associating with an access point, through which all traffic passes. This mode is the default.

IBSS mode — Also known as *IEEE ad-hoc* mode or *peer-to-peer* mode. This is the standardized method of operating without an access point. Stations associate with a service set. However, actual connections between stations are peer-to-peer.

Host AP — In this mode the driver acts as an access point (base station) for other cards.

monitor mode — In this mode the driver is able to receive packets without associating with an access point. This disables the internal receive filter and enables the card to capture packets from networks which it wouldn't normally have access to, or to scan for access points.

ral supports software WEP. Wired Equivalent Privacy (WEP) is the de facto encryption standard for wireless networks. It can be typically configured in one of three modes: no encryption; 40-bit encryption; or 104-bit encryption. Unfortunately, due to serious weaknesses in WEP protocol it is strongly recommended that it not be used as the sole mechanism to secure wireless communication. WEP is not enabled by default.

The transmit speed is user-selectable or can be adapted automatically by the driver depending on the received signal strength and on the number of hardware transmission retries. See `rssadapt`(9) for more information.

CONFIGURATION

The **ral** driver can be configured at runtime with `ifconfig`(8) or on boot with `ifconfig.if`(5) using the following parameters:

bssid *bssid*
Set the desired BSSID.

-bssid
Unset the desired BSSID. The interface will automatically select a BSSID in this mode, which is the default.

chan *n*
Set the channel (radio frequency) to be used by the driver based on the given channel ID *n*.

-chan
Unset the desired channel to be used by the driver. The driver will automatically select a channel in this mode, which is the default.

media *media*
The **ral** driver supports the following *media* types:

autoselect	Enable autoselection of the media type and options.
DS1	Set 802.11b DS 1Mbps operation.
DS2	Set 802.11b DS 2Mbps operation.
DS5	Set 802.11b DS 5.5Mbps operation.
DS11	Set 802.11b DS 11Mbps operation.
OFDM6	Set 802.11a/g OFDM 6Mbps operation.
OFDM9	Set 802.11a/g OFDM 9Mbps operation.
OFDM12	Set 802.11a/g OFDM 12Mbps operation.
OFDM18	Set 802.11a/g OFDM 18Mbps operation.
OFDM24	Set 802.11a/g OFDM 24Mbps operation.
OFDM36	Set 802.11a/g OFDM 36Mbps operation.
OFDM48	Set 802.11a/g OFDM 48Mbps operation.
OFDM54	Set 802.11a/g OFDM 54Mbps operation.

mediaopt *opts*
The **ral** driver supports the following media options:

hostap	Select Host AP operation.
ibss	Select IBSS operation.
monitor	Select monitor mode.

-mediaopt *opts*
Disable the specified media options on the driver and return it to the default mode of operation (BSS).

mode *mode*
The **ral** driver supports the following modes:

11a	Force 802.11a operation.
11b	Force 802.11b operation.
11g	Force 802.11g operation.

nwid *id*
Set the network ID. The *id* can either be any text string up to 32 characters in length, or a series of hexadecimal digits up to 64 digits. An empty *id* string allows the interface to connect to any available access points. By default the **ral** driver uses an empty string. Note that network ID is synonymous with Extended Service Set ID (ESSID).

nwkey *key*
Enable WEP encryption using the specified *key*. The *key* can either be a string, a series of hexadecimal digits (preceded by '0x'), or a set of keys of the form "n:k1,k2,k3,k4", where 'n' specifies which of the keys will be used for transmitted packets, and the four keys, "k1" through "k4", are configured as WEP keys. If a set of keys is specified, a comma (',') within the key must be

escaped with a backslash. Note that if multiple keys are used, their order must be the same within the network. **ral** is capable of using both 40-bit (5 characters or 10 hexadecimal digits) or 104-bit (13 characters or 26 hexadecimal digits) keys.

-nwkey
Disable WEP encryption. This is the default mode of operation.

FILES

The following firmware files are potentially loaded when an interface is brought up:

/libdata/firmware/ral/ral-rt2561
/libdata/firmware/ral/ral-rt2561s
/libdata/firmware/ral/ral-rt2661

RT2500 adapters do not require a firmware to operate.

HARDWARE

The following PCI adapters should work:

A-Link WL54H. Amigo AWI-926W. AMIT WL531P. AOpen AOI-831. ASUS WL-130g. ASUS WIFI-G-AAY. Atlantis Land A02-PCI-W54. Belkin F5D7000 v3. Canyon CN-WF511. CNet CWP-854. Compex WLP54G. Conceptronic C54Ri. Corega CG-WLPCI54GL. Digitus DN-7006G-RA. Dynalink WLG25PCI. E-Tech WGPI02. Edimax EW-7128g. Eminent EM3037. Encore ENLWI-G-RLAM. Eusso UGL2454-VPR. Fiberline WL-400P. Foxconn WLL-3350. Gigabyte GN-WPKG. Hawking HWP54GR. Hercules HWGPCI-54. iNexQ CR054g-009 (R03). JAHT WN-4054PCI. KCORP LifeStyle KLS-660. LevelOne WNC-0301 v2. Linksys WMP54G v4. Micronet SP906GK. Minitar MN54GPC-R. MSI MS-6834. MSI PC54G2. OvisLink EVO-W54PCI. PheeNet HWL-PCIG/RA. Pro-Nets PC80211G. Repotec RP-WP0854. SATech SN-54P. Signamax 065-1798. Sitecom WL-115. SparkLAN WL-660R. Surecom EP-9321-g. Sweex LC700030. TekComm NE-9321-g. Tonze PC-6200C. Unex CR054g-R02. Zinwell ZWX-G361. Zonet ZEW1600.

The following CardBus adapters should work:

A-Link WL54PC. Alfa AWPC036. Amigo AWI-914W. AMIT WL531C. ASUS WL-107G. Atlantis Land A02-PCM-W54. Belkin F5D7010 v2. Canyon CN-WF513. CC&C WL-2102. CNet CWC-854. Conceptronic C54RC. Corega CG-WLCB54GL. Digitus DN-7001G-RA. Dynalink WLG25CARDBUS. E-Tech WGPC02. E-Tech WGPC03. Edimax EW-7108PCg. Eminent EM3036. Encore ENPWI-G-RLAM. Eusso UGL2454-01R. Fiberline WL-400X. Gigabyte GN-WMKG. Hawking HWC54GR. Hercules HWGPCM-CIA-54. JAHT WN-4054P(E). KCORP LifeStyle KLS-611. LevelOne WPC-0301 v2. Micronet SP908GK V3. Minitar MN54GCB-R. MSI CB54G2. MSI MS-6835. Pro-Nets CB80211G. Repotec RP-WB7108. SATech SN-54C. Sitecom WL-112. SparkLAN WL-611R. Surecom EP-9428-g. Sweex LC500050. TekComm NE-9428-g. Tonze PW-6200C. Unex MR054g-R02. Zinwell ZWX-G160. Zonet ZEW1500.

The following Mini PCI adapters should work:

Amigo AWI-922W. Billionton MIWLGRL. Gigabyte GN-WIKG. MSI MP54G2. MSI MS-6833. Tonze PC-620C. Zinwell ZWX-G360.

The following USB 2.0 adapters should work:

AMIT WL532U. ASUS WL-167g. Belkin F5D7050 v2000. Buffalo WLI-U2-KG54. Buffalo WLI-U2-KG54-AI. Buffalo WLI-U2-KG54-YB. CNet CWD-854. Compex WLU54G 2A1100. Conceptronic C54RU. D-Link DWL-G122 (b1). Dynalink WLG25USB. E-Tech WGUS02. Gigabyte GN-WBKG. Hercules HWGUSB2-54. KCORP LifeStyle KLS-685. Linksys HU200-TS. Linksys WUSB54G v4. Linksys WUSB54GP v4. MSI MS-6861. MSI MS-6865. MSI MS-6869. Nintendo Wi-Fi USB Connector. OvisLink Evo-W54USB. SerComm UB801R. SparkLAN WL-685R. Surecom EP-9001-g. Sweex LC100060. Tonze UW-6200C. Zinwell ZWX-G261. Zonet ZEW2500P.

EXAMPLES

The following `ifconfig.if`(5) example creates a host-based access point on boot:

```
inet 192.168.1.1 netmask 255.255.255.0 media autoselect \
        mediaopt hostap nwid my_net chan 11
```

Configure ral0 for WEP, using hex key "0x1deadbeef1":

```
# ifconfig ral0 nwkey 0x1deadbeef1
```

Return ral0 to its default settings:

```
# ifconfig ral0 -bssid -chan media autoselect \
        nwid "" -nwkey
```

Join an existing BSS network, "my_net":

```
# ifconfig ral0 192.168.1.1 netmask 0xffffff00 nwid my_net
```

DIAGNOSTICS

ral%d: could not read microcode %s For some reason, the driver was unable to read the microcode file from the filesystem. The file might be missing or corrupted.

ral%d: could not load 8051 microcode An error occurred while attempting to upload the microcode to the onboard 8051 microcontroller unit.

ral%d: timeout waiting for MCU to initialize The onboard 8051 microcontroller unit failed to initialize in time.

ral%d: device timeout A frame dispatched to the hardware for transmission did not complete in time. The driver will reset the hardware. This should not happen.

SEE ALSO

`arp`(4), `cardbus`(4), `ifmedia`(4), `intro`(4), `netintro`(4), `pci`(4), `usb`(4), `ifconfig.if`(5), `hostapd`(8), `ifconfig`(8)

Ralink Technology: `http://www.ralinktech.com`

HISTORY

The **`ral`** driver first appeared in OpenBSD 3.7 and in NetBSD 3.0. Support for the RT2501 and RT2600 chipsets was added in OpenBSD 3.9 and in NetBSD 4.0.

AUTHORS

The **`ral`** driver was written by Damien Bergamini ⟨damien@openbsd.org⟩.

CAVEATS

Some PCI **`ral`** adapters seem to strictly require a system supporting PCI 2.2 or greater and will likely not work in systems based on older revisions of the PCI specification. Check the board's PCI version before purchasing the card.

The USB **`ral`** driver supports automatic control of the transmit speed in BSS mode only. Therefore the use of a USB **`ral`** adapter in Host AP mode is discouraged.

NAME

ray — Raytheon Raylink / WebGear Aviator IEEE 802.11 2Mbps Wireless

SYNOPSIS

```
ray* at pcmcia? function ?
options RAY_PID_COUNTRY_CODE_DEFAULT=RAY_PID_COUNTRY_CODE_USA
```

DESCRIPTION

The **ray** device driver supports the Raytheon Raylink and Aviator 2.4/PRO 802.11 FH 2Mbps wireless PCMCIA cards. The cards can be operated in either ad-hoc or infrastructure modes. The operating mode is selectable with `ifconfig`(8) through a media option.

To communicate with other 802.11 cards a common network id or NWID must be specified on each station that wishes to participate in the shared wireless network. The NWID can be set with `ifconfig`(8).

The device uses IEEE 802.11 standard Frequency Hopping Spread Spectrum signaling and operates in the ranges of 2.400 to 2.4835 Gigahertz. This frequency range is further restricted by country according to that country's regulations. Currently the **ray** driver defaults to using the ranges appropriate for the USA. To change this setting you must define the kernel option RAY_PID_COUNTRY_CODE_DEFAULT to one of the following values:

RAY_PID_COUNTRY_CODE_USA
RAY_PID_COUNTRY_CODE_EUROPE
RAY_PID_COUNTRY_CODE_JAPAN
RAY_PID_COUNTRY_CODE_KOREA
RAY_PID_COUNTRY_CODE_SPAIN
RAY_PID_COUNTRY_CODE_FRANCE
RAY_PID_COUNTRY_CODE_ISRAEL
RAY_PID_COUNTRY_CODE_AUSTRALIA

The output power of the transceiver is 100mW and the card's power consumption is 365 mA @ 5 volts. The transmission range in open air (line of sight) is a maximum of 1000 feet (or ˜304 meters), and indoors (i.e., with obstructions) it is a maximum of 500 feet (152 meters).

HARDWARE

Cards supported by the **ray** driver include:

WebGear Aviator 2.4
WebGear Aviator PRO
Raytheon Raylink WLAN

DIAGNOSTICS

ray0: card failed self test: status x Indicates the card has failed its initial startup tests.

SEE ALSO

`ifmedia`(4), `intro`(4), `pcmcia`(4), `ifconfig`(8)

HISTORY

The **ray** driver first appeared in NetBSD 1.5.

AUTHORS

The **ray** driver was written by Christian E. Hopps ⟨chopps@NetBSD.org⟩.

BUGS

Currently the infrastructure mode is untested, and authentication using WEP is unimplemented.

NAME

rbox — HP98720 ‘‘Renaissance’’ graphics device interface

SYNOPSIS

```
rbox* at intio?
rbox* at dio? scode ?
```

DESCRIPTION

This driver is for the HP98720 and 98721 graphics device, also known as the Renaissance. This driver has not been tested with all possible combinations of frame buffer boards and scan boards installed in the device. The driver merely checks for the existence of the device and does minimal set up.

The Renaissance can be configured at either the ‘‘internal’’ address (frame buffer address 0x200000, control register space address 0x560000) or at an external select code less than 32. At the internal address it will be the ‘‘preferred’’ console device. The hardware installation manual describes the procedure for setting these values.

A user process communicates to the device initially by means of `ioctl`(2) calls. For the HP-UX `ioctl`(2) calls supported, refer to HP-UX manuals. The BSD calls supported are:

`GRFIOCGINFO`
: Get Graphics Info

  Get info about device, setting the entries in the *`grfinfo`* structure, as defined in ⟨`hpdev/grfioctl.h`⟩. For the standard 98720, the number of planes should be 4. The number of colors would therefore be 15, excluding black. If one 98722A frame buffer board is installed, there will still be 4 planes, with the 4 planes on the colormap board becoming overlay planes. With each additional 98722 frame buffer board 4 planes will be added up to a maximum of 32 planes total.

`GRFIOCON`
: Graphics On

  Turn graphics on by enabling CRT output. The screen will come on, displaying whatever is in the frame buffer, using whatever colormap is in place.

`GRFIOCOFF`
: Graphics Off

  Turn graphics off by disabling output to the CRT. The frame buffer contents are not affected.

`GRFIOCMAP`
: Map Device to user space

  Map in control registers and framebuffer space. Once the device file is mapped, the frame buffer structure is accessible. The structure describing the 98720 is defined in `hpdev/grf_rbreg.h`.

FILES

`/dev/grf?`	BSD special file
`/dev/crt98720`	
`/dev/ocrt98720`	HP-UX *starbase* special files
`/dev/MAKEDEV.hpux`	script for creating HP-UX special files

EXAMPLES

This is a short segment of code showing how the device is opened and mapped into user process address space assuming that it is grf0:

```
struct rboxfb *rbox;
u_char *Addr, frame_buffer;
struct grfinfo gi;
int disp_fd;

disp_fd = open("/dev/grf0",1);

if (ioctl (disp_fd, GRFIOCGINFO, &gi) < 0) return -1;

(void) ioctl (disp_fd, GRFIOCON, 0);

Addr = (u_char *) 0;
if (ioctl (disp_fd, GRFIOCMAP, &Addr) < 0) {
       (void) ioctl (disp_fd, GRFIOCOFF, 0);
       return -1;
}
rbox = (rboxfb *) Addr;                          /* Control Registers   */
frame_buffer = (u_char *) Addr + gi.gd_regsize; /* Frame buffer memory */
```

DIAGNOSTICS

None under BSD. The HP-UX CE.utilities must be used.

ERRORS

[ENODEV] no such device.

[EBUSY] Another process has the device open.

[EINVAL] Invalid ioctl specification.

SEE ALSO

ioctl(2), grf(4), ite(4)

For extensive code examples using the Renaissance, see the X device dependent source.

BUGS

Not tested for all configurations of scan board and frame buffer memory boards.

NAME

rcons — raster console access

SYNOPSIS

```
options RASTERCONSOLE_FGCOL=n
options RASTERCONSOLE_BGCOL=n
options RCONS_2BPP
options RCONS_4BPP
options RCONS_16BPP
```

pmax

```
pseudo-device rasterconsole N
```

DESCRIPTION

The **rcons** driver provides support for machine-independent access to the raster console. Use of the **rcons** driver is deprecated in favour of the wscons(4) machine-independent console driver.

The **rcons** driver provides simple raster and frame buffer routines. It's enough to implement a console terminal emulator on monochrome and pseudo-colour screens.

SEE ALSO

wscons(4)

NAME

rd — CS/80 disk interface

SYNOPSIS

rd* at hpibbus? slave?

DESCRIPTION

This is a generic CS/80 disk driver. Only a small number of possible CS/80 drives are supported, but others can easily be added by adding tables to the driver. It is a typical block-device driver; see physio(4).

The script MAKEDEV(8) should be used to create the **rd** special files; consult mknod(8) if a special file needs to be made manually.

DISK SUPPORT

The driver interrogates the controller to determine the type of drive attached. The driver recognizes the following drives: 7912, 7914, 7933, 7936, 7937, 7945, 757A/B, 7958A/B, 7959B, 7962, 7963, 9122, 9134, 7912, 7936, and 9122, not all of which have been tested. Special file names begin with ‘rd’ and ‘rrd’ for the block and character files respectively. The second component of the name, a drive unit number in the range of zero to seven, is represented by a ‘?’ in the disk layouts below. The last component of the name is the file system partition and is designated by a letter from ‘a’ to ‘h’ which also corresponds to a minor device number sets: zero to seven, eight to 15, 16 to 23 and so forth for drive zero, drive two and drive three respectively (see physio(4)). The location and size (in sectors) of the partitions for these drives:

7945/7946 partitions:

disk	start	length	cyls
rd?a	112	15904	1-142
rd?b	16016	20160	143-322
rd?c	0	108416	0-967
rd?d	16016	40320	143-502
rd?e	undefined		
rd?f	undefined		
rd?g	36176	72240	323-967
rd?h	56336	52080	503-967

9134D partitions:

disk	start	length	cyls
rd?a	96	15936	1-166
rd?b	16032	13056	167-302
rd?c	0	29088	0-302
rd?d	undefined		
rd?e	undefined		
rd?f	undefined		
rd?g	undefined		
rd?h	undefined		

9122S partitions:

disk	start	length	cyls
rd?a	undefined		
rd?b	undefined		
rd?c	0	1232	0-76
rd?d	undefined		
rd?e	undefined		
rd?f	undefined		
rd?g	undefined		

rd?h	undefined

7912P partitions:

disk	start	length	cyls
rd?a	0	15904	0-70
rd?b	16128	22400	72-171
rd?c	0	128128	0-571
rd?d	16128	42560	72-261
rd?e	undefined		
rd?f	undefined		
rd?g	38528	89600	172-571
rd?h	58688	69440	262-571

7914CT/P partitions:

disk	start	length	cyls
rd?a	224	15904	1-71
rd?b	16128	40320	72-251
rd?c	0	258048	0-1151
rd?d	16128	64960	72-361
rd?e	81088	98560	362-801
rd?f	179648	78400	802-1151
rd?g	56448	201600	252-1151
rd?h	81088	176960	362-1151

7958A partitions:

disk	start	length	cyls
rd?a	252	16128	1-64
rd?b	16380	32256	65-192
rd?c	0	255276	0-1012
rd?d	16380	48384	65-256
rd?e	64764	100800	257-656
rd?f	165564	89712	657-1012
rd?g	48636	206640	193-1012
rd?h	64764	190512	257-1012

7957A partitions:

disk	start	length	cyls
rd?a	154	16016	1-104
rd?b	16170	24640	105-264
rd?c	0	159544	0-1035
rd?d	16170	42350	105-379
rd?e	58520	54824	380-735
rd?f	113344	46200	736-1035
rd?g	40810	118734	265-1035
rd?h	58520	101024	380-1035

7933H partitions:

disk	start	length	cyls
rd?a	598	16146	1-27
rd?b	16744	66976	28-139
rd?c	0	789958	0-1320
rd?d	83720	16146	140-166
rd?e	99866	165646	167-443
rd?f	265512	165646	444-720

rd?g	83720	706238	140-1320
rd?h	431158	358800	721-1320

9134L partitions:

disk	**start**	**length**	**cyls**
rd?a	80	15920	1-199
rd?b	16000	20000	200-449
rd?c	0	77840	0-972
rd?d	16000	32000	200-599
rd?e	undefined		
rd?f	undefined		
rd?g	36000	41840	450-972
rd?h	48000	29840	600-972

7936H partitions:

disk	**start**	**length**	**cyls**
rd?a	861	16359	1-19
rd?b	17220	67158	20-97
rd?c	0	600978	0-697
rd?d	84378	16359	98-116
rd?e	100737	120540	117-256
rd?f	220416	120540	256-395
rd?g	84378	516600	98-697
rd?h	341817	259161	397-697

7937H partitions:

disk	**start**	**length**	**cyls**
rd?a	1599	15990	1-10
rd?b	17589	67158	11-52
rd?c	0	111610	20-697
rd?d	84747	15990	53-62
rd?e	100737	246246	63-216
rd?f	346983	246246	217-370
rd?g	84747	1031355	53-697
rd?h	593229	522873	371-697

7957B/7961B partitions:

disk	**start**	**length**	**cyls**
rd?a	126	16002	1-127
rd?b	16128	32760	128-387
rd?c	0	159894	0-1268
rd?d	16128	49140	128-517
rd?e	65268	50400	518-917
rd?f	115668	44226	918-1268
rd?g	48888	111006	388-1268
rd?h	65268	94626	518-1268

7958B/7962B partitions:

disk	**start**	**length**	**cyls**
rd?a	378	16254	1-43
rd?b	16632	32886	44-130
rd?c	0	297108	0-785
rd?d	16632	49140	44-173
rd?e	65772	121716	174-495

rd?f	187488	109620	496-785
rd?g	49518	247590	131-785
rd?h	65772	231336	174-785

7959B/7963B partitions:

disk	**start**	**length**	**cyls**
rd?a	378	16254	1-43
rd?b	16632	49140	44-173
rd?c	0	594216	0-1571
rd?d	16632	65772	44-217
rd?e	82404	303912	218-1021
rd?f	386316	207900	1022-1571
rd?g	65772	528444	174-1571
rd?h	82404	511812	218-1571

The eight partitions as given support four basic, non-overlapping layouts, though not all partitions exist on all drive types.

In the first layout there are three partitions and a ‘‘bootblock’’ area. The bootblock area is at the beginning of the disk and holds the standalone disk boot program. The `rd?a` partition is for the root file system, `rd?b` is a paging/swapping area, and `rd?g` is for everything else.

The second layout is the same idea, but has a larger paging/swapping partition (`rd?d`) and a smaller ‘‘everything else’’ partition (`rd?h`). This layout is better for environments which run many large processes.

The third layout is a variation of the second, but breaks the `rd?h` partition into two partitions, `rd?e` and `rd?f`.

The final layout is intended for a large, single file system second disk. It is also used when writing out the boot program since it is the only partition mapping the bootblock area.

FILES

`/dev/rd[0-7][a-h]` block files
`/dev/rrd[0-7][a-h]` raw files

DIAGNOSTICS

rd%d err: v%d u%d, R0x%x F0x%x A0x%x I0x%x, block %d An unrecoverable data error occurred during transfer of the specified block on the specified disk.

BUGS

The current disk partitioning is totally bogus. CS/80 drives have 256 byte sectors which are mapped to 512 byte ‘‘sectors’’ by the driver. Since some CS/80 drives have an odd number of sectors per cylinder, the disk geometry used is not always accurate.

The partition tables for the file systems should be read off of each pack, as they are never quite what any single installation would prefer, and this would make packs more portable.

A program to analyze the logged error information (even in its present reduced form) is needed.

NAME

re — RealTek 8139C+/8169/8169S/8110S PCI Ethernet adapter driver

SYNOPSIS

```
re* at pci? dev ? function ?
re* at cardbus? function ?
```

DESCRIPTION

The **re** driver provides support for various NICs based on the RealTek RTL8139C+, RTL8169, RTL8169S, and RTL8110S PCI/Cardbus Ethernet controllers, including the following:

- Alloy Computer Products EtherGOLD 1439E 10/100 (8139C+)
- Compaq Evo N1015v Integrated Ethernet (8139C+)
- Gigabyte 7N400 Pro2 Integrated Gigabit Ethernet (8110S)
- NETGEAR GA311 (8169S)
- PLANEX COMMUNICATIONS Inc. GN-1200TC (8169S)
- Xterasys XN-152 10/100/1000 NIC (8169)
- Corega CG-LAPCIGT Gigabit Ethernet (8169S)
- D-Link DGE-528T Gigabit Ethernet (8169S)
- US Robotics (3Com) USR997902 Gigabit Ethernet (8169S)
- Linksys EG1032 rev. 3 Gigabit Ethernet (8169S)

NICs based on the 8139C+ are capable of 10 and 100Mbps speeds over CAT5 cable. NICs based on the 8169, 8169S, and 8110S are capable of 10, 100, and 1000Mbps operation.

All NICs supported by the **re** driver have TCP/IP checksum offload and hardware VLAN tagging/insertion features, and use a descriptor-based DMA mechanism. They are also capable of TCP large send (TCP segmentation offload).

The 8139C+ is a single-chip solution combining both a 10/100 MAC and PHY, and its PHY is supported by `rlphy`(4). The 8169 is a 10/100/1000 MAC only, requiring a GMII or TBI external PHY and some 8169 based boards have Marvell 88E1000 PHY supported by `makphy`(4). The 8169S and 8110S are single-chip devices containing both a 10/100/1000 MAC and 10/100/1000 copper PHY, which is supported by `rgephy`(4). Standalone 10/100/1000 cards are available in both 32-bit PCI and 64-bit PCI models. The 8110S is designed for embedded LAN-on-motherboard applications.

The 8169, 8169S, and 8110S also support jumbo frames, which can be configured via the interface MTU setting. Selecting an MTU larger than 1500 bytes with the `ifconfig`(8) utility configures the adapter to receive and transmit jumbo frames.

The **re** driver supports the following media types:

autoselect Enable autoselection of the media type and options. The user can manually override the autoselected mode by adding media options to `rc.conf`(5).

10baseT/UTP Set 10Mbps operation. The `ifconfig`(8) **mediaopt** option can also be used to select either **full-duplex** or **half-duplex** modes.

100baseTX Set 100Mbps (Fast Ethernet) operation. The `ifconfig`(8) **mediaopt** option can also be used to select either **full-duplex** or **half-duplex** modes.

1000baseTX Set 1000baseTX operation over twisted pair. The RealTek GigE chips support 1000Mbps in **full-duplex** mode only.

The **re** driver supports the following media options:

full-duplex Force full duplex operation.

half-duplex Force half duplex operation.

For more information on configuring this device, see ifconfig(8).

DIAGNOSTICS

re%d: can't map i/o space A fatal initialization error has occurred.

re%d: can't map mem space A fatal initialization error has occurred.

re%d: couldn't map interrupt A fatal initialization error has occurred.

re%d: watchdog timeout The device has stopped responding to the network, or there is a problem with the network connection (cable).

SEE ALSO

arp(4), cardbus(4), mii(4), netintro(4), pci(4), rgephy(4), rlphy(4), ifconfig(8)

RealTek Semiconductor RTL8139C+, RTL8169, RTL8169S, and RTL8110S datasheets, http://www.realtek.com.tw.

HISTORY

The **re** device driver first appeared in FreeBSD 5.2 and was ported to NetBSD 2.0.

AUTHORS

The **re** driver was written by Bill Paul ⟨wpaul@windriver.com⟩.

BUGS

The Xterasys XN-152 32-bit PCI NIC, which uses the RTL8169 MAC and Marvell 88E1000 PHY, has a defect that causes DMA corruption if the board is plugged into a 64-bit PCI slot. The defect lies in the board design, not the chip itself: the PCI REQ64# and ACK64# lines should be pulled high, but they are not. The result is that the 8169 chip is tricked into performing 64-bit DMA transfers even though a 64-bit data path between the NIC and the bus does not actually exist.

Unfortunately, it is not possible to correct this problem in software, however it is possible to detect it. When the **re** driver is loaded, it will run a diagnostic routine designed to validate DMA operation by placing the chip in digital loopback mode and initiating a packet transmission. If the card functions properly, the transmitted data will be echoed back unmodified. If the echoed data is corrupt, the driver will print an error message on the console and abort the device attach. The user should ensure the NIC is installed in a 32-bit PCI slot to avoid this problem.

The RealTek 8169, 8169S, and 8110S chips appear to only be capable of transmitting jumbo frames up to 7.5K in size.

NAME

rf — DEC RX01 / RX02 floppy disk interface

SYNOPSIS

```
rfc0 at uba? csr 0177170     # RX01/RX02 controller
rf*  at rfc? drive?          # RX01/RX02 floppy disk drive
```

DESCRIPTION

The **rf** device provides access to DEC RX01 and RX02 floppy disk drives and clones thereof. These drives use preformatted 8" single sided, soft sectored media. The RX01 and RX02 drives use a geometry of 77 cylinders, one head and 26 sectors. Each sector contains 128 bytes in case of single density (RX01 and RX02 in RX01 mode) or 256 bytes in double density mode. As NetBSD is not able to handle non-512 byte media the driver translates this to a geometry of 50 cylinders, one head and 10 sectors in single density and 77 cylinders, one head and 13 sectors in double density mode. While the later matches the total number of sectors, the fake geometry in single density does not cover the last two physical sectors exact, but it is possible to access this sectors at 512 byte LBN 501. When a 512 byte block is written to LBN 501 the last 256 bytes are ignored. When this 512 byte block is read the last 256 bytes contain undefined data.

This driver supports three minor devices corresponding to the slices:

Slice	Description
a	Single density only mode.
b	Double density only mode.
c	Density autodetect.

As the RX01 and RX02 hardware is not able to support formatting a blank disk, this driver has no support for according IOCTLs. But there are clones from third party vendors that support formatting. Formatting a blank disk may be initiated by the following commands on the VAX chevron prompt:

Single density
d/p/w 20001E78 9
d/p/w 20001E7A 92

Double density
d/p/w 20001E78 109
d/p/w 20001E7A 92

FILES

`/dev/rf?[abc]`
`/dev/rrf?[abc]`

DIAGNOSTICS

rfc_attach: Error creating bus_dma map: %d
did not respond to INIT CMD Possible errors during `autoconf`(4). %d is the return code of `bus_dmamap_create`(9).

%s: did not respond to CMD %x An error occurred while the driver tried to send command %x to drive %s.

rfc_intr: Error while reading sector: %x
rfc_intr: Error while writing sector: %x
rfc_intr: Error while DMA: %x %x is status code from the controller error and status register.

rfc_intr: Error while loading bus_dma map: %d %d is return code of `bus_dmamap_load`(9).

%s: density error. A single density disk was opened in double density only mode or vice versa or the medium in the drive attached as %s was not readable at all.

SEE ALSO

dd(1), tar(1), intro(4), disklabel(5), disklabel(8), mknod(8), mount(8), newfs(8)

HISTORY

The rf driver appeared in NetBSD 2.0. It is a complete rewrite, not related to the old 4.2BSD **rx** driver.

AUTHORS

Jochen Kunz

BUGS

Writing of a disklabel(5) is not supported. The driver return always the internally fake disklabel.

NAME

rgephy — Realtek 8169S/8110S internal 10/100/1000 PHY driver

SYNOPSIS

rgephy* **at mii? phy ?**

DESCRIPTION

The **rgephy** driver supports the internal PHY found on Realtek 8169S/8110S Ethernet adapters.

SEE ALSO

ifmedia(4), intro(4), mii(4), re(4), ifconfig(8)

NAME

rl — RL11/ RL01 and RL02 disk interface

SYNOPSIS

```
rlc0 at uba? csr 0177440
rl0 at rlc0 drive 0
rl* at rlc? drive ?
```

DESCRIPTION

The **rl** driver is a typical block-device disk driver; block device I/O is described in physio(4).

The script MAKEDEV(8) should be used to create the special files; if a special file needs to be created by hand consult mknod(8).

FILES

`/dev/rl[0-7][a-h]` block files
`/dev/rrl[0-7][a-h]` raw files

DIAGNOSTICS

rl%d: operation incomplete The current command to the disk did not complete within the timeout period. This may be due to hardware failure or a heavily loaded UNIBUS.

rl%d: read data CRC The controller detected a CRC error on data read from the disk. Probably a bad disk pack.

rl%d: header CRC The controller detected a CRC error on header data read from the disk. Probably a bad disk pack.

rl%d: data late The controller was not able to transfer data over the bus fast enough to not overflow/underflow the internal FIFO, probably because a heavily loaded UNIBUS or mis-ordered UNIBUS devices.

rl%d: header not found The requested sector was not found before the timer expired. If this error is the only error then it may indicate a software bug.

rl%d: non-existent memory The controller tried to do DMA to/from a non-mapped address. This is a software bug.

rl%d: memory parity error The host memory data sent had a parity error. This is a hardware failure.

SEE ALSO

hp(4), uda(4), up(4), syslogd(8)

HISTORY

The **rl** driver has been around nearly forever.
A complete new **rl** driver showed up in NetBSD 1.5.

BUGS

Error handling is less than optimal.

Seeks should be interleaved between multiple disks.

NAME

rlphy — Realtek 8139/8201L Ethernet PHY driver

SYNOPSIS

rlphy* **at mii? phy ?**

DESCRIPTION

The **rlphy** driver supports the internal physical layer interface (PHY) found on Realtek Semiconductor RTL8139 based Ethernet adapters, as well as the RTL8201L Ethernet PHY chip.

SEE ALSO

ifmedia(4), intro(4), mii(4), rtk(4), ifconfig(8)

NAME

rmp — HP Remote Maintenance Protocol Family

SYNOPSIS

```
options RMP
#include <sys/types.h>
#include <sys/socket.h>
#include <netrmp/rmp.h>
#include <netrmp/rmp_var.h>

int
socket(AF_RMP, SOCK_RAW, proto);
```

DESCRIPTION

Hewlett-Packard's Remote Maintenance Protocol family is a collection of protocols layered atop IEEE 802.3. The current implementation of the RMP family provides protocol support only for the SOCK_RAW socket type. As a result, sendto(2) and recvfrom(2) must be used to send and receive RMP packets.

The format of an RMP packet is defined in the include file ⟨netrmp/rmp_var.h⟩. The RMP packet arrives encapsulated in an (HP extended) IEEE 802.2 packet. The IEEE 802.2 packet is preceded by the kernel address of an *ifnet struct* which is used to 'route' a packet out the same interface it arrived on. Outgoing packets are encapsulated in a standard IEEE 802.3 packet, while incoming packets have this information stripped away.

ADDRESSING

RMP (IEEE 802.3) addresses are 6 octets in length (48 bytes). Sockets in the Remote Maintenance Protocol family use the following addressing structure:

```
struct sockaddr_rmp {
        short           srmp_family;
        u_char          srmp_dhost[6];
};
```

PROTOCOLS

The RMP protocol family supported by the operating system currently consists of the Boot Protocol (*proto*=RMPPROTO_BOOT). Unfortunately, we have no documentation on the Remote Maintenance Protocol and only sketchy information about the Boot Protocol.

SEE ALSO

bind(2), recvfrom(2), sendto(2), socket(2), intro(4), rbootd(8)

Stuart Sechrest, *An Introductory 4.4BSD Interprocess Communication Tutorial*. (see /usr/share/doc/psd/20.ipctut)

Samuel J. Leffler, Robert S. Fabry, William N. Joy, Phil Lapsley, Steve Miller, and Chris Torek, *Advanced 4.4BSD IPC Tutorial*. (see /usr/share/doc/psd/21.ipc)

HISTORY

The **rmp** protocol interface currently under development.

BUGS

- The HP ROM uses IEEE 802.3 (as opposed to Ethernet) packets. While the kernel heuristically recognizes these packets, a more general mechanism for doing so should be provided.

- The HP ROM uses a multicast address when first trying to locate boot servers. While the Ethernet [sic] board is programmed to recognize this particular multicast address (9:0:9:0:0:4), a more general mechanism for doing so should be provided.
- The kernel supports only RAW sockets for the RMP protocol. This is either a bug or a feature, since the kernel is smaller at the price of greater complexity in the server.
- There is no support for `bind`(2)'ing an address in the RMP domain. Something like an `RMPADDR_ANY` should be provided to prevent more than one `rbootd`(8) server from running at the same time.

NAME

rnd — in kernel entropy collection and random number generation

SYNOPSIS

```
pseudo-device rnd
```

DESCRIPTION

The **rnd** pseudo-device uses event timing information collected from many devices, and mixes this into an entropy pool. This pool is stirred with a cryptographically strong hash function when data is extracted from the pool.

INTERNAL ENTROPY POOL MANAGEMENT

When a hardware event occurs (such as completion of a hard drive transfer or an interrupt from a network device) a timestamp is generated. This timestamp is compared to the previous timestamp recorded for the device, and the first, second, and third order differentials are calculated.

If any of these differentials is zero, no entropy is assumed to have been gathered. If all are non-zero, one bit is assumed. Next, data is mixed into the entropy pool using an LFSR (linear feedback shift register).

To extract data from the entropy pool, a cryptographically strong hash function is used. The output of this hash is mixed back into the pool using the LFSR, and then folded in half before being returned to the caller.

Mixing the actual hash into the pool causes the next extraction to return a different value, even if no timing events were added to the pool. Folding the data in half prevents the caller to derive the actual hash of the pool, preventing some attacks.

USER ACCESS

User code can obtain random values from the kernel in two ways.

Reading from `/dev/random` will only return values while sufficient entropy exists in the internal pool. When sufficient entropy does not exist, EAGAIN is returned for non-blocking reads, or the read will block for blocking reads.

Reading from `/dev/urandom` will return as many values as requested, even when the entropy pool is empty. This data is not as good as reading from `/dev/random` since when the pool is empty, data is still returned, degenerating to a pseudo-random generator.

Writing to either device will mix the data written into the pool using the LFSR as above, without modifying the entropy estimation for the pool.

RANDOM SOURCE STRUCTURE

Each source has a state structure which the kernel uses to hold the timing information and other state for that source.

```
typedef struct {
        char            name[16];
        uint32_t       last_time;
        uint32_t       last_delta;
        uint32_t       last_delta2;
        uint32_t       total;
        uint32_t       type;
       uint32_t       flags;
} rndsource_t;
```

This structure holds the internal representation of a device's timing state. The *name* field holes the device name, as known to the kernel. The *last_time* entry is the timestamp of the last time this device generated an event. It is for internal use only, and not in any specific representation. The *last_delta* and *last_delta2* fields hold the last first- and second-order deltas. The *total* field holds a count of how many bits this device has potentially generated. This is not the same as how many bits were used from it. The *type* field holds the device type.

Currently, these types are defined:

`RND_TYPE_DISK` The device is a physical hard drive.

`RND_TYPE_NET` The device is a network interface. By default, timing information is collected from this source type, but entropy is not estimated.

`RND_TYPE_TAPE` The device is a tape device.

`RND_TYPE_TTY` The device is a terminal, mouse, or other user input device.

`RND_TYPE_RNG` The device is a random number generator.

flags is a bitfield.

`RND_FLAG_NO_ESTIMATE` Do not assume any entropy is in the timing information.

`RND_FLAG_NO_COLLECT` Do not even add timing information to the pool.

IOCTL

Various `ioctl`(2) functions are available to control device behavior, gather statistics, and add data to the entropy pool. These are all defined in the ⟨`sys/rnd.h`⟩ file, along with the data types and constants.

`RNDGETENTCNT` (`uint32_t`) Return the current entropy count (in bits).

`RNDGETSRCNUM` (`rndstat_t`)

```
typedef struct {
        uint32_t       start;
        uint32_t       count;
        rndsource_t     source[RND_MAXSTATCOUNT];
} rndstat_t;
```

Return data for sources, starting at *start* and returning at most *count* sources.

The values returned are actual in-kernel snapshots of the entropy status for devices. Leaking the internal timing information will weaken security.

`RNDGETSRCNAME` (`rndstat_name_t`)

```
typedef struct {
        char            name[16];
        rndsource_t     source;
} rndstat_name_t;
```

Return the device state for a named device.

`RNDCTL` (`rndctl_t`)

```
typedef struct {
        char            name[16];
        uint32_t       type;
        uint32_t       flags;
        uint32_t       mask;
```

```
} rndctl_t;
```

Change bits in the device state information. If *type* is 0xff, only the device name stored in *name* is used. If it is any other value, all devices of type *type* are altered. This allows all network interfaces to be disabled for entropy collection with one call, for example. The *flags* and *mask* work together to change flag bits. The *mask* field specifies which bits in *flags* are to be set or cleared.

`RNDADDDATA` `(rnddata_t)`

```
typedef struct {
        uint32_t       len;
        uint32_t       entropy;
        u_char          data[RND_POOLWORDS * 4];
} rnddata_t;
```

FILES

`/dev/random` Returns "good" values only
`/dev/urandom` Always returns data, degenerates to a pseudo-random generator

SEE ALSO

`rndctl`(8), `rnd`(9)

HISTORY

The random device was first made available in NetBSD 1.3.

AUTHORS

This implementation was written by Michael Graff <explorer@flame.org> using ideas and algorithms gathered from many sources, including the driver written by Ted Ts'o.

NAME

route — kernel packet forwarding database

SYNOPSIS

```
#include <sys/socket.h>
#include <net/if.h>
#include <net/route.h>

int
socket(PF_ROUTE, SOCK_RAW, int family);
```

DESCRIPTION

UNIX provides some packet routing facilities. The kernel maintains a routing information database, which is used in selecting the appropriate network interface when transmitting packets.

A user process (or possibly multiple co-operating processes) maintains this database by sending messages over a special kind of socket. This supplants fixed size `ioctl`(2)'s used in earlier releases. Routing table changes may only be carried out by the super user.

The operating system may spontaneously emit routing messages in response to external events, such as receipt of a redirect, or failure to locate a suitable route for a request. The message types are described in greater detail below.

Routing database entries come in two flavors: for a specific host, or for all hosts on a generic subnetwork (as specified by a bit mask and value under the mask. The effect of wildcard or default route may be achieved by using a mask of all zeros, and there may be hierarchical routes.

When the system is booted and addresses are assigned to the network interfaces, each protocol family installs a routing table entry for each interface when it is ready for traffic. Normally the protocol specifies the route through each interface as a "direct" connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface is requested to address the packet to the gateway listed in the routing entry (i.e. the packet is forwarded).

When routing a packet, the kernel will attempt to find the most specific route matching the destination. (If there are two different mask and value-under-the-mask pairs that match, the more specific is the one with more bits in the mask. A route to a host is regarded as being supplied with a mask of as many ones as there are bits in the destination). If no entry is found, the destination is declared to be unreachable, and a routing−miss message is generated if there are any listeners on the routing control socket described below.

A wildcard routing entry is specified with a zero destination address value, and a mask of all zeroes. Wildcard routes will be used when the system fails to find other routes matching the destination. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

One opens the channel for passing routing control messages by using the socket call shown in the synopsis above:

The *family* parameter may be `AF_UNSPEC` which will provide routing information for all address families, or can be restricted to a specific address family by specifying which one is desired. There can be more than one routing socket open per system.

Messages are formed by a header followed by a small number of sockaddrs (now variable length particularly in the ISO case), interpreted by position, and delimited by the new length entry in the sockaddr. An example of a message with four addresses might be an ISO redirect: Destination, Netmask, Gateway, and Author of the redirect. The interpretation of which address are present is given by a bit mask within the header, and the sequence is least significant to most significant bit within the vector.

Any messages sent to the kernel are returned, and copies are sent to all interested listeners. The kernel will provide the process ID for the sender, and the sender may use an additional sequence field to distinguish between outstanding messages. However, message replies may be lost when kernel buffers are exhausted.

The kernel may reject certain messages, and will indicate this by filling in the *rtm_errno* field. The routing code returns EEXIST if requested to duplicate an existing entry, ESRCH if requested to delete a non-existent entry, or ENOBUFS if insufficient resources were available to install a new route. In the current implementation, all routing processes run locally, and the values for *rtm_errno* are available through the normal *errno* mechanism, even if the routing reply message is lost.

A process may avoid the expense of reading replies to its own messages by issuing a setsockopt(2) call indicating that the SO_USELOOPBACK option at the SOL_SOCKET level is to be turned off. A process may ignore all messages from the routing socket by doing a shutdown(2) system call for further input.

If a route is in use when it is deleted, the routing entry will be marked down and removed from the routing table, but the resources associated with it will not be reclaimed until all references to it are released. User processes can obtain information about the routing entry to a specific destination by using a RTM_GET message, or by reading the /dev/kmem device, or by calling sysctl(3).

The messages are:

```
#define RTM_ADD        0x1    /* Add Route */
#define RTM_DELETE     0x2    /* Delete Route */
#define RTM_CHANGE     0x3    /* Change Metrics, Flags, or Gateway */
#define RTM_GET        0x4    /* Report Information */
#define RTM_LOSING     0x5    /* Kernel Suspects Partitioning */
#define RTM_REDIRECT   0x6    /* Told to use different route */
#define RTM_MISS       0x7    /* Lookup failed on this address */
#define RTM_RESOLVE    0xb    /* request to resolve dst to LL addr */
#define RTM_NEWADDR    0xc    /* address being added to iface */
#define RTM_DELADDR    0xd    /* address being removed from iface */
#define RTM_OIFINFO    0xe    /* Old (pre-1.5) RTM_IFINFO message */
#define RTM_IFINFO     0xf    /* iface/link going up/down etc. */
#define RTM_IFANNOUNCE 0x10   /* iface arrival/departure */
```

A message header consists of one of the following:

```
struct rt_msghdr {
    u_short rtm_msglen;        /* to skip over non-understood messages */
    u_char  rtm_version;       /* future binary compatibility */
    u_char  rtm_type;          /* message type */
    u_short rtm_index;         /* index for associated ifp */
    int     rtm_flags;         /* flags, incl kern & message, e.g. DONE */
    int     rtm_addrs;         /* bitmask identifying sockaddrs in msg */
    pid_t   rtm_pid;           /* identify sender */
    int     rtm_seq;           /* for sender to identify action */
    int     rtm_errno;         /* why failed */
    int     rtm_use;           /* from rtentry */
    u_long  rtm_inits;         /* which metrics we are initializing */
    struct  rt_metrics rtm_rmx;       /* metrics themselves */
};

struct if_msghdr {
    u_short ifm_msglen;        /* to skip over non-understood messages */
    u_char  ifm_version;       /* future binary compatibility */
```

```
    u_char  ifm_type;          /* message type */
    int     ifm_addrs;         /* like rtm_addrs */
    int     ifm_flags;         /* value of if_flags */
    u_short ifm_index;         /* index for associated ifp */
    struct  if_data ifm_data;  /* statistics and other data about if */
};

struct ifa_msghdr {
    u_short ifam_msglen;       /* to skip over non-understood messages */
    u_char  ifam_version;      /* future binary compatibility */
    u_char  ifam_type;         /* message type */
    int     ifam_addrs;        /* like rtm_addrs */
    int     ifam_flags;        /* value of ifa_flags */
    u_short ifam_index;        /* index for associated ifp */
    int     ifam_metric;       /* value of ifa_metric */
};

struct if_announcemsghdr {
    u_short ifan_msglen;       /* to skip over non-understood messages */
    u_char  ifan_version;      /* future binary compatibility */
    u_char  ifan_type;         /* message type */
    u_short ifan_index;        /* index for associated ifp */
    char    ifan_name[IFNAMSIZ]; /* if name, e.g. "en0" */
    u_short ifan_what;         /* what type of announcement */
};
```

The RTM_IFINFO message uses a *if_msghdr* header, the RTM_NEWADDR and RTM_DELADDR messages use a *ifa_msghdr* header, the RTM_IFANNOUNCE message uses a *if_announcemsghdr* header, and all other messages use the *rt_msghdr* header.

The metrics structure is:

```
struct rt_metrics {
    u_long rmx_locks;          /* Kernel must leave these values alone */
    u_long rmx_mtu;            /* MTU for this path */
    u_long rmx_hopcount;       /* max hops expected */
    u_long rmx_expire;         /* lifetime for route, e.g. redirect */
    u_long rmx_recvpipe;       /* inbound delay-bandwidth product */
    u_long rmx_sendpipe;       /* outbound delay-bandwidth product */
    u_long rmx_ssthresh;       /* outbound gateway buffer limit */
    u_long rmx_rtt;            /* estimated round trip time */
    u_long rmx_rttvar;         /* estimated rtt variance */
    u_long rmx_pksent;         /* packets sent using this route */
};
```

Flags include the values:

```
#define RTF_UP        0x1      /* route usable */
#define RTF_GATEWAY   0x2      /* destination is a gateway */
#define RTF_HOST      0x4      /* host entry (net otherwise) */
#define RTF_REJECT    0x8      /* host or net unreachable */
#define RTF_DYNAMIC   0x10     /* created dynamically (by redirect) */
#define RTF_MODIFIED  0x20     /* modified dynamically (by redirect) */
#define RTF_DONE      0x40     /* message confirmed */
```

```
#define RTF_MASK      0x80     /* subnet mask present */
#define RTF_CLONING   0x100    /* generate new routes on use */
#define RTF_XRESOLVE  0x200    /* external daemon resolves name */
#define RTF_LLINFO    0x400    /* generated by ARP or ESIS */
#define RTF_STATIC    0x800    /* manually added */
#define RTF_BLACKHOLE 0x1000   /* just discard pkts (during updates) */
#define RTF_CLONED    0x2000   /* this is a cloned route */
#define RTF_PROTO2    0x4000   /* protocol specific routing flag */
#define RTF_PROTO1    0x8000   /* protocol specific routing flag */
```

Specifiers for metric values in rmx_locks and rtm_inits are:

```
#define RTV_MTU       0x1    /* init or lock _mtu */
#define RTV_HOPCOUNT  0x2    /* init or lock _hopcount */
#define RTV_EXPIRE    0x4    /* init or lock _expire */
#define RTV_RPIPE     0x8    /* init or lock _recvpipe */
#define RTV_SPIPE     0x10   /* init or lock _sendpipe */
#define RTV_SSTHRESH  0x20   /* init or lock _ssthresh */
#define RTV_RTT       0x40   /* init or lock _rtt */
#define RTV_RTTVAR    0x80   /* init or lock _rttvar */
```

Specifiers for which addresses are present in the messages are:

```
#define RTA_DST       0x1    /* destination sockaddr present */
#define RTA_GATEWAY   0x2    /* gateway sockaddr present */
#define RTA_NETMASK   0x4    /* netmask sockaddr present */
#define RTA_GENMASK   0x8    /* cloning mask sockaddr present */
#define RTA_IFP       0x10   /* interface name sockaddr present */
#define RTA_IFA       0x20   /* interface addr sockaddr present */
#define RTA_AUTHOR    0x40   /* sockaddr for author of redirect */
#define RTA_BRD       0x80   /* for NEWADDR, broadcast or p-p dest addr */
```

SEE ALSO

socket(2), sysctl(3)

NAME

rs5c372rtc — RICOH RS5C372A and RS5C372B real-time clock

SYNOPSIS

```
rs5c372rtc at iic? addr 0x32
```

DESCRIPTION

The **rs5c372rtc** driver provides support for the RICOH RS5C372A and RS5C372B real-time clock chips.

Access methods to retrieve and set date and time are provided through the *TODR* interface defined in todr(9).

SEE ALSO

todr(9)

HISTORY

The **rs5c372rtc** device appeared in NetBSD 4.0.

BUGS

The driver only supports clock function.

NAME

rt — AIMS Lab Radiotrack FM radio device driver

SYNOPSIS

```
rt0    at isa? port 0x20c
rt1    at isa? port 0x284
rt2    at isa? port 0x30c
rt3    at isa? port 0x384
radio* at rt?
```

DESCRIPTION

The **rt** driver provides support for the AIMS Lab Radiotrack FM radio tuners and compatible RadioReveal RA300 and SoundForte RadioX SF16-FMI FM radio tuners.

The Radiotrack is a stereo FM tuner that allows to tune in the range 87.5 - 108.0 MHz, report signal status on the current frequency, and force audio output to mono.

The Radiotrack cards take only one I/O port. The I/O port is set by the driver to the value specified in the configuration file and must be either 0x20c or 0x30c.

SEE ALSO

isa(4), radio(4)

HISTORY

The **rt** device driver appeared in OpenBSD 3.0 and NetBSD 1.6.

AUTHORS

The **rt** driver was written by Vladimir Popov and Maxim Tsyplakov. The man page was written by Vladimir Popov.

BUGS

Support for the SF16-FMI cards is rather ugly, volume control is not working and the driver can not correctly determine signal state.

NAME

rtc — Atari Real Time Clock

SYNOPSIS

```
clock0 at mainbus0
```

DESCRIPTION

The **rtc** driver supports reading from and writing to the Atari real time clock (RTC). When reading from the RTC, the format string of the output, as described in the `strftime`(3) manual page, is:

```
''%Y%m%d%H%M.%S''
```

The same format is used to set the RTC to the current date. Note that the kernel expects the RTC to run in UTC.

FILES

`/dev/rtc`

EXAMPLES

date -u +%Y%m%d%H%M.%S > /dev/rtc

SEE ALSO

`date`(1), `strftime`(3)

NAME

rtc — DS17485 support

SYNOPSIS

```
rtc0 at mainbus0 addr 0x7ff00000
```

DESCRIPTION

The **rtc** provides support for the real time clock DS17485.

SEE ALSO

mainbus(4)

HISTORY

The **rtc** driver appeared in NetBSD 2.0.

NAME

rtc — hp300 real-time clock

SYNOPSIS

rtc* **at intio?**

DESCRIPTION

The **rtc** driver provides support for the hp300 real-time clock (RTC). It is a mandatory device and must be compiled into the kernel.

Note that the kernel expects the real-time clock to run in UTC unless otherwise specified. See options(4).

SEE ALSO

intio(4), intro(4), options(4)

NAME

rtfps — multiplexing serial communications interface

SYNOPSIS

```
rtfps0 at isa? port 0x1230 irq 10
com2 at rtfps0 slave 0
com3 at rtfps0 slave 1
com4 at rtfps0 slave 2
com5 at rtfps0 slave 3
```

DESCRIPTION

The **rtfps** driver provides support for IBM RT PC boards that multiplex together up to four EIA RS-232C (CCITT V.28) or RS-422A communications interfaces.

Each **rtfps** device is the master device for up to four **com** devices. The kernel configuration specifies these **com** devices as slave devices of the **rtfps** device, as shown in the synopsis. The port specification for the **rtfps** device is used to compute the base addresses for the **com** subdevices.

FILES

`/dev/tty0?`

SEE ALSO

com(4)

HISTORY

The **rtfps** driver was written by Charles Hannum, based on the **ast** driver.

BUGS

The **rtfps** driver is unlikely to work on non-EISA and non-PCI machines. The ISA bus only asserts 10 I/O address lines, and this is not enough.

Even on EISA and PCI machines, some address conflicts have been observed. On one machine, the second port always conflicted with something (though it's not clear what) and caused strange results. Disabling the second port in the kernel config allowed the other three ports to function correctly.

NAME

rtii — AIMS Lab Radiotrack II FM radio device driver

SYNOPSIS

```
option RADIO_TEA5759

rtii0  at isa? port 0x20c
rtii1  at isa? port 0x30c
radio* at rtii0
```

DESCRIPTION

The **rtii** driver provides support for the AIMS Lab Radiotrack II FM radio tuners.

The Radiotrack II is a stereo FM tuner that allows to tune in the range 87.5 - 108.0 MHz, report signal status on the current frequency, force audio output to mono, perform hardware signal search, and has an internal AFC.

The Radiotrack II cards take only one I/O port. The I/O port is set by the driver to the value specified in the configuration file and must be either 0x20c or 0x30c.

Philips Semiconductors released two almost identical chips TEA5757 and TEA5759. The TEA5757 is used in FM-standards in which the local oscillator frequency is above the radio frequency (e.g. European and American standards). The TEA5759 is the version in which the oscillator frequency is below the radio frequency (e.g. Japanese standards). The option *RADIO_TEA5759* changes the algorithm of frequency calculation for the TEA5757 based cards to conform with the Japanese standards.

SEE ALSO

isa(4), radio(4)

HISTORY

The **rtii** device driver appeared in OpenBSD 3.0 and NetBSD 1.6.

AUTHORS

The **rtii** driver was written by Vladimir Popov and Maxim Tsyplakov. The man page was written by Vladimir Popov.

NAME

rtk — Ethernet driver for Realtek 8129/8139/8100 based Ethernet boards

SYNOPSIS

```
rtk* at cardbus? function ?
rtk* at pci? dev ? function ?
```

Configuration of PHYs are necessary. See mii(4).

DESCRIPTION

The **rtk** device driver supports network adapters based on the Realtek 8129/8139/8100 chips.

MEDIA SELECTION

Media selection is done using ifconfig(8) using the standard ifmedia(4) mechanism. Refer to those manual pages for more information.

SEE ALSO

cardbus(4), ifmedia(4), mii(4), netintro(4), pci(4), ifconfig(8)

NAME

rtw — Realtek RTL8180L IEEE 802.11b wireless network driver

SYNOPSIS

```
rtw* at cardbus? function ?
rtw* at pci? dev ? function ?
```

DESCRIPTION

The **rtw** driver supports PCI/CardBus 802.11b wireless adapters based on the Realtek RTL8180L.

A variety of radio tranceivers can be found in these devices, including the Philips SA2400A, Maxim MAX2820, and GCT GRF5101, though not all of them are currently supported.

These are the modes the **rtw** driver can operate in:

BSS mode — Also known as *infrastructure* mode, this is used when associating with an access point, through which all traffic passes. This mode is the default.

IBSS mode — Also known as *IEEE ad-hoc* mode or *peer-to-peer* mode. This is the standardized method of operating without an access point. Stations associate with a service set. However, actual connections between stations are peer-to-peer.

Host AP — In this mode the driver acts as an access point (base station) for other cards.

monitor mode — In this mode the driver is able to receive packets without associating with an access point. This disables the internal receive filter and enables the card to capture packets from networks which it wouldn't normally have access to, or to scan for access points.

rtw supports software WEP. Wired Equivalent Privacy (WEP) is the de facto encryption standard for wireless networks. It can be typically configured in one of three modes: no encryption; 40-bit encryption; or 104-bit encryption. Unfortunately, due to serious weaknesses in WEP protocol it is strongly recommended that it not be used as the sole mechanism to secure wireless communication. WEP is not enabled by default.

CONFIGURATION

The **rtw** driver can be configured at runtime with `ifconfig`(8) or on boot with `ifconfig.if`(5) using the following parameters:

bssid *bssid*
Set the desired BSSID.

-bssid
Unset the desired BSSID. The interface will automatically select a BSSID in this mode, which is the default.

chan *n*
Set the channel (radio frequency) to be used by the driver based on the given channel ID *n*.

-chan
Unset the desired channel to be used by the driver. The driver will automatically select a channel in this mode, which is the default.

media *media*
The **rtw** driver supports the following *media* types:

autoselect Enable autoselection of the media type and options.
DS1 Set 802.11b DS 1Mbps operation.

DS2 Set 802.11b DS 2Mbps operation.
DS5 Set 802.11b DS 5.5Mbps operation.
DS11 Set 802.11b DS 11Mbps operation.

mediaopt *opts*
The **rtw** driver supports the following media options:

hostap Select Host AP operation.
ibss Select IBSS operation.
monitor Select monitor mode.

-mediaopt *opts*
Disable the specified media options on the driver and return it to the default mode of operation (BSS).

ssid *id*
Set the network ID. The *id* can either be any text string up to 32 characters in length, or a series of hexadecimal digits up to 64 digits. An empty *id* string allows the interface to connect to any available access points. By default the **rtw** driver uses an empty string. Note that network ID is synonymous with Extended Service Set ID (ESSID).

nwkey *key*
Enable WEP encryption using the specified *key*. The *key* can either be a string, a series of hexadecimal digits (preceded by '0x'), or a set of keys of the form "n:k1,k2,k3,k4", where 'n' specifies which of the keys will be used for transmitted packets, and the four keys, "k1" through "k4", are configured as WEP keys. If a set of keys is specified, a comma (',') within the key must be escaped with a backslash. Note that if multiple keys are used, their order must be the same within the network. **rtw** is capable of using both 40-bit (5 characters or 10 hexadecimal digits) or 104-bit (13 characters or 26 hexadecimal digits) keys.

-nwkey
Disable WEP encryption. This is the default mode of operation.

nwkey persist
Enable WEP encryption with the persistent key stored in the network card.

HARDWARE

The following adapters should work:

Card	*Bus*
Belkin F5D6020 V3	CardBus
Buffalo WLI-CB-B11	CardBus
Corega CG-WLCB11V3	CardBus
D-Link DWL-610	CardBus
Level-One WPC-0101	CardBus
Linksys WPC11 v4	CardBus
Netgear MA521	CardBus
Ovislink AirLive WL-1120PCM	CardBus
Planet WL-3553	CardBus
TrendNET TEW-266PC	CardBus
VCTnet PC-11B1	CardBus

EXAMPLES

The following ifconfig.if(5) example creates a host-based access point on boot:

```
inet 192.168.1.1 255.255.255.0 NONE media autoselect \
        mediaopt hostap ssid my_net chan 11
```

Configure rtw0 for WEP, using hex key “0x1deadbeef1”:

```
# ifconfig rtw0 nwkey 0x1deadbeef1
```

Return rtw0 to its default settings:

```
# ifconfig rtw0 -bssid -chan media autoselect \
        ssid "" -nwkey
```

Join an existing BSS network, “my_net”:

```
# ifconfig rtw0 192.168.1.1 netmask 0xffffff00 ssid my_net
```

SEE ALSO

arp(4), cardbus(4), ifmedia(4), intro(4), netintro(4), pci(4), ifconfig.if(5), ifconfig(8)

Realtek, http://www.realtek.com.tw.

HISTORY

The **rtw** device driver first appeared in NetBSD 3.0 and then in OpenBSD 3.7.

AUTHORS

The **rtw** driver was written by David Young ⟨dyoung@NetBSD.org⟩ and ported to OpenBSD by Jonathan Gray ⟨jsg@openbsd.org⟩, who wrote this man page.

BUGS

Only the Philips SA2400A and Maxim MAX2820 RF transceivers are known to work. Devices incorporating a GCT RF transceiver are not supported due to a lack of documentation from GCT.

While PCI devices will attach most of them are not able to transmit.

NAME

rum — Ralink Technology USB IEEE 802.11a/b/g wireless network device

SYNOPSIS

rum* **at uhub? port ?**

DESCRIPTION

The **rum** driver supports USB 2.0 wireless adapters based on the Ralink RT2501USB and RT2601USB chipsets.

The RT2501USB chipset is the second generation of 802.11a/b/g adapters from Ralink. It consists of two integrated chips, an RT2571W MAC/BBP and an RT2528 or RT5226 radio transceiver.

The RT2601USB chipset consists of two integrated chips, an RT2671 MAC/BBP and an RT2527 or RT5225 radio transceiver. This chipset uses the MIMO (multiple-input multiple-output) technology with multiple antennas to extend the operating range of the adapter and to achieve higher throughput. MIMO is the basis of the forthcoming IEEE 802.11n standard.

These are the modes the **rum** driver can operate in:

BSS mode — Also known as *infrastructure* mode, this is used when associating with an access point, through which all traffic passes. This mode is the default.

IBSS mode — Also known as *IEEE ad-hoc* mode or *peer-to-peer* mode. This is the standardized method of operating without an access point. Stations associate with a service set. However, actual connections between stations are peer-to-peer.

Host AP — In this mode the driver acts as an access point (base station) for other cards.

monitor mode — In this mode the driver is able to receive packets without associating with an access point. This disables the internal receive filter and enables the card to capture packets from networks which it wouldn't normally have access to, or to scan for access points.

rum supports software WEP. Wired Equivalent Privacy (WEP) is the de facto encryption standard for wireless networks. It can be typically configured in one of three modes: no encryption; 40-bit encryption; or 104-bit encryption. Unfortunately, due to serious weaknesses in WEP protocol it is strongly recommended that it not be used as the sole mechanism to secure wireless communication. WEP is not enabled by default.

CONFIGURATION

The **rum** driver can be configured at runtime with `ifconfig`(8) or on boot with `ifconfig.if`(5) using the following parameters:

bssid *bssid*
Set the desired BSSID.

-bssid
Unset the desired BSSID. The interface will automatically select a BSSID in this mode, which is the default.

chan *n*
Set the channel (radio frequency) to be used by the driver based on the given channel ID *n*.

-chan
Unset the desired channel to be used by the driver. The driver will automatically select a channel in this mode, which is the default.

media *media*
The **rum** driver supports the following *media* types:

autoselect	Enable autoselection of the media type and options.
DS1	Set 802.11b DS 1Mbps operation.
DS2	Set 802.11b DS 2Mbps operation.
DS5	Set 802.11b DS 5.5Mbps operation.
DS11	Set 802.11b DS 11Mbps operation.
OFDM6	Set 802.11a/g OFDM 6Mbps operation.
OFDM9	Set 802.11a/g OFDM 9Mbps operation.
OFDM12	Set 802.11a/g OFDM 12Mbps operation.
OFDM18	Set 802.11a/g OFDM 18Mbps operation.
OFDM24	Set 802.11a/g OFDM 24Mbps operation.
OFDM36	Set 802.11a/g OFDM 36Mbps operation.
OFDM48	Set 802.11a/g OFDM 48Mbps operation.
OFDM54	Set 802.11a/g OFDM 54Mbps operation.

mediaopt *opts*
The **rum** driver supports the following media options:

hostap	Select Host AP operation.
ibss	Select IBSS operation.
monitor	Select monitor mode.

-mediaopt *opts*
Disable the specified media options on the driver and return it to the default mode of operation (BSS).

mode *mode*
The **rum** driver supports the following modes:

11a	Force 802.11a operation.
11b	Force 802.11b operation.
11g	Force 802.11g operation.

nwid *id*
Set the network ID. The *id* can either be any text string up to 32 characters in length, or a series of hexadecimal digits up to 64 digits. An empty *id* string allows the interface to connect to any available access points. By default the **rum** driver uses an empty string. Note that network ID is synonymous with Extended Service Set ID (ESSID).

nwkey *key*
Enable WEP encryption using the specified *key*. The *key* can either be a string, a series of hexadecimal digits (preceded by '0x'), or a set of keys of the form "n:k1,k2,k3,k4", where 'n' specifies which of the keys will be used for transmitted packets, and the four keys, "k1" through "k4", are configured as WEP keys. If a set of keys is specified, a comma (',') within the key must be escaped with a backslash. Note that if multiple keys are used, their order must be the same within the network. **rum** is capable of using both 40-bit (5 characters or 10 hexadecimal digits) or 104-bit (13 characters or 26 hexadecimal digits) keys.

-nwkey
Disable WEP encryption. This is the default mode of operation.

FILES

The following firmware file is loaded when an interface is brought up:

/libdata/firmware/rum/rum-rt2573
See `firmload`(9) for how to change this.

HARDWARE

The following adapters should work:

Airlink101 AWLL5025
ASUS WL-167g ver 2
Belkin F5D7050 ver 3
Belkin F5D9050 ver 3
CNet CWD-854 ver F
Conceptronic C54RU ver 2
D-Link DWL-G122 rev C1
D-Link WUA-1340
Edimax EW-7318USG
Gigabyte GN-WB01GS
Hawking HWUG1
LevelOne WNC-0301USB
Linksys WUSB54G rev C
Planex GW-USMM
Senao NUB-3701
Sitecom WL-113 ver 2
Sitecom WL-172
TP-LINK TL-WN321G

EXAMPLES

The following `ifconfig.if`(5) example configures rum0 to join whatever network is available on boot, using WEP key “0x1deadbeef1”, channel 11:

```
inet 192.168.1.1 netmask 255.255.255.0 nwkey 0x1deadbeef1 chan 11
```

The following `ifconfig.if`(5) example creates a host-based access point on boot:

```
inet 192.168.1.1 netmask 255.255.255.0 media autoselect \
        mediaopt hostap nwid my_net chan 11
```

Configure rum0 for WEP, using hex key “0x1deadbeef1”:

```
# ifconfig rum0 nwkey 0x1deadbeef1
```

Return rum0 to its default settings:

```
# ifconfig rum0 -bssid -chan media autoselect \
        nwid "" -nwkey
```

Join an existing BSS network, “my_net”:

```
# ifconfig rum0 192.168.1.1 netmask 0xffffff00 nwid my_net
```

DIAGNOSTICS

rum%d: failed loadfirmware of file %s For some reason, the driver was unable to read the microcode file from the filesystem. The file might be missing or corrupted.

rum%d: could not load 8051 microcode An error occurred while attempting to upload the microcode to the onboard 8051 microcontroller unit.

rum%d: device timeout A frame dispatched to the hardware for transmission did not complete in time. The driver will reset the hardware. This should not happen.

SEE ALSO

`arp`(4), `ifmedia`(4), `netintro`(4), `usb`(4), `ifconfig.if`(5), `hostapd`(8), `ifconfig`(8), `firmload`(9)

Ralink Technology: `http://www.ralinktech.com`

HISTORY

The **`rum`** driver first appeared in NetBSD 4.0 and OpenBSD 4.0.

AUTHORS

The **`rum`** driver was written by Niall O'Higgins ⟨niallo@openbsd.org⟩ and Damien Bergamini ⟨damien@openbsd.org⟩.

CAVEATS

The **`rum`** driver supports automatic control of the transmit speed in BSS mode only. Therefore the use of a **`rum`** adapter in Host AP mode is discouraged.

NAME

sab, **sabtty** — Infineon 82532 Enhanced Serial Communication Controller (ESCC2)

SYNOPSIS

```
sab*    at ebus?
sabtty* at sab? channel ?
```

DESCRIPTION

The **sab** driver provides TTY support for the Infineon (previously Siemens) 82532 serial communications interface found in Sun Ultra 5/10 workstations.

Input and output speeds for each line may be set to any baud rate in the following list: 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 76800, 115200, 153600, 230400, 307200, 460800, 614400, 921600.

FILES

`/dev/ttyh[01]`
serial ports

DIAGNOSTICS

sab*: ring overflow
The software input "ring" has overflowed. This usually means input flow-control is not configured correctly (i.e. incorrect cable wiring).

SEE ALSO

`tty`(4), `zstty`(4)

HISTORY

The **sabtty** driver first appeared in OpenBSD 3.1 and was then ported to NetBSD 2.0.

AUTHORS

Jason L. Wright

NAME

sableio — AlphaServer 2100 (Sable) STD I/O module

SYNOPSIS

sableio* at ttwopci?

DESCRIPTION

The **sableio** driver provides support for STD I/O module found on the AlphaServer 2100. The module supports regular ISA peripherals and their regular ISA I/O addresses. The wiring of interrupts is peculiar on the AlphaServer 2100.

The following devices are supported by the **sableio** driver:

com	serial communications interface
fdc	NEC 765 floppy disk controller driver
lpt	Parallel port driver
pckbc	PC keyboard controller driver

SEE ALSO

com(4), fdc(4), intro(4), lpt(4), pckbc(4), ttwopci(4)

NAME

satalink — Silicon Image SATALink disk controller driver

SYNOPSIS

satalink* **at pci? dev ? function ? flags 0x0000**

DESCRIPTION

The **satalink** driver supports the Silicon Image SATALink 3112 2-port and 3114 4-port Serial ATA controllers, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **satalink** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the SATA controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

NAME

sb — SoundBlaster family (and compatible) audio device driver

SYNOPSIS

```
sb0    at isa? port 0x220 irq 5 drq 1 drq2 5
sb1    at isa? port 0x240 irq 7 drq 1 flags 1
sb*    at isapnp?
sb*    at pnpbios? index ?
audio* at audiobus?
midi*  at sb?
mpu*   at sb?
opl*   at sb?
```

DESCRIPTION

The **sb** driver provides support for the SoundBlaster, SoundBlaster Pro, SoundBlaster 16, Jazz 16, SoundBlaster AWE 32, SoundBlaster AWE 64, and hardware register-level compatible audio cards.

The SoundBlaster series are half-duplex cards, capable of 8- and 16-bit audio sample recording and playback at rates up to 44.1kHz (depending on the particular model).

The base I/O port address is usually jumper-selected to either 0x220 or 0x240 (newer cards may provide software configuration, but this driver does not directly support them--you must configure the card for its I/O addresses with other software). The SoundBlaster takes 16 I/O ports. For the SoundBlaster and SoundBlaster Pro, the IRQ and DRQ channels are jumper-selected. For the SoundBlaster 16, the IRQ and DRQ channels are set by this driver to the values specified in the config file. The IRQ must be selected from the set {5,7,9,10}.

The configuration file must use 1 **flags** specification to enable the Jazz16 support. This is to avoid potential conflicts with other devices when probing the Jazz 16 because it requires use of extra I/O ports not in the base port range.

With a SoundBlaster 16 card the device is full duplex, but it can only sensibly handle a precision of 8 bits. It does so by extending the output 8 bit samples to 16 bits and using the 8 bit DMA channel for input and the 16 bit channel for output.

The joystick interface (if enabled by a jumper) is handled by the joy(4) driver, and the optional SCSI CD-ROM interface is handled by the aic(4) driver.

SoundBlaster 16 cards have MPU401 emulation and can use the mpu attachment, older cards have a different way to generate MIDI and has a midi device attached directly to the **sb**.

SEE ALSO

aic(4), audio(4), isa(4), isapnp(4), joy(4), midi(4), mpu(4), opl(4), pnpbios(4)

HISTORY

The **sb** device driver appeared in NetBSD 1.0.

BUGS

Non-SCSI CD-ROM interfaces are not supported.

The MIDI interface on the SB hardware is braindead, and the driver needs to busy wait while writing MIDI data. This will consume a lot of system time.

NAME

sbp — Serial Bus Protocol 2 (SBP-2) Mass Storage Devices driver

SYNOPSIS

sbp* **at ieee1394if? euihi ? euilo ?**

DESCRIPTION

The **sbp** driver provides support for SBP-2 devices that attach to the IEEE1394 port. It should work with all SBP-2 devices which the scsi(4) layer supports, for example, HDDs, CDROM drives, and DVD drives.

Some users familiar with umass(4) might wonder why the device is not detached at the scsi(4) layer when the device is unplugged. It is detached only if the device has not been plugged again during several bus resets. This is for preventing to detach an active file system even when the device cannot be probed correctly for some reason after a bus reset or when the device is temporary disconnected because the user changes the bus topology. If you want to force to detach the device, run **fwctl -r** several times.

SEE ALSO

ieee1394if(4), scsi(4), fwctl(8), scsictl(8), sysctl(8)

AUTHORS

The **sbp** driver was written by Katsushi Kobayashi and Hidetoshi Shimokawa.

This manual page was written by Katsushi Kobayashi. It was added to NetBSD 4.0 by KIYOHARA Takashi.

NAME

SBus — introduction to machine-independent SBus bus support and drivers

SYNOPSIS

```
sbus* at mainbus?
sbus* at iommu?
sbus* at xbox?
```

These
SBus attachments are specific to the NetBSD/sparc and NetBSD/sparc64 ports.

DESCRIPTION

SBus is a I/O interconnect bus mostly found in SPARC workstations and small to medium server class systems. It supports both on-board peripherals and extension boards. The **SBus** specifications define the bus protocol as well as the electrical and mechanical properties of the extension slots.

HARDWARE

NetBSD includes machine-independent **SBus** drivers, sorted by device type and driver name:

SCSI interfaces

esp	NCR53c94 and compatible SCSI interfaces.
isp	Qlogic SCSI interfaces.

Network interfaces

le	Lance 7990 series Ethernet interfaces.
hme	“Happy Meal” Ethernet interfaces.
en	Midway-based Efficient Networks Inc. and Adaptec ATM interfaces.
be	“Big Mac” Ethernet board.
qe	Quad Ethernet Controller board.

Bridges

xbox	an **Sbus** expansion box.

Graphics devices

bwtwo	framebuffer device.
cgthree	framebuffer device.
cgsix	framebuffer device.
pnozz	framebuffer device.
tcx	framebuffer device.
zx	framebuffer device.

Audio devices

audiocs	CS4231 codec.

Serial interfaces

magma Magma Serial/Parallel combo device.

SEE ALSO

audiocs(4), be(4), bwtwo(4), cgsix(4), cgthree(4), en(4), esp(4), hme(4), intro(4), isp(4), le(4), magma(4), pnozz(4), qe(4), tcx(4), xbox(4), zx(4)

HISTORY

The machine-independent **SBus** subsystem appeared in NetBSD 1.3.

NAME

sc — Sun Sun-2 SCSI bus host adaptor driver

SYNOPSIS

sun2

```
sc0 at mbmem0 addr 0x80000 ipl 2
sc1 at mbmem0 addr 0x84000 ipl 2
```

sun2 and sun4

```
sc0 at vme0 addr 0x200000 irq 2 vec 0x40
```

DESCRIPTION

The **sc** driver provides support for the Sun Microsystems "Sun-2" SCSI Bus Controller chipset found on various VME boards (Sun part #s 501-1045, 501-1138, 501-1149, and 501-1167) and on the "Sun-2 SCSI/Serial" (Sun part # 501-1006) Multibus board.

All versions of this driver can be configured with a *flags* directive in the `config`(1) file. The values are bits in a bitfield, and are interpreted as follows:

0x0ff	Set bit (1<<target) to disable SCSI parity checking
0x100	Set this bit to disable DMA interrupts (poll)
0x200	Set this bit to disable DMA entirely (use PIO)

For example: "flags 0x1ff" would disable DMA interrupts, and disable parity checking for targets 0-7. The "target" is the SCSI ID number of a particular device on a particular SCSI bus.

SEE ALSO

`cd`(4), `ch`(4), `intro`(4), `scsi`(4), `sd`(4), `st`(4)

AUTHORS

Matt Fredette ⟨fredette@NetBSD.org⟩,
David Jones,
Gordon Ross ⟨gwr@NetBSD.org⟩,
Adam Glass ⟨glass@NetBSD.org⟩,
Jason R. Thorpe ⟨thorpej@NetBSD.org⟩.

BUGS

This SCSI chipset is rumored to have bugs in its handling of SCSI parity, therefore it is recommended that you disable parity on all SCSI devices connected to this controller, and configure it with a 0x0ff value for its *flags* directive in the `config`(1) file.

This chipset has no support for raising the ATN signal, so there is no way to ever schedule a MSG_OUT phase on the bus. Currently, the driver will ultimately reset the bus if this phase is ever requested by the upper layer SCSI driver.

This chipset has no support for SCSI disconnect/reselect. This means that slow devices, such as tape drives, can hog, or "lock up" the SCSI bus.

This driver has not been tested in combination with non-SCSI devices behind Emulex or Adaptec bridges, which are common in Sun 2s and in Sun Shoebox-type configurations. These devices pre-date the SCSI-I spec, and might not behave the way the chipset code currently expects.

NAME

scc — Zilog 8530 Serial Communications Controller interface

SYNOPSIS

scc* at ioasic? offset ?

DESCRIPTION

The **scc** driver provides support for the Zilog 8530 Serial Communications Controller (SCC) via the IOASIC found on DECstation 5000 models in the /100, /20, and /240 series (supported by NetBSD/pmax).

Each SCC device has two serial ports. The DECstation 5000 model 20 provides one SCC device. Other models provide two, but one port of each device is dedicated to mouse and keyboard input, respectively.

Input and output for each line may set to one of following baud rates: 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, or 57600.

Speeds up to 230400 are supported by the chip and the motherboard, but speeds higher than 57600 do not work reliably without an external clock signal.

FILES

`/dev/ttya`
`/dev/ttyb`
`/dev/ttyc`
`/dev/ttyd`

The mapping from units to serial-hardware outlets is idiosyncratic. The even ports are wired serial connectors and the odd-numbered ports are reserved for mouse and keyboard.

On machines with one port like Personal DECstations, the single device is `ttya`.

On the 5000/1xx and 5000/2xx, the first serial port (default serial console) is `ttyc` and the second port is `ttya`.

SEE ALSO

`intro`(4), `ioasic`(4), `ttys`(5), `MAKEDEV`(8)

HISTORY

The **scc** driver first appeared in 4.4BSD.

The **scc** driver was also used for the IOASIC SCC found in DEC Alpha model 3000 TurboCHANNEL based systems; NetBSD/alpha has since been converted to use the machine-independent `zstty`(4).

BUGS

The IOASIC provides internal DMA channels that can be programmed to transfer up to 4096 bytes of data into, or out, of an SCC without further software intervention. This feature of the IOASIC is not yet supported.

The mapping from device-special files (major and minor number) to chip and port is arguably backwards. ULTRIX tries to hide the hardware mapping, but NetBSD does not. Users wanting to use ULTRIX compatible tty names `/dev/tty0` and `/dev/tty1` for the **scc** comm-port lines should make links or device-special files which match their hardware setup.

NAME

scsi, **scsibus** — Small Computer Systems Interface (SCSI) bus driver

SYNOPSIS

```
scsibus* at scsi?
atapibus* at atapi?
options SCSIDEBUG
options SCSIVERBOSE
```

DESCRIPTION

The **scsi** driver is the top, machine-independent layer of the two-layer software system that provides an interface for the implementation of drivers to control various SCSI or ATAPI bus devices, and to use different SCSI bus host adapters or EIDE controllers. SCSI bus is capable of supporting a wide variety of peripherals, including hard disks, removable disks, CD-ROMs, scanners, tape drives, and other miscellaneous high-speed devices.

The bottom layer is composed of the drivers for individual EIDE or SCSI bus controller chips (e.g. NCR 5380), accessed through various host bus interfaces, including, but not limited to PCI, ISA, Sbus, TURBOchannel, and NuBus. These individual devices are referred to as "host adaptors" in SCSI terminology, because they connect the SCSI bus to the host computer.

When NetBSD probes the SCSI busses, it "attaches" any devices it finds to the appropriate drivers.

- sd(4) hard disks
- cd(4) CD-ROM drives
- st(4) tape drives
- ch(4) media changers
- ss(4) scanners

If no specific driver matches the device, then **scsi** attaches the device to the uk(4) driver so that user level SCSI ioctl(2) calls may still be performed against the device. Currently, only sd(4), cd(4), st(4) and uk(4) can attach to an atapi bus.

Please see the intro(4) manual page to see which SCSI bus host adaptors are supported by NetBSD on your computer system.

KERNEL CONFIGURATION

The **scsi** software supports some NetBSD kernel config(1) options. They are:

SCSIDEBUG Compile in a wide variety of **printf**() statements that can be turned on by ioctl(2).

SCSIVERBOSE Enable additional and more descriptive error and status messages from the **scsi** software.

All devices and the SCSI busses support boot time allocation so that an upper number of devices and controllers does not need to be configured.

The devices are either *wired* so they appear at a particular device unit number or *counted* so that they appear as the next available unused unit number.

To configure a driver in the kernel without wiring down the device use a config line similar to

```
ch* at scsibus? target ? lun ?
```

to include the ch(4) changer driver.

To wire down a unit use a config line similar to

```
ch1 at scsibus0 target 4 lun 0
```

to assign changer 1 as the changer with SCSI ID 4, logical unit 0, on bus 0. Individual SCSI busses can be wired down to specific controllers with a config line similar to

```
scsibus0 at ahc0
```

which assigns SCSI bus 0 to the first unit using the ahc(4) driver.

When you have a mixture of wired down and counted devices then the counting begins with the first non-wired down unit for a particular type. That is, if you have a disk wired down as

```
sd1 at scsibus0 target 1 lun 0
```

then the first non-wired disk shall come on line as *sd2*.

IOCTLS

There are a number of ioctl(2) calls that work on any SCSI device. They are defined in sys/scsiio.h and can be applied against any SCSI device that permits them. For the tape, it must be applied against the control device. See the manual page for each device type for more information about how generic SCSI ioctl(2) calls may be applied to a specific device.

SCIOCRESET — Reset a SCSI device.

SCIOCDEBUG — Turn on debugging. All SCSI operations originating from this device's driver will be traced to the console, along with other information. Debugging is controlled by four bits, described in the header file. If no debugging is configured into the kernel, debugging will have no effect. SCSI debugging is controlled by the configuration option SCSIDEBUG.

SCIOCCOMMAND — Take a SCSI command and data from a user process and apply them to the SCSI device. Return all status information and return data to the process. The ioctl(2) call will return a successful status even if the device rejected the command. As all status is returned to the user, it is up to the user process to examine this information to decide the success of the command.

SCIOCREPROBE — Ask the system to probe the SCSI busses for any new devices. If it finds any, they will be attached to the appropriate drivers. The search can be narrowed to a specific bus, target or Logical Unit Number (LUN). The new device may or may not be related to the device on which the ioctl was performed.

SCIOCIDENTIFY — Ask the driver what its bus, target and LUN are.

SCIOCDECONFIG — Ask the device to disappear. This may not happen if the device is in use.

ADAPTERS

The system allows common device drivers to work through many different types of adapters. The adapters take requests from the upper layers and do all IO between the SCSI bus and the system. The maximum size of a transfer is governed by the adapter. Most adapters can transfer 64KB in a single operation, however many can transfer larger amounts.

TARGET MODE

Some adapters support *Target Mode* in which the system is capable of operating as a device, responding to operations initiated by another system. Target Mode will be supported for some host adapters, but is not yet complete for this version of the SCSI system.

DIAGNOSTICS

When the kernel is compiled with option `SCSIDEBUG`, the `SCIOCDEBUG` `ioctl`(2) can be used to enable various amounts of tracing information on any specific device. Devices not being traced will not produce trace information. The four bits that make up the debug level, each control certain types of debugging information.

`Bit 0` shows all SCSI bus operations including SCSI commands, error information and the first 48 bytes of any data transferred.

`Bit 1` shows routines called.

`Bit 2` shows information about what branches are taken and often some of the return values of functions.

`Bit 3` shows more detailed information including DMA scatter-gather logs.

SEE ALSO

`config`(1), `ioctl`(2), `ata`(4), `cd`(4), `ch`(4), `intro`(4), `sd`(4), `se`(4), `ss`(4), `st`(4), `uk`(4)

HISTORY

This **`scsi`** system appeared in MACH 2.5 at TRW.

This man page was originally written by Julian Elischer ⟨julian@freebsd.org⟩ for FreeBSD and extensively modified by Erik Fair ⟨fair@NetBSD.org⟩ for NetBSD.

BUGS

Not every device obeys the SCSI specification as faithfully as it should. As such devices are discovered by the NetBSD Project, their names are added to a *quirk list* compiled into the **`scsi`** driver along a list of flags indicating which particular bad behaviors the device exhibits (and that the driver should be prepared to work around).

NAME

sd — SCSI and ATAPI disk driver

SYNOPSIS

```
sd* at scsibus? target ? lun ?
sd3 at scsibus0 target 3 lun 0
sd* at atapibus? drive ? flags 0x0000
```

DESCRIPTION

The **sd** driver provides support for SCSI bus and Advanced Technology Attachment Packet Interface (ATAPI) disks. It allows the disk to be divided up into a set of pseudo devices called *partitions*. In general the interfaces are similar to those described by wd(4).

Where the wd(4) device has a fairly low level interface to the system, SCSI devices have a much higher level interface and talk to the system via a SCSI host adapter (e.g., ahc(4)). A SCSI adapter must also be separately configured into the system before a SCSI disk can be configured.

When the SCSI adapter is probed during boot, the SCSI bus is scanned for devices. Any devices found which answer as '*Direct*' type devices will be attached to the **sd** driver.

For the use of flags with ATAPI devices, see wd(4).

PARTITIONING

On many systems disklabel(8) is used to partition the drive into filesystems. On some systems the NetBSD portion of the disk resides within a native partition, and another program is used to create the NetBSD portion.

For example, the i386 port uses fdisk(8) to partition the disk into a BIOS level partition. This allows sharing the disk with other operating systems.

CONFIGURATION OPTIONS

The following config(1) options may be applied to SCSI disks as well as to other disks.

SDRETRIES — Set the number of retries that will be performed for operations it makes sense to retry (e.g., normal reads and writes). The default is four (4).

SD_IO_TIMEOUT — Set amount of time, in milliseconds, a normal read or write is expected to take. The defaults is sixty seconds (60000 milliseconds). This is used to set watchdog timers in the SCSI HBA driver to catch commands that might have died on the device.

IOCTLS

The following ioctl(2) calls apply to SCSI disks as well as to other disks. They are defined in the header file ⟨disklabel.h⟩.

DIOCGDINFO — Read, from the kernel, the in-core copy of the disklabel for the drive. This may be a fictitious disklabel if the drive has never been initialized, in which case it will contain information read from the SCSI inquiry commands.

DIOCSDINFO — Give the driver a new disklabel to use. The driver *will not* write the new disklabel to the disk.

DIOCKLABEL — Keep or drop the in-core disklabel on the last close.

DIOCWLABEL — Enable or disable the driver's software write protect of the disklabel on the disk.

DIOCWDINFO — Give the driver a new disklabel to use. The driver *will* write the new disklabel to the disk.

`DIOCLOCK` Lock the media cartridge into the device, or unlock a cartridge previously locked. Used to prevent user and software eject while the media is in use.

`DIOCEJECT` Eject the media cartridge from a removable device.

In addition, the `scsi`(4) general **`ioctl`**() commands may be used with the **`sd`** driver, but only against the '`c`' (whole disk) partition.

NOTES

If a removable device is attached to the **`sd`** driver, then the act of changing the media will invalidate the disklabel and information held within the kernel. To avoid corruption, all accesses to the device will be discarded until there are no more open file descriptors referencing the device. During this period, all new open attempts will be rejected. When no more open file descriptors reference the device, the first next open will load a new set of parameters (including disklabel) for the drive.

FILES

`/dev/sd`*up* block mode SCSI disk unit *u*, partition *p*
`/dev/rsd`*up* raw mode SCSI disk unit *u*, partition *p*

DIAGNOSTICS

None.

SEE ALSO

`ioctl`(2), `intro`(4), `scsi`(4), `wd`(4), `disklabel`(5), `disklabel`(8), `fdisk`(8), `scsictl`(8)

HISTORY

The **`sd`** driver was originally written for Mach 2.5, and was ported to FreeBSD by Julian Elischer. It was later ported to NetBSD.

NAME

se — Cabletron EA41x SCSI bus Ethernet interface driver

SYNOPSIS

```
se* at scsibus? target ? lun ?
```

DESCRIPTION

The **se** driver supports the Cabletron EA41x SCSI bus Ethernet interface.

This driver is a bit unusual. It must look like a network interface and it must also appear to be a SCSI device to the SCSI system.

In addition, to facilitate SCSI commands issued by userland programs, there are **open**(), **close**(), and **ioctl**() entry points. This allows a user program to, for example, display the EA41x statistic and download new code into the adaptor − functions which can't be performed through the `ifconfig`(8) interface. Normal operation does not require any special userland program.

SEE ALSO

`scsi`(4), `ifconfig`(8)

AUTHORS

Ian Dall <ian.dall@dsto.defence.gov.au>

Acknowledgement: Thanks are due to Philip L. Budne <budd@cs.bu.edu> who reverse engineered the EA41x. In developing this code, Phil's userland daemon "etherd", was referred to extensively in lieu of accurate documentation for the device.

BUGS

The EA41x doesn't conform to the SCSI specification in much at all. About the only standard command supported is "inquiry". Most commands are 6 bytes long, but the recv data is only 1 byte. Data must be received by periodically polling the device with the recv command.

NAME

sea — Seagate/Future Domain ISA SCSI adapter card driver

SYNOPSIS

```
sea0 at isa? iomem 0xc8000 irq 5
scsibus* at sea?
```

DESCRIPTION

The **sea** driver provides support for the following SCSI controllers on ISA bus boards:

ST01/02
Future Domain TMC-885
Future Domain TMC-950

SEE ALSO

cd(4), ch(4), intro(4), scsi(4), sd(4), st(4)

NAME

sec — Acorn SCSI Expansion Card driver

SYNOPSIS

```
sec* at podulebus0 slot ?
scsibus* at sec?
```

DESCRIPTION

The **sec** driver supports the Acorn SCSI Expansion Card, model numbers AKA30, AKA31, and AKA32. These cards support 8-bit parallel synchronous transfers at up to 4 MHz.

SEE ALSO

podulebus(4), scsi(4)

NAME

seeprom — 24-series I2C EEPROM driver

SYNOPSIS

```
seeprom0 at iic0 addr 0x50 size 128
```

DESCRIPTION

The **seeprom** driver provides support for the 24-series of I2C EEPROMs, avaliable from a variety of vendors.

Access to the contents of the memory is through a character device.

HISTORY

The **seeprom** device appeared in NetBSD 2.0.

BUGS

A device major number is not assigned.

NAME

sem — POSIX semaphores

SYNOPSIS

To link into the kernel:

```
options P1003_1B_SEMAPHORE
```

DESCRIPTION

The **sem** facility provides system calls used by the standard C library (libc) to implement POSIX semaphores.

SEE ALSO

config(1), sem_destroy(3), sem_getvalue(3), sem_init(3), sem_open(3), sem_post(3), sem_wait(3), options(4)

HISTORY

The **sem** facility appeared as a kernel option in NetBSD 2.0.

BUGS

The current implementation does not support shared, unnamed semaphores.

NAME

ser — Amiga 8520 serial communications interface

SYNOPSIS

```
ser0 at mainbus0
```

DESCRIPTION

The Amiga 8520 controls, among other things, a single port EIA RS-232C (CCITT V.28) communications interface with a single character buffer. Such an interface is built-in to all Amiga machines.

Input and output for each line may set to one of following baud rates; 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600 or 76800.

FILES

`/dev/tty00`

DIAGNOSTICS

ser0: silo overflow. The single-character input “silo” has overflowed and incoming data has been lost.

ser0: %d ring buffer overflows. The software based input ring buffer has overflowed %d times and incoming data has been lost.

SEE ALSO

`tty`(4)

HISTORY

The Amiga **ser** device first appeared in NetBSD 1.0

BUGS

Data loss is possible on busy systems with baud rates greater than 300.

NAME

ses — SCSI Environmental Services Driver

SYNOPSIS

ses* **at scsibus? target**? **lun**?

DESCRIPTION

The **ses** driver provides support for all SCSI devices of the environmental services class that are attached to the system through a supported SCSI Host Adapter, as well as emulated support for SAF-TE (SCSI Accessible Fault Tolerant Enclosures). The environmental services class generally are enclosure devices that provide environmental information such as number of power supplies (and state), temperature, device slots, and so on.

A SCSI Host adapter must also be separately configured into the system before a SCSI Environmental Services device can be configured.

IOCTLS

The following `ioctl`(2) calls apply to devices. They are defined in the header file ⟨`scsipi/ses.h`⟩ (q.v.).

`SESIOC_GETNOBJ` Used to find out how many objects are driven by this particular device instance.

`SESIOC_GETOBJMAP` Read, from the kernel, an array of SES objects which contains the object identifier, which sub-enclosure it is in, and the type of the object.

`SESIOC_GETENCSTAT` Get the overall enclosure status.

`SESIOC_SETENCSTAT` Set the overall enclosure status.

`SESIOC_GETOBJSTAT` Get the status of a particular object.

`SESIOC_SETOBJSTAT` Set the status of a particular object.

`SESIOC_GETTEXT` Get the associated help text for an object (not yet implemented). devices often have descriptive text for an object which can tell you things like location (e.g, "left power supply").

`SESIOC_INIT` Initialize the enclosure.

FILES

`/dev/ses`*N* The *Nth* **ses** device.

DIAGNOSTICS

When the kernel is configured with DEBUG enabled, the first open to an SES device will spit out overall enclosure parameters to the console.

SEE ALSO

`getencstat`(8), `sesd`(8), `setencstat`(8), `setobjstat`(8)

HISTORY

The **ses** driver was written for the SCSI subsystem by Matthew Jacob. This is the functional equivalent of a similar driver available in Solaris, Release 7.

NAME

sf — Adaptec AIC-6915 10/100 Ethernet driver

SYNOPSIS

```
sf* at pci? dev ? function ?
```

Configuration of PHYs may also be necessary. See mii(4).

DESCRIPTION

The **sf** device driver supports the Adaptec AIC-6915 10/100 Ethernet chip. This chip is found on several Adaptec Ethernet boards:

- ANA-62011 Single port 10/100 64-bit
- ANA-62022 Dual port 10/100 64-bit
- ANA-62044 Quad port 10/100 64-bit
- ANA-62020 Single port 100BASE-FX 64-bit
- ANA-69011 Single port 10/100 32-bit

SEE ALSO

arp(4), ifmedia(4), mii(4), netintro(4), pci(4), ifconfig(8)

HISTORY

The **sf** driver first appeared in NetBSD 1.6.

AUTHORS

The **sf** driver was written by Jason R. Thorpe ⟨thorpej@NetBSD.org⟩.

BUGS

The **sf** driver does not support the IPv4/TCP/UDP checksum function of the AIC-6915.

The **sf** driver does not support the VLAN function of the AIC-6915.

NAME

sf2r — SoundForte RadioLink SF16-FMR2 FM radio device driver

SYNOPSIS

```
option RADIO_TEA5759

sf2r0  at isa? port 0x384
radio* at sf2r0
```

DESCRIPTION

The **sf2r** driver provides support for the SF16-FMR2 FM radio tuners.

The SF16-FMR2 is a stereo FM tuner that allows to tune in the range 87.5 - 108.0 MHz, report signal status on the current frequency, force audio output to mono, perform hardware signal search, and has an internal AFC.

The SF16-FMR2 cards take only one I/O port. The I/O port is set by the driver to the value specified in the configuration file and must be 0x384.

Philips Semiconductors released two almost identical chips TEA5757 and TEA5759. The TEA5757 is used in FM-standards in which the local oscillator frequency is above the radio frequency (e.g. European and American standards). The TEA5759 is the version in which the oscillator frequency is below the radio frequency (e.g. Japanese standards). The option *RADIO_TEA5759* changes the algorithm of frequency calculation for the TEA5757 based cards to conform with the Japanese standards.

SEE ALSO

isa(4), radio(4)

HISTORY

The **sf2r** device driver appeared in OpenBSD 3.0 and NetBSD 1.6.

AUTHORS

The **sf2r** driver was written by Vladimir Popov and Maxim Tsyplakov. The man page was written by Vladimir Popov.

BUGS

MediaForte made two variants of the SF16-FMR2 cards, the first one has an internal amplifier of the output sound, the second one does not have such an amplifier. The current driver supports only the second variant.

NAME

sfb — PMAGB-B HX colour/grayscale accelerated 2-D framebuffer

SYNOPSIS

```
sfb* at tc? slot ? offset ?
wsdisplay* at sfb?
```

DESCRIPTION

The **sfb** driver provides support for the PMAGB-B HX colour/grayscale 2-D framebuffer for the TURBOchannel bus. The PMAGB-B is an 8 bpp colour/grayscale framebuffer capable of running at a resolution of 1280-by-1024 at 66 or 72 Hz.

SEE ALSO

cfb(4), mfb(4), px(4), pxg(4), tc(4), tfb(4), wscons(4)

BUGS

NetBSD/pmax does not currently support the machine-independent wscons(4) interface and uses a machine-dependent version.

NAME

sgsmix — driver for SGS 7433 Basic Audio Processor found in some Apple machines

SYNOPSIS

```
iic* at cuda?
sgsmix* at iic? addr 0x8a
```

DESCRIPTION

The **sgsmix** driver provides support for the external mixer chip on Apple's "personality cards" found in beige Power Macintosh G3. All mixer controls are accessed through the awacs(4) driver, **sgsmix** additionally provides bass and treble controls.

SEE ALSO

awacs(4), cuda(4), iic(9)

NAME

shb — bus for SuperH on-chip peripheral devices

SYNOPSIS

```
shb* at mainbus?
```

DESCRIPTION

The **shb** driver provides a bus to which drivers for integrated peripheral devices found in various SuperH microprocessors attach.

adc	analog/digital converter.
sci	serial communication interface.
scif	serial communication interface with FIFO.
wdog	watchdog timer.

SEE ALSO

adc(4), sci(4), scif(4), wdog(4)

HISTORY

The **shb** driver first appeared in NetBSD 1.5.

NAME

shpcic — RENESAS SuperH on-chip PCI controller

SYNOPSIS

```
shpcic* at mainbus?
pci* at shpcic? bus?
```

DESCRIPTION

The **shpcic** driver provides support for the RENESAS SuperH PCI controller. SHPCIC is an on-chip module found in following models:

SH7751

SH7751R

SEE ALSO

pci(4)

HISTORY

The **shpcic** driver first appeared in NetBSD 4.0.

NAME

si, **sw** — NCR 5380 SCSI bus host adaptor driver

SYNOPSIS

sun3

```
si0 at obio0 addr 0x140000 ipl 2
```

sun3 and sun3x

```
si0 at vme2 addr 0x200000 ipl 2 vect 0x40
si1 at vme2 addr 0x204000 ipl 2 vect 0x41
```

sun3/E

```
sebuf0 at vme2 addr 0x300000 ipl 2 vect 0x74 # and 0x75
sebuf1 at vme2 addr 0x340000 ipl 2 vect 0x76 # and 0x77
si* at sebuf?
```

sun4 and sun2

```
si0 at vme0 addr 0x200000 pri 2 vec 0x40
```

sun4/100

```
sw0 at obio0 addr 0x0a000000 level 3
```

DESCRIPTION

The **si** and **sw** "SCSI Weird" drivers provide support for the NCR 5380 SCSI Bus Controller (SBC) chip found on various Sun Microsystems CPU motherboards (obio), and on the "Sun-3 VME SCSI" (Sun part # 501-1236) board used in systems with VME bus.

sun3 and sun3x

The sun3 and sun3x version of this driver can be configured with a *flags* directive in the `config`(1) file. The values are bits in a bitfield, and are interpreted as follows:

0x000ff	Set bit (1<<target) to disable SCSI disconnect/reselect
0x0ff00	Set bit (1<<(target+8)) to disable SCSI parity checking
0x10000	Set this bit to disable DMA interrupts (poll)
0x20000	Set this bit to disable DMA entirely (use PIO)

For example: "flags 0x1000f" would disable DMA interrupts, and disable disconnect/reselect for targets 0-3. The "target" is the SCSI ID number of a particular device on a particular SCSI bus.

sun4

The sun4 version of this driver can also be configured with a *flags* directive in the `config`(1) file. The values are bits in a bitfield, and are interpreted as follows:

0x01	Use DMA (may be polled)
0x02	Use DMA completion interrupts
0x04	Allow SCSI disconnect/reselect

For example: "flags 0x07" would enable DMA, interrupts, and reselect. By default, DMA is enabled in the sun4 driver.

SEE ALSO

`cd`(4), `ch`(4), `intro`(4), `scsi`(4), `sd`(4), `st`(4)

AUTHORS

David Jones, Gordon Ross ⟨gwr@NetBSD.org⟩,
Adam Glass ⟨glass@NetBSD.org⟩,
Jason R. Thorpe ⟨thorpej@NetBSD.org⟩.

BUGS

The VME variant has a bit to enable or disable the DMA engine, but that bit also gates the interrupt line from the NCR5380 (!!). Therefore, in order to get any interrupt from the NCR5380, (i.e. for reselect) one must clear the DMA engine transfer count and then enable DMA. This has the further complication that you CAN NOT touch the NCR5380 while the DMA enable bit is set, so we have to turn DMA back off before we even look at the NCR5380.

Support for the Sun 4/100 **sw** "SCSI Weird" is not complete. DMA works, but interrupts (and, thus, reselection) don't for reasons unknown. Further progress has halted pending the availability of a machine for testing.

DMA, DMA completion interrupts, and reselection work fine on a Sun 4/260 with modern SCSI-II disks attached. There have been reports of reselection failing on Sun Shoebox-type configurations where there are multiple non-SCSI devices behind Emulex or Adaptec bridges. These devices pre-date the SCSI-I spec, and might not behave the way the NCR5380 code expects. For this reason, only DMA is enabled by default in the sun4 driver.

NAME

sii — SII SCSI adaptor

SYNOPSIS

```
sii* at ibus0 addr ?
scsibus* at sii?
```

DESCRIPTION

The **sii** driver provides support for the DEC SII SCSI adaptor ASIC used in the DECstation 2100, 3100, and 5100, and in the VAXstation 3100.

The **sii** is a medium-performance implementation of the SCSI-I common command set supporting synchronous and asynchronous SCSI devices.

The SII cabling is unusual: the bulkhead connector is very similar to a standard submicro wide SCSI-II connector, but has the other gender.

The **sii** chip only supports DMA to or from a fixed window in memory. The driver uses this "window" area as a bounce buffer, copying data to and from the DMA region.

SEE ALSO

cd(4), ch(4), ibus(4), intro(4), sd(4), st(4)

HISTORY

The **sii** driver first appeared in 4.4BSD.

NAME

siisata — Silicon Image SATA-II controllers driver

SYNOPSIS

siisata* **at pci? dev ? function ?**

DESCRIPTION

The **siisata** driver supports the Silicon Image SteelVine family of SATA-II controllers, interfacing the hardware with the ata(4) and atapi(4) subsystems.

The following controllers are supported by the **siisata** driver:

Silicon Image SiI3124 4-port PCI/PCI-X
Silicon Image SiI3132 2-port PCI-Express x1
Silicon Image SiI3531 1-port PCI-Express x1

SEE ALSO

ata(4), atapi(4), pci(4), wd(4)

HISTORY

The **siisata** driver first appeared in NetBSD 5.0.

AUTHORS

The **siisata** driver was written by Jonathan A. Kollasch ⟨jakllsch@kollasch.net⟩.

BUGS

SATA Native Command Queueing is not yet supported.
Device hot swapping is not yet supported.
Silicon Image's Software RAID is not yet supported by the ataraid(4) driver. raid(4) can be used instead.

NAME

siop — Symbios Logic/NCR 53c8xx SCSI driver

SYNOPSIS

hp700

```
siop* at mainbus? irq 3
siop* at phatomas? irq 3
```

PCI

```
siop* at pci? dev ? function ?

options SIOP_SYMLED
scsibus* at siop?
```

DESCRIPTION

The **siop** driver provides support for the Symbios Logic/NCR 53x8xx series of SCSI controller chips:

- 53c810, 53c810a and 53c815 (Fast SCSI)
- 53c820, 53c825 and 53c825a (Fast-Wide SCSI)
- 53c860 (Ultra SCSI)
- 53c875 and 53c875j (Ultra-Wide SCSI)
- 53c876 (Dual Ultra-Wide SCSI)
- 53c885 (Ultra-Wide SCSI and Ethernet)
- 53c895 (Ultra2-Wide SCSI)
- 53c896 (PCI 64bit, dual Ultra2-Wide SCSI)
- 53c1010-33 (PCI 64bit, dual Ultra160 SCSI)
- 53c1510d (PCI 64bit, dual Ultra2-wide SCSI)

The SIOP_SYMLED option causes the driver to report SCSI activity on the GPIO pin 1, which is connected to the activity LED on some adapters. At this time only the 53c895 based Symbios and Tekram adapters are known to require this.

If both **siop** and the esiop(4) drivers are compiled, esiop(4) will take precedence for controllers it supports.

SEE ALSO

cd(4), ch(4), esiop(4), intro(4), pci(4), scsi(4), sd(4), st(4), uk(4)

HISTORY

The **siop** driver first appeared in NetBSD 1.5.

AUTHORS

The **siop** driver was written by Manuel Bouyer ⟨Manuel.Bouyer@lip6.fr⟩ for NetBSD.

BUGS

siop supports the 53c1010-33 in Ultra2-wide mode only. Use esiop(4) for Ultra160 support.

NAME

sip — SiS 900-based Ethernet driver

SYNOPSIS

sip* **at pci? dev ? function ?**

Configuration of PHYs is also necessary. See mii(4).

DESCRIPTION

The **sip** device driver supports Ethernet interfaces based on the Silicon Integrated Systems SiS 900, SiS 7016, and National Semiconductor DP83815 Ethernet chips.

The SiS 900 is found on many motherboards featuring SiS chipsets, and on some embedded systems, and has a built-in PHY. The SiS 7016 is an older version of the chip, which uses an external PHY.

The National Semiconductor DP83815 is found on NetGear FA-311 and FA-312 Ethernet boards, and has a built-in PHY.

SEE ALSO

arp(4), ifmedia(4), mii(4), netintro(4), pci(4), ifconfig(8)

HISTORY

The **sip** driver first appeared in NetBSD 1.5.

AUTHORS

The **sip** driver was originally written by Jason R. Thorpe for Network Computer, Inc.

It has since been modified by Jason R. Thorpe ⟨thorpej@NetBSD.org⟩ and Allen Briggs ⟨briggs@NetBSD.org⟩.

NAME

siside — Silicon Integrated System IDE disk controllers driver

SYNOPSIS

siside* **at pci? dev ? function ? flags 0x0000**

DESCRIPTION

The **siside** driver supports the Silicon Integrated System IDE controllers, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **siside** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

The timings used for the PIO and DMA modes for controllers listed above are for a PCI bus running at 30 or 33 MHz. This driver may not work properly on overclocked systems.

NAME

sk, **skc** — SysKonnect XMAC II and Marvell GMAC based gigabit ethernet

SYNOPSIS

```
skc* at pci? dev ? function ?
sk* at skc?
mskc* at pci? dev ? function ?
msk* at skc?
```

DESCRIPTION

The **sk** driver provides support for SysKonnect based gigabit ethernet adapters and Marvell based gigabit ethernet adapters, including the following:

- SK-9821 SK-NET GE-T single port, copper adapter
- SK-9822 SK-NET GE-T dual port, copper adapter
- SK-9841 SK-NET GE-LX single port, single mode fiber adapter
- SK-9842 SK-NET GE-LX dual port, single mode fiber adapter
- SK-9843 SK-NET GE-SX single port, multimode fiber adapter
- SK-9844 SK-NET GE-SX dual port, multimode fiber adapter
- SK-9521 V2.0 single port, copper adapter (32-bit)
- SK-9821 V2.0 single port, copper adapter
- SK-9843 V2.0 single port, copper adapter
- 3Com 3c940 single port, copper adapter
- Belkin Gigabit Desktop Network PCI Card, single port, copper (32-bit)
- D-Link DGE-530T single port, copper adapter
- Linksys EG1032v2 single-port, copper adapter
- Linksys EG1064v2 single-port, copper adapter

The **msk** driver provides support for the Marvell Yukon-2 based Gigabit Ethernet adapters, including the following:

- Marvell Yukon 88E8035, copper adapter
- Marvell Yukon 88E8036, copper adapter
- Marvell Yukon 88E8038, copper adapter
- Marvell Yukon 88E8050, copper adapter
- Marvell Yukon 88E8052, copper adapter
- Marvell Yukon 88E8053, copper adapter
- Marvell Yukon 88E8055, copper adapter
- SK-9E21 1000Base-T single port, copper adapter
- SK-9E22 1000Base-T dual port, copper adapter
- SK-9E81 1000Base-SX single port, multimode fiber adapter
- SK-9E82 1000Base-SX dual port, multimode fiber adapter
- SK-9E91 1000Base-LX single port, single mode fiber adapter
- SK-9E92 1000Base-LX dual port, single mode fiber adapter
- SK-9S21 1000Base-T single port, copper adapter
- SK-9S22 1000Base-T dual port, copper adapter
- SK-9S81 1000Base-SX single port, multimode fiber adapter
- SK-9S82 1000Base-SX dual port, multimode fiber adapter
- SK-9S91 1000Base-LX single port, single mode fiber adapter
- SK-9S92 1000Base-LX dual port, single mode fiber adapter
- SK-9E21D 1000Base-T single port, copper adapter

The SysKonnect based adapters consist of two main components: the XaQti Corp. XMAC II Gigabit MAC (**sk**) and the SysKonnect GEnesis controller ASIC (**skc**). The XMAC provides the Gigabit MAC and PHY support while the GEnesis provides an interface to the PCI bus, DMA support, packet buffering and arbitration. The GEnesis can control up to two XMACs simultaneously, allowing dual-port NIC configurations.

The Marvell based adapters are a single integrated circuit, but are still presented as a separate MAC (**sk**) and controller ASIC (**skc**). At this time, there are no dual-port Marvell based NICs.

The **sk** driver configures dual port SysKonnect adapters such that each XMAC is treated as a separate logical network interface. Both ports can operate independently of each other and can be connected to separate networks. The SysKonnect driver software currently only uses the second port on dual port adapters for failover purposes: if the link on the primary port fails, the SysKonnect driver will automatically switch traffic onto the second port.

The XaQti XMAC II supports full and half duplex operation with autonegotiation. The XMAC also supports unlimited frame sizes. Support for jumbo frames is provided via the interface MTU setting. Selecting an MTU larger than 1500 bytes with the `ifconfig`(8) utility configures the adapter to receive and transmit jumbo frames. Using jumbo frames can greatly improve performance for certain tasks, such as file transfers and data streaming.

Hardware TCP/IP checksum offloading for IPv4 is supported.

The following media types and options (as given to `ifconfig`(8)) are supported:

media autoselect
: Enable autoselection of the media type and options. The user can manually override the autoselected mode.

media 1000baseSX **mediaopt** full-duplex
: Set 1000Mbps (Gigabit Ethernet) operation on fiber and force full-duplex mode.

media 1000baseSX **mediaopt** half-duplex
: Set 1000Mbps (Gigabit Ethernet) operation on fiber and force half-duplex mode.

media 1000baseT **mediaopt** full-duplex
: Set 1000Mbps (Gigabit Ethernet) operation and force full-duplex mode.

For more information on configuring this device, see `ifconfig`(8). To view a list of media types and options supported by the card, try **ifconfig -m** ⟨device⟩. For example, **ifconfig -m** *sk0*.

DIAGNOSTICS

sk%d: couldn't map memory A fatal initialization error has occurred.

sk%d: couldn't map ports A fatal initialization error has occurred.

sk%d: couldn't map interrupt A fatal initialization error has occurred.

sk%d: failed to enable memory mapping! The driver failed to initialize PCI shared memory mapping. This might happen if the card is not in a bus-master slot.

sk%d: no memory for jumbo buffers! The driver failed to allocate memory for jumbo frames during initialization.

sk%d: watchdog timeout The device has stopped responding to the network, or there is a problem with the network connection (cable).

SEE ALSO

`ifmedia`(4), `intro`(4), `netintro`(4), `pci`(4), `ifconfig`(8)

XaQti XMAC II datasheet, http://www.xaqti.com.

SysKonnect GEnesis programming manual, http://www.syskonnect.com.

HISTORY

The **sk** device driver first appeared in FreeBSD 3.0. OpenBSD support was added in OpenBSD 2.6. NetBSD support was added in NetBSD 2.0.

The **msk** driver first appeared in OpenBSD 4.0, and was ported to NetBSD 4.0.

AUTHORS

The **sk** driver was written by Bill Paul ⟨wpaul@ctr.columbia.edu⟩. Support for the Marvell Yukon-2 was added by
Mark Kettenis ⟨kettenis@openbsd.org⟩.

BUGS

This driver is *experimental*.

Support for checksum offload is unimplemented.

Performance with at least some Marvell-based adapters is poor, especially on loaded PCI buses or when the adapters are behind PCI-PCI bridges. It is believed that this is because the Marvell parts have significantly less buffering than the original SysKonnect cards had.

NAME

sl — Serial Line IP (SLIP) network interface

SYNOPSIS

```
pseudo-device sl
```

DESCRIPTION

The **sl** interface allows asynchronous serial lines to be used as IPv4 network interfaces using the SLIP protocol.

To use the **sl** interface, the administrator must first create the interface and assign a tty line to it. The **sl** interface is created using the `ifconfig`(8) **create** subcommand, and `slattach`(8) is used to assign a tty line to the interface. Once the interface is attached, network source and destination addresses and other parameters are configured via `ifconfig`(8).

The **sl** interface can use Van Jacobson TCP header compression and ICMP filtering. The following flags to `ifconfig`(8) control these properties of a SLIP link:

link0	Turn on Van Jacobson header compression.
-link0	Turn off header compression. (default)
link1	Don't pass through ICMP packets.
-link1	Do pass through ICMP packets. (default)
link2	If a packet with a compressed header is received, automatically enable compression of outgoing packets. (default)
-link2	Don't auto-enable compression.

DIAGNOSTICS

sl%d: af%d not supported . The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

SEE ALSO

`inet`(4), `intro`(4), `ppp`(4), `strip`(4), `ifconfig`(8), `slattach`(8), `sliplogin`(8), `slstats`(8)

J. Romkey, *A Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP*, RFC, 1055, June 1988.

Van Jacobson, *Compressing TCP/IP Headers for Low-Speed Serial Links*, RFC, 1144, February 1990.

HISTORY

The **sl** device appeared in NetBSD 1.0.

BUGS

SLIP can only transmit IPv4 packets between preconfigured hosts on an asynchronous serial link. It has no provision for address negotiation, carriage of additional protocols (e.g. XNS, AppleTalk, DECNET), and is not designed for synchronous serial links. This is why SLIP has been superseded by the Point-to-Point Protocol (PPP), which does all of those things, and much more.

NAME

slhci — Cypress/ScanLogic SL811HS USB Host Controller driver

SYNOPSIS

PCMCIA (CF) controllers

```
slhci*   at pcmcia? function ?
usb*     at slhci?
```

ISA controllers

```
slhci*   at isa? port ? irq ?
usb*     at slhci?
```

x68k

```
slhci0   at intio0 addr 0xece380 intr 251
slhci1   at intio0 addr 0xeceb80 intr 250
usb*     at slhci?
```

```
options SLHCI_TRY_LSVH
```

DESCRIPTION

The **slhci** driver provides support for Cypress/ScanLogic SL811HS USB Host Controller.

The driver supports control, bulk, and interrupt transfers but not isochronous (audio), which cannot be supported by this chip without perfectly reliable 1ms interrupts. USB is polled and this chip requires the driver to initiate all transfers. The driver interrupts at least once every ms when a device is attached even if no data is transferred. The driver polls the chip when the transfer is expected to be completed soon; with maximum use of the bus, the driver will not exit for most of each ms. Use of this driver can easily have a significant performance impact on any system.

The chip is unreliable in some conditions, possibly due in part to difficulty meeting timing restrictions (this is likely to be worse on multiprocessor systems). Unexpected device behavior may trigger some problems; power cycling externally powered devices may help resolve persistent problems. Detection of invalid chip state will usually cause the driver to halt, however is recommended that all data transfers be verified. Data corruption due to controller error will not be detected automatically. Unmounting and remounting a device is necessary to prevent use of cached data.

The driver currently will start the next incoming packet before copying in the previous packet but will not copy the next outgoing packet before the previous packet is transferred. Reading or writing the chip is about the same speed as the USB bus, so this means that one outgoing transfer is half the speed of one incoming transfer and two outgoing transfers are needed to use the full available bandwidth.

All revisions of the SL811HS have trouble with low speed devices attached to some (likely most) hubs. Low speed traffic via hub is not allowed by default, but can be enabled with **options SLHCI_TRY_LSVH** in the kernel config file or by setting the *slhci_try_lsvh* variable to non-zero using ddb(4) or gdb(1).

Many USB keyboards have built in hubs and may be low speed devices. All USB mice I have seen are low speed devices, however a serial mouse should be usable on a hub with a full speed Serial-USB converter. A PS2-USB keyboard and mouse converter is likely to be a single low speed device.

Some hardware using this chip does not provide the USB minimum 100mA current, which could potentially cause problems even with externally powered hubs. The system can allow excess power use in some other cases as well. Some signs of excess power draw may cause the driver to halt, however this may not stop the power draw. To be safe verify power use and availability before connecting any device.

HARDWARE

Hardware supported by the **slhci** driver includes:

Ratoc *CFU1U*

Nereid Ethernet/USB/Memory board

SEE ALSO

pcmcia(4), isa(4), usb(4), config(1)

HISTORY

The **slhci** driver appeared in NetBSD 2.0 and was rewritten in NetBSD 5.0.

AUTHORS

Tetsuya Isaki ⟨isaki@NetBSD.org⟩
Matthew Orgass

NAME

slide — Symphony Labs and Winbond IDE disk controllers driver

SYNOPSIS

```
slide* at pci? dev ? function ? flags 0x0000
```

DESCRIPTION

The **slide** driver supports the Symphony Labs 82C105 and Winbond W83C553F IDE controllers, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **slide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

The timings used for the PIO and DMA modes for controllers listed above are for a PCI bus running at 30 or 33 MHz. This driver may not work properly on overclocked systems.

NAME

sm — SMC91Cxx-based Ethernet interfaces device driver

SYNOPSIS

```
sm0 at isa? port 0x300 irq 10
sm* at mhzc?
sm* at pcmcia? function ?
sm* at nubus?
```

DESCRIPTION

The **sm** device driver supports SMC91C9x-based and SMC91C1xx-based Ethernet interfaces.

The ISA attachment of the **sm** driver supports any SMC91C9x-based Ethernet interface on the ISA bus, including the EFA Info*Express SVC VLB interface, and the on-board SMC91C9x Ethernet found in many embedded PCs and laptop docking stations.

The mhzc attachment of the **sm** driver supports the Megahertz combo Ethernet and modem card.

The PCMCIA attachment of the **sm** driver supports Megahertz X-JACK Ethernet cards.

The nubus attachment of the **sm** driver supports SMC-based Ethernet cards for the macintosh.

MEDIA SELECTION

The SMC91C1xx supports the MII interface and so the list of supported media depends on several factors, including which PHY is attached. The SMC91C9x supports AUI and UTP media types.

To enable the AUI media, select the *10base5* or *aui* media type with `ifconfig`(8)'s **media** directive. To select UTP, select the *10baseT* or *utp* media type.

`ifconfig`(8)'s **-m** option will display the full list of supported **media** directives for a particular configuration.

DIAGNOSTICS

sm0: unable to read MAC address from CIS This indicates that the Ethernet address, which is stored in the CIS information on the Megahertz X-JACK PCMCIA Ethernet card, is corrupted.

SEE ALSO

`ifmedia`(4), `intro`(4), `isa`(4), `mhzc`(4), `mii`(4), `pcmcia`(4), `ifconfig`(8)

NAME

smsc — Standard Microsystems Corporation LPC47B397 Super I/O

SYNOPSIS

```
smsc0 at isa? port 0x2e
```

DESCRIPTION

The **smsc** driver provides support for the hardware monitoring portion of the Standard Microsystems Corporation LPC47B397, SCH5307-NS and SCH5317 Super I/O chips to be used with the envsys(4) API.

The chip supports 8 sensor inputs:

Sensor	Units	Typical Use
Temp0	uK	CPU 1 Temperature
Temp1	uK	CPU 2 Temperature
Temp2	uK	Ambient Temperature
Temp3	uK	unused
Fan0	RPM	CPU 1 Fan
Fan1	RPM	unused
Fan2	RPM	unused
Fan3	RPM	CPU 2 Fan

HARDWARE

Motherboards with the chip supported by the **smsc** driver include:

Tyan Thunder K8WE (S2895)

SEE ALSO

envsys(4), envstat(8)

HISTORY

The **smsc** device appeared in NetBSD 5.0.

NAME

sn — National Semiconductor DP83932 (SONIC) based Ethernet device driver

SYNOPSIS

arc

```
sn0 at jazzio?
```

mac68k

```
sn* at obio?
sn* at nubus?
```

DESCRIPTION

The **sn** interface provides access to a 10 Mb/s Ethernet network via the National Semiconductor DP83932 (SONIC) Ethernet chip set.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **sn** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

HARDWARE

arc

The **sn** driver supports on-board JAZZ based SONIC interfaces found on Acer PICA and NEC machines.

mac68k

The **sn** driver is currently known to support the following NuBus cards:

- Apple LC Twisted-pair (part #820-0532-A) PDS card
- Cayman Gatorcard PDS
- Dayna DaynaPort/E30

In addition, the **sn** interface supports the following interfaces:

- on-board Ethernet for non-AV Quadras
- on-board Ethernet for 500-series PowerBooks
- Apple CS Ethernet Twisted-pair card for Comm Slot found on LC575, Quadra 630, LC630, and Performa 580.

DIAGNOSTICS

sn%d: transmit FIFO underrun
sn%d: receive FIFO overrun
sn%d: receive buffer exceeded
sn%d: receive buffers exhausted
sn%d: receive descriptors exhausted These messages indicate that the interface gets errors (due to heavy load etc.) and reinitialized.

SEE ALSO

`arp`(4), `inet`(4), `netintro`(4), `ifconfig`(8)

HISTORY

The **sn** interface for mac68k, which was derived from a driver for old NetBSD pica port, first appeared in NetBSD 1.3.

Jason Thorpe has rewritten a new machine independent SONIC driver which uses `bus_dma`(9) and `bus_space`(9) APIs after NetBSD 1.5 release, and arc has been switched to using the MI driver. mac68k has been also switched to using the MI driver after NetBSD 4.0 release.

NAME

snapper — Apple audio device driver

SYNOPSIS

```
snapper* at obio?
audio* at snapper?
```

DESCRIPTION

The **snapper** driver provides support for audio hardware using the TAS3004 chipset found in many Apple iBooks, PowerBooks, and PowerMacs. Other i2s compatible controllers found in Mac Minis, which lack a TAS3004, are also supported.

Hardware features:

- Supports data word lengths of 16 to 24 bits
- Processes sample rates from 32kHz to 48kHz
- Provides digital equalization and compression capabilities

SEE ALSO

audio(4), awacs(4)

HISTORY

The **snapper** device driver appeared in NetBSD 2.0.

AUTHORS

The **snapper** driver was written by Tsubai Masanari with modifications by Jared D. McNeill.

BUGS

This driver needs more testing.

NAME

sony — Sony Miscellaneous Controller

SYNOPSIS

sony* **at acpi?**

DESCRIPTION

Some Sony notebook computers have a controller that handles various built-in devices. The **sony** driver provides support for accessing/modifying the settings of some of these devices via the sysctl(8) interface.

The following sysctl(8) variables are available:

hw.sony0.brt [R/W] Controls current LCD brightness. Range [0-8].
hw.sony0.pbr [R/W] Controls power on LCD brightness. Range [0-8].
hw.sony0.cdp [R/W] Controls CD power.
hw.sony0.pid [R/O] Unknown
hw.sony0.ctr [R/W] Unknown
hw.sony0.pcr [R/W] Unknown
hw.sony0.cmi [R/W] Unknown
hw.sony0.ams [R/W] Audio control (mute when 0)
hw.sony0.hke [R/O] Indicates a Host Key Event. Bits are set when an event occurs and cleared when this value is read. The following table describes the bit set for each button pressed:

0x1000 S1 button
0x0800 S2 button
0x0200 Fn + F10 (magnify)
0x0100 Mute button
0x0020 Fn + F12 (suspend to disk)
0x0010 Fn + F7 (LCD/external monitor)
0x0008 Fn + F6 (brighter backlight)
0x0004 Fn + F5 (darker backlight)
0x0002 Fn + F4 (volume up)
0x0001 Fn + F3 (volume down)

SEE ALSO

acpi(4), spic(4)

HISTORY

The **sony** driver appeared in NetBSD 4.0.

AUTHORS

Sami Kantoluoto for the original driver and manual information. Christos Zoulas for cleaning up the driver and this manual page.

BUGS

- The **sony** driver just parses integer values from the acpi(4) tree. It could be more intelligent and parse other controls.
- The sysctl(8) interface is not great. The names of the sysctl(8) tree are not self-explanatory.
- No validity checks are performed on the user input. Playing with random values and/or unknown controls can harm your machine.

- The name of the driver is too generic.

NAME

spc — Fujitsu MB87030/MB89352 SCSI driver

SYNOPSIS

```
spc* at pcmcia? function ?
```

hp300

```
spc0 at dio? scode ?
```

luna68k

```
spc0 at mainbus0
```

x68k

```
spc0 at scsirom0
spc1 at scsirom1
```

```
scsibus* at spc?
```

DESCRIPTION

The **spc** driver provides support for the Fujitsu MB87030/MB89352 SCSI Protocol Controller (SPC) chips.

SEE ALSO

cd(4), ch(4), intro(4), pcmcia(4), scsi(4), sd(4), ss(4), st(4), uk(4), scsipi(9)

BUGS

Synchronous data transfers are not currently supported.

NAME

spdmem — Generic Memory Module Serial Presence Detect

SYNOPSIS

```
spdmem* at iic? addr 0x50
spdmem* at iic? addr 0x51
spdmem* at iic? addr 0x52
spdmem* at iic? addr 0x53
```

DESCRIPTION

The **spdmem** driver provides support for generic memory module Serial Presence Detect. During kernel boot it displays module type, and where possible, the size and rated speed of the memory module. Additional information such as bank configuration and width is printed if the kernel is booted in verbose mode.

SEE ALSO

iic(4), intro(4), ichsmb(4), nfsmb(4)

HISTORY

The **spdmem** driver first appeared in NetBSD 5.0.

AUTHORS

The **spdmem** driver was written by Nicolas Joly <njoly@pasteur.fr> with additional work by Paul Goyette <paul@whooppee.com>.

NAME

speaker — console speaker audio device driver

SYNOPSIS

```
spkr0 at pcppi?
#include <machine/spkr.h>
/dev/speaker
```

DESCRIPTION

The speaker device driver allows applications to control the console speaker on machines with a PC-like 8253 timer implementation.

Only one process may have this device open at any given time; open() and close() are used to lock and relinquish it. An attempt to open() when another process has the device locked will return -1 with an EBUSY error indication. Writes to the device are interpreted as 'play strings' in a simple ASCII melody notation. An **ioctl**() for tone generation at arbitrary frequencies is also supported.

Sound-generation does *not* monopolize the processor; in fact, the driver spends most of its time sleeping while the PC hardware is emitting tones. Other processes may emit beeps while the driver is running.

Applications may call **ioctl**() on a speaker file descriptor to control the speaker driver directly; definitions for the **ioctl**() interface are in ⟨machine/spkr.h⟩. The tone_t structure used in these calls has two fields, specifying a frequency (in hz) and a duration (in 1/100ths of a second). A frequency of zero is interpreted as a rest.

At present there are two such ioctls. SPKRTONE accepts a pointer to a single tone structure as third argument and plays it. SPKRTUNE accepts a pointer to the first of an array of tone structures and plays them in continuous sequence; this array must be terminated by a final member with a zero duration.

The play-string language is modelled on the PLAY statement conventions of IBM BASIC 2.0. The MB, MF and X primitives of PLAY are not useful in a UNIX environment and are omitted. The 'octave-tracking' feature is also new.

There are 84 accessible notes numbered 1-83 in 7 octaves, each running from C to B, numbered 0-6; the scale is equal-tempered A440 and octave 3 starts with middle C. By default, the play function emits half-second notes with the last 1/16th second being 'rest time'.

Play strings are interpreted left to right as a series of play command groups; letter case is ignored. Play command groups are as follows:

CDEFGAB -- letters A through G cause the corresponding note to be played in the current octave. A note letter may optionally be followed by an *accidental sign*, one of # + or -; the first two of these cause it to be sharped one half-tone, the last causes it to be flatted one half-tone. It may also be followed by a time value number and by sustain dots (see below). Time values are interpreted as for the L command below;.

O <n> -- if <n> is numeric, this sets the current octave. <n> may also be one of 'L' or 'N' to enable or disable octave-tracking (it is disabled by default). When octave-tracking is on, interpretation of a pair of letter notes will change octaves if necessary in order to make the smallest possible jump between notes. Thus "olbc" will be played as "olb>c", and "olcb" as "olc<b". Octave locking is disabled for one letter note following by >, < and O[0123456].

> -- bump the current octave up one.

< -- drop the current octave down one.

N <n> -- play note n, n being 1 to 84 or 0 for a rest of current time value. May be followed by sustain dots.

L <n> -- sets the current time value for notes. The default is L4, quarter notes. The lowest possible value is 1; values up to 64 are accepted. L1 sets whole notes, L2 sets half notes, L4 sets quarter notes, etc..

P <n> -- pause (rest), with <n> interpreted as for L. May be followed by sustain dots. May also be written '~'.

T <n> -- Sets the number of quarter notes per minute; default is 120. Musical names for common tempi are:

	Tempo	*Beats per Minute*
very slow	Larghissimo	
	Largo	40-60
	Larghetto	60-66
	Grave	
	Lento	
	Adagio	66-76
slow	Adagietto	
	Andante	76-108
medium	Andantino	
	Moderato	108-120
fast	Allegretto	
	Allegro	120-168
	Vivace	
	Veloce	
	Presto	168-208
very fast	Prestissimo	

M[LNS] -- set articulation. MN (N for normal) is the default; the last 1/8th of the note's value is rest time. You can set ML for legato (no rest space) or MS (staccato) 1/4 rest space.

Notes (that is, CDEFGAB or N command character groups) may be followed by sustain dots. Each dot causes the note's value to be lengthened by one-half for each one. Thus, a note dotted once is held for 3/2 of its undotted value; dotted twice, it is held 9/4, and three times would give 27/8.

Whitespace in play strings is simply skipped and may be used to separate melody sections.

FILES

`/dev/speaker`

SEE ALSO

`pcppi`(4)

AUTHORS

Eric S. Raymond ⟨esr@snark.thyrsus.com⟩

BUGS

Due to roundoff in the pitch tables and slop in the tone-generation and timer hardware (neither of which was designed for precision), neither pitch accuracy nor timings will be mathematically exact.

There is no volume control.

In play strings which are very long (longer than your system's physical I/O blocks) note suffixes or numbers may occasionally be parsed incorrectly due to crossing a block boundary.

NAME

spi — introduction to machine-independent SPI bus support and drivers

SYNOPSIS

```
spi* at mainbus?
```

Other attachments are machine-dependent and will depend on the bus topology of your system. See intro(4) for your system for more information.

DESCRIPTION

NetBSD includes a machine dependent SPI (Serial Peripherial Interface) bus subsystem, and several different machine-independent SPI device drivers.

Your system may support additional machine-dependent SPI devices. Consult your system's intro(4) for additional information.

SPI is a 4-wire synchronous full-duplex serial bus. Some systems provide support for Microwire, which is Philips' name for a strict subset of SPI, with more rigidly defined signaling. Therefore, Microwire devices are also supported by the SPI framework.

Note that when referencing SPI devices in a config(1) file, the 'slave' must be provided, as SPI lacks any way to automatically probe devices.

HARDWARE

NetBSD includes the following machine-independent SPI drivers

m25p	STMicroelectronics M25P family of NOR flash devices.
tm121temp	Texas Instruments TMP121 temperature sensor.

SEE ALSO

m25p(4), tm121temp(4)

HISTORY

The machine-independent SPI framework was written by Garrett D'Amore for the Champaign-Urbana Community Wireless Network Project (CUWiN), and appeared in NetBSD 4.0.

NAME

spic — Sony Programmable I/O Controller

SYNOPSIS

```
spic*    at acpi?
wsmouse* at spic?
```

DESCRIPTION

Some Sony computers have an I/O controller that handles various interface devices, e.g., the jog dial. The **spic** driver provides support for some of these.

The actual protocol used by the SPIC seems to vary a lot between different models; this driver has only been tested on the Vaio SRX77.

The **spic** driver handles the jog dial and its associated buttons. They are turned into a **wsmouse** which can the be accessed in the normal way. The jog dial works as a scroll button, and clicking it generates a click with the "middle" button.

SEE ALSO

acpi(4), wsmouse(4)

HISTORY

The **spic** driver appeared in NetBSD 1.6.

BUGS

The **spic** driver only works with some hardware configurations.

NAME

spif — SBus (spiffy) Serial/Parallel Interface

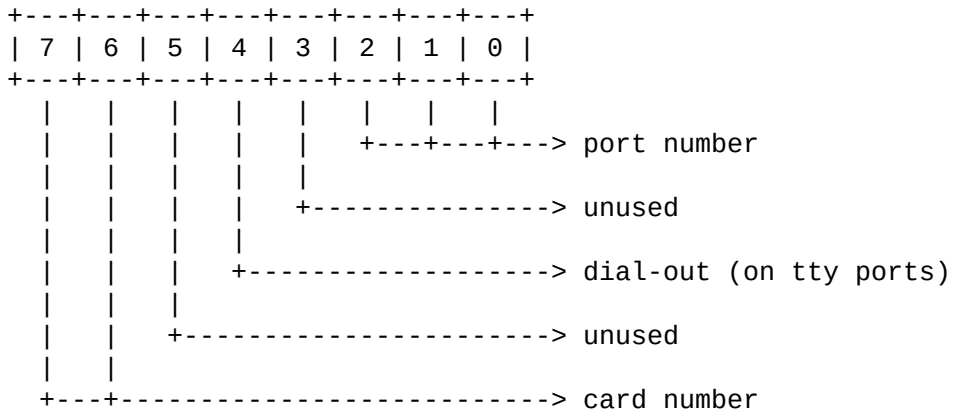
SYNOPSIS

```
spif* at sbus? slot ? offset ?          ( sun4c/sun4m )
stty* at spif?                          ( sun4c/sun4m )
sbpp* at spif?                          ( sun4c/sun4m )
```

DESCRIPTION

The **spif** driver provides support for the Sun Serial/Parallel Interface card (Sun part number 501-1931) based around the Cirrus Logic CD180 octal serial controller and the Cirrus Logic PPC2 parallel port controller.

The device minor numbers for this driver are encoded as follows:



Up to four cards are supported in the system.

Each of the serial ports has an 8 byte FIFO for receive and transmit as well as automatic hardware (RTS/CTS) flow control.

FILES

/dev/tty[jkl][0-7]	Serial ports
/dev/sbpp[0-3]	Parallel ports

DIAGNOSTICS

spif%d: ccr timeout A timeout occurred while writing to one of the CD180 registers.

stty%d-%d: ring overflow Incoming characters were discarded because the application in control of the device did not read the input fast enough.

SEE ALSO

intro(4), sbus(4), tty(4)

HISTORY

The **spif** driver was first supported in OpenBSD 2.5.

AUTHORS

The driver was written by Jason Wright ⟨jason@thought.net⟩, and is heavily based on the magma(4) driver written by Iain Hibbert.

TODO

The parallel port is not supported yet.

Dial-out (cua) devices are not yet supported.

NAME

sq — SEEQ 8003/80c03 Ethernet driver

SYNOPSIS

sq* **at hpc? offset ?**

DESCRIPTION

The **sq** interface provides access to a Ethernet network via the SEEQ 8003 and 80c03 (aka SGI EDLC) chip sets. DMA is provided by hpc(4).

The **sq** is found in the Personal Iris 4D/3x, Indigo, Indy, Challenge S, Challenge M, and Indigo2 machines, as well as the SGI E++ GIO32bis Ethernet adapter.

SEE ALSO

arp(4), hpc(4), ifmedia(4), mii(4), netintro(4), ifconfig(8)

HISTORY

The **sq** driver first appeared in NetBSD 1.6 with support for hpc(4) revision 3. Revision 1 and 1.5 support was added in NetBSD 2.0.

NAME

sqphy — Driver for Seeq 80220/80221, 80223, and 80225 10/100 Ethernet PHYs

SYNOPSIS

```
sqphy* at mii? phy ?
```

DESCRIPTION

The **sqphy** driver supports the Seeq 80220/80221, 80223, and 80225 10/100 Ethernet PHYs. The 80223 is a 3.3 volt version of the 80221. The 80225 is a stripped down version of the 80223. These PHYs are found on a variety of high-performance Ethernet interfaces.

SEE ALSO

`ifmedia`(4), `intro`(4), `mii`(4), `ifconfig`(8)

NAME

ss — SCSI scanner driver

SYNOPSIS

ss* **at scsibus? target ? lun ?**

DESCRIPTION

The **ss** driver provides support for SCSI bus based document scanners.

FILES

/dev/ss*u*	SCSI bus scanner unit *u*
/dev/nss*u*	SCSI bus scanner unit *u* − no rewind.
/dev/enss*u*	SCSI bus scanner unit *u* − ioctl(2) operations only.

SEE ALSO

scsi(4), uk(4)

AUTHORS

Kenneth Stailey, Joachim Koenig

BUGS

Only a limited number of HP Scanjet and Mustek scanners are supported. Other scanners can probably be accessed using the SCSI uk(4) driver instead.

NAME

st — SCSI/ATAPI tape driver

SYNOPSIS

```
st* at scsibus? target ? lun ?
st1 at scsibus0 target 4 lun 0
st* at atapibus? drive ? flags 0x0000
```

DESCRIPTION

The **st** driver provides support for SCSI and Advanced Technology Attachment Packet Interface (ATAPI) tape drives. It allows a tape drive to be run in several different modes depending on minor numbers and supports several different 'sub-modes'. The device can have both a *raw* interface and a *block* interface; however, only the raw interface is usually used (or recommended).

SCSI and ATAPI devices have a relatively high level interface and talk to the system via a SCSI or ATAPI adapter and a SCSI or ATAPI adapter driver (e.g. `ahc`(4), `pciide`(4)). A SCSI or ATAPI adapter must also be separately configured into the system before a SCSI or ATAPI tape can be configured.

As the SCSI or ATAPI adapter is probed during boot, the SCSI or ATAPI bus is scanned for devices. Any devices found which answer as '*Sequential*' type devices will be attached to the **st** driver.

MOUNT SESSIONS

The **st** driver is based around the concept of a "*mount session*", which is defined as the period between the time that a tape is mounted, and the time when it is unmounted. Any parameters set during a mount session remain in effect for the remainder of the session or until replaced. The tape can be unmounted, bringing the session to a close in several ways. These include:

1. Closing an 'unmount device', referred to as sub-mode 00 below. An example is `/dev/rst0`.
2. Using the `MTOFFL` `ioctl`(2) command, reachable through the '**offline**' command of `mt`(1).
3. Opening a different mode will implicitly unmount the tape, thereby closing off the mode that was previously mounted. All parameters will be loaded freshly from the new mode (See below for more on modes).

MODES AND SUB-MODES

There are several different 'operation' modes. These are controlled by bits 2 and 3 of the minor number and are designed to allow users to easily read and write different formats of tape on devices that allow multiple formats. The parameters for each mode can be set individually by hand with the `mt`(1) command. When a device corresponding to a particular mode is first mounted, The operating parameters for that mount session are copied from that mode. Further changes to the parameters during the session will change those in effect for the session but not those set in the operation mode. To change the parameters for an operation mode, one must compile them into the "*quirk*" table in the driver's source code.

In addition to the operating modes mentioned above, bits 0 and 1 of the minor number are interpreted as 'sub-modes'. The sub-modes differ in the action taken when the device is closed:

00 A close will rewind the device; if the tape has been written, then a file mark will be written before the rewind is requested. The device is unmounted.

01 A close will leave the tape mounted. If the tape was written to, a file mark will be written. No other head positioning takes place. Any further reads or writes will occur directly after the last read, or the written file mark.

10 A close will rewind the device. If the tape has been written, then a file mark will be written before the rewind is requested. On completion of the rewind an unload command will be issued. The device is unmounted.

11 This is Control mode, which allows the tape driver to be opened without a tape inserted to allow various ioctls (e.g. MTIOCGET or MTIOCTOP to set density or blocksize) and raw SCSI command on through. I/O can be done in this mode, if desired, with the same rewind/eject behaviour as mode 01. This isn't really an 'action taken on close' type of distinction, but this seems to be the place to put this mode.

BLOCKING MODES

SCSI tapes may run in either '*variable*' or '*fixed*' block-size modes. Most QIC-type devices run in fixed block-size mode, where most nine-track tapes and many new cartridge formats allow variable block-size. The difference between the two is as follows:

Variable block-size Each write made to the device results in a single logical record written to the tape. One can never read or write *part* of a record from tape (though you may request a larger block and read a smaller record); nor can one read multiple blocks. Data from a single write is therefore read by a single read. The block size used may be any value supported by the device, the SCSI adapter and the system (usually between 1 byte and 64 Kbytes, sometimes more).

When reading a variable record/block from the tape, the head is logically considered to be immediately after the last item read, and before the next item after that. If the next item is a file mark, but it was never read, then the next process to read will immediately hit the file mark and receive an end-of-file notification.

Fixed block-size Data written by the user is passed to the tape as a succession of fixed size blocks. It may be contiguous in memory, but it is considered to be a series of independent blocks. One may never write an amount of data that is not an exact multiple of the blocksize. One may read and write the same data as a different set of records, In other words, blocks that were written together may be read separately, and vice-versa.

If one requests more blocks than remain in the file, the drive will encounter the file mark. Because there is some data to return (unless there were no records before the file mark), the read will succeed, returning that data. The next read will return immediately with an EOF (as above, if the file mark is never read, it remains for the next process to read if in no-rewind mode).

FILE MARK HANDLING

The handling of file marks on write is automatic. If the user has written to the tape, and has not done a read since the last write, then a file mark will be written to the tape when the device is closed. If a rewind is requested after a write, then the driver assumes that the last file on the tape has been written, and ensures that there are two file marks written to the tape. The exception to this is that there seems to be a standard (which we follow, but don't understand why) that certain types of tape do not actually write two file marks to tape, but when read, report a 'phantom' file mark when the last file is read. These devices include the QIC family of devices (it might be that this set of devices is the same set as that of fixed block devices. This has not been determined yet, and they are treated as separate behaviors by the driver at this time).

EOM HANDLING

Attempts to write past EOM and how EOM is reported are handled slightly differently based upon whether EARLY WARNING recognition is enabled in the driver.

If EARLY WARNING recognitions is *not* enabled, then detection of EOM (as reported in SCSI Sense Data with an EOM indicator) causes the write operation to be flagged with I/O error (EIO). This has the effect for the user application of not knowing actually how many bytes were read (since the return of the `read`(2) system call is set to –1).

If EARLY WARNING recognition *is* enabled, then detection of EOM (as reported in SCSI Sense Data with an EOM indicator) has no immediate effect except that the driver notes that EOM has been detected. If the write completing didn't transfer all data that was requested, then the residual count (counting bytes *not* written) is returned to the user application. In any event, the next attempt to write (if that is the next action the user application takes) is immediately completed with no data transferred, and a residual returned to the user application indicating that no data was transferred. This is the traditional UNIX EOF indication. The state that EOM had been seen is then cleared.

In either mode of operation, the driver does not prohibit the user application from writing more data, if it chooses to do so. This will continue up until the physical end of media, which is usually signalled internally to the driver as a CHECK CONDITION with the Sense Key set to VOLUME OVERFLOW. When this or any otherwise unhandled error occurs, an error return of EIO will be transmitted to the user application. This does indeed mean that if EARLY WARNING is enables and the device continues to set EOM indicators prior to hitting physical end of media, that an indeterminate number of 'short write returns' as described in the previous paragraph will occur. However, the expected user application behaviour (in common with other systems) is to close the tape and rewind and request another tape upon the receipt of the first EOM indicator, possibly after writing one trailer record.

KERNEL CONFIGURATION

Because different tape drives behave differently, there is a mechanism within the source to **st** to quickly and conveniently recognize and deal with brands and models of drive that have special requirements.

There is a table (called the "*quirk table*") in which the identification strings of known errant drives can be stored. Alongside each is a set of flags that allows the setting of densities and blocksizes for each of the modes, along with a set of 'QUIRK' flags that can be used to enable or disable sections of code within the driver if a particular drive is recognized.

IOCTLS

The following `ioctl`(2) calls apply to SCSI tapes. Some also apply to other tapes. They are defined in the header file ⟨`sys/mtio.h`⟩.

`MTIOCGET` `(struct mtget)` Retrieve the status and parameters of the tape. Error status and residual is unlatched and cleared by the driver when it receives this ioctl.

`MTIOCTOP` `(struct mtop)` Perform a multiplexed operation. The argument structure is as follows:

```
struct mtop {
        short   mt_op;
        daddr_t mt_count;
};
```

The following operation values are defined for *mt_op*:

`MTWEOF` Write *mt_count* end of file marks at the present head position.

`MTFSF` Skip over *mt_count* file marks. Leave the head on the EOM side of the last skipped file mark.

`MTBSF` Skip *backwards* over *mt_count* file marks. Leave the head on the BOM (beginning of media) side of the last skipped file mark.

`MTFSR` Skip forwards over *mt_count* records.

`MTBSR` Skip backwards over *mt_count* records.

`MTREW` Rewind the device to the beginning of the media.

`MTOFFL` Rewind the media (and, if possible, eject). Even if the device cannot eject the media it will often no longer respond to normal requests.

`MTNOP` No-op; set status only.

`MTERASE` Erase the media from current position. If the field *mt_count* is nonzero, a full erase is done (from current position to end of media). If *mt_count* is zero, only an erase gap is written. It is hard to say which drives support only one but not the other option

`MTCACHE` Enable controller buffering.

`MTNOCACHE` Disable controller buffering.

`MTSETBSIZ` Set the blocksize to use for the device/mode. If the device is capable of variable blocksize operation, and the blocksize is set to 0, then the drive will be driven in variable mode. This parameter is in effect for the present mount session only, unless the device was opened in Control Mode (in which case this set value persists until a reboot).

`MTSETDNSTY` Set the density value (see `mt`(1)) to use when running in the mode opened (minor bits 2 and 3). This parameter is in effect for the present mount session only, unless the device was opened in Control Mode (in which case this set value persists until a reboot). Any byte sized value may be specified. Note that only a very small number of them will actually usefully work. The rest will cause the tape drive to spit up.

`MTCMPRESS` Enable or disable tape drive data compression. Typically tape drives will quite contentedly ignore settings on reads, and will probably keep you from changing density for writing anywhere but BOT.

`MTEWARN` Enable or disable EARLY WARNING at EOM behaviour (using the count as a boolean value).

`MTIOCRDSPOS`
(`uint32_t`) Read device logical block position. Not all drives support this option.

`MTIOCRDHPOS`
(`uint32_t`) Read device hardware block position. Not all drives support this option.

`MTIOCSLOCATE`
(`uint32_t`) Position the tape to the specified device logical block position.

`MTIOCHLOCATE`
(`uint32_t`) Position the tape to the specified hardware block position. Not all drives support this option.

FILES

`/dev/[n][e]rst[0-9]`	general form:
`/dev/rst0`	Mode 0, Rewind on close
`/dev/nrst0`	Mode 1, No rewind on close
`/dev/erst0`	Mode 2, Eject on close (if capable)
`/dev/enrst0`	Mode 3, Control Mode (elsewise like mode 0)

SEE ALSO

`mt`(1), `intro`(4), `mtio`(4), `scsi`(4)

HISTORY

This **st** driver was originally written for Mach 2.5 by Julian Elischer, and was ported to NetBSD by Charles Hannum. This man page was edited for NetBSD by Jon Buller.

BUGS

The selection of compression could possibly also be usefully done as with a minor device bit.

NAME

ste — Sundance ST-201 10/100 Ethernet driver

SYNOPSIS

ste* **at pci? dev ? function ?**

Configuration of PHYs may also be necessary. See mii(4).

DESCRIPTION

The **ste** device driver supports the Sundance ST-201 10/100 Ethernet chip. This chip is found on the D-Link DFE-550TX, the quad-port D-Link DFE-580TX, and some other mid-range 10/100 Ethernet boards.

SEE ALSO

arp(4), ifmedia(4), mii(4), netintro(4), pci(4), ifconfig(8)

HISTORY

The **ste** driver first appeared in NetBSD 1.6.

AUTHORS

The **ste** driver was written by Jason R. Thorpe ⟨thorpej@NetBSD.org⟩.

NAME

stf — 6to4 tunnel interface

SYNOPSIS

```
pseudo-device stf
```

DESCRIPTION

The **stf** interface supports "6to4" IPv6 in IPv4 encapsulation. It can tunnel IPv6 traffic over IPv4, as specified in RFC3056. **stf** interfaces are dynamically created and destroyed with the ifconfig(8) **create** and **destroy** subcommands. Only one **stf** interface may be created.

For ordinary nodes in 6to4 sites, you do not need a **stf** interface. The **stf** interface is only necessary on the site border router (called the "6to4 router" in the specification).

Due to the way the 6to4 protocol is specified, **stf** interfaces require certain configuration to work properly. A single (no more than one) valid 6to4 address needs to be configured on the interface. "A valid 6to4 address" is an address which has the following properties. If any of the following properties are not satisfied, **stf** raises a runtime error on packet transmission. Read the specification for more details.

- matches 2002:xxyy:zzuu::/48, where xxyy:zzuu is the hexadecimal notation of an IPv4 address for the node. The IPv4 address used can be taken from any interface your node has. Since the specification forbids the use of IPv4 private address, the address needs to be a global IPv4 address.
- Subnet identifier portion (48th to 63rd bit) and interface identifier portion (lower 64 bits) are properly filled to avoid address collisions.

If you would like the node to behave as a relay router, the prefix length for the IPv6 interface address needs to be 16 so that the node would consider any 6to4 destination as "on-link". If you would like to restrict 6to4 peers to be inside a certain IPv4 prefix, you may want to configure the IPv6 prefix length to be "16 + IPv4 prefix length". The **stf** interface will check the IPv4 source address on packets if the IPv6 prefix length is larger than 16.

stf can be configured to be ECN (Explicit Congestion Notification) friendly. This can be configured by IFF_LINK1. See gif(4) for details.

Please note that the 6to4 specification is written as an "accept tunneled packet from everyone" tunneling device. By enabling the **stf** device, you are making it much easier for malicious parties to inject fabricated IPv6 packets to your node. Also, malicious parties can inject IPv6 packets with fabricated source addresses to make your node generate improper tunneled packets. Administrators must be cautious when enabling the interface. To prevent possible attacks, the **stf** interface filters out the following packets (note that the checks are in no way complete):

- Packets with IPv4 unspecified addresses as outer IPv4 source/destination (0.0.0.0/8)
- Packets with the loopback address as outer IPv4 source/destination (127.0.0.0/8)
- Packets with IPv4 multicast addresses as outer IPv4 source/destination (224.0.0.0/4)
- Packets with limited broadcast addresses as outer IPv4 source/destination (255.0.0.0/8)
- Packets with private addresses as outer IPv4 source/destination (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16)
- Packets with IPv4 link-local addresses as outer IPv4 source/destination (169.254.0.0/16)
- Packets with subnet broadcast addresses as outer IPv4 source/destination. The check is made against subnet broadcast addresses for all of the directly connected subnets.

- Packets that do not pass ingress filtering. Outer IPv4 source addresses must meet the IPv4 topology on the routing table. Ingress filtering can be turned off by `IFF_LINK2` bit.
- The same set of rules are applied against the IPv4 address embedded into the inner IPv6 address, if the IPv6 address matches the 6to4 prefix.
- Packets with site-local or link-local unicast addresses as inner IPv6 source/destination
- Packets with node-local or link-local multicast addresses as inner IPv6 source/destination

It is recommended to filter/audit incoming IPv4 packets with IP protocol number 41, as necessary. It is also recommended to filter/audit encapsulated IPv6 packets as well. You may also want to run normal ingress filtering against inner IPv6 addresses to avoid spoofing.

By setting the `IFF_LINK0` flag on the **`stf`** interface, it is possible to disable the input path, making direct attacks from the outside impossible. Note, however, that other security risks exist. If you wish to use the configuration, you must not advertise your 6to4 addresses to others.

EXAMPLES

Note that `8504:0506` is equal to `133.4.5.6`, written in hexadecimal.

```
# ifconfig ne0 inet 133.4.5.6 netmask 0xffffff00
# ifconfig stf0 create inet6 2002:8504:0506:0000:a00:5aff:fe38:6f86 \
        prefixlen 16 alias
```

The following configuration accepts packets from IPv4 source address `9.1.0.0/16` only. It emits 6to4 packets only for IPv6 destination 2002:0901::/32 (IPv4 destination will match `9.1.0.0/16`).

```
# ifconfig ne0 inet 9.1.2.3 netmask 0xffff0000
# ifconfig stf0 create inet6 2002:0901:0203:0000:a00:5aff:fe38:6f86 \
        prefixlen 32 alias
```

The following configuration uses the **`stf`** interface as an output-only device. You need to have alternative IPv6 connectivity (other than 6to4) to use this configuration. For outbound traffic, you can reach other 6to4 networks efficiently via **`stf`**. For inbound traffic, you will not receive any 6to4-tunneled packets (less security drawbacks). Be careful not to advertise your 6to4 prefix to others (`2002:8504:0506::/48`), and not to use your 6to4 prefix as a source address.

```
# ifconfig ne0 inet 133.4.5.6 netmask 0xffffff00
# ifconfig stf0 create inet6 2002:8504:0506:0000:a00:5aff:fe38:6f86 \
        prefixlen 16 alias deprecated link0
# route add -inet6 2002:: -prefixlen 16 ::1 -ifp stf0
```

SEE ALSO

`gif`(4), `inet`(4), `inet6`(4)

`http://www.6bone.net/6bone_6to4.html`

Brian Carpenter and Keith Moore, *Connection of IPv6 Domains via IPv4 Clouds*, RFC, 3056, February 2001.

F. Baker and P. Savola, *Ingress Filtering for Multihomed Networks*, RFC, 3704, March 2004.

Jun-ichiro itojun Hagino, *Possible abuse against IPv6 transition technologies*, draft-itojun-ipv6-transition-abuse-01.txt, July 2000, work in progress.

HISTORY

The **stf** device first appeared in WIDE/KAME IPv6 stack.

BUGS

No more than one **stf** interface is allowed for a node, and no more than one IPv6 interface address is allowed for an **stf** interface. This is to avoid source address selection conflicts between the IPv6 layer and the IPv4 layer, and to cope with ingress filtering rules on the other side. This is a feature to make **stf** work right for all occasions.

NAME

stge — Sundance/Tamarack TC9021 Gigabit Ethernet driver

SYNOPSIS

stge* **at pci? dev ? function ?**

Configuration of PHYs or Ten-bit interfaces may also be necessary. See mii(4).

DESCRIPTION

The **stge** device driver supports the Sundance/Tamarack TC9021 Gigabit Ethernet chip.

The Sundance/Tamarack TC9021 is found on the D-Link DGE-550T, and the Antares Microsystems Gigabit Ethernet board. It uses an external PHY or an external 10-bit interface.

The TC9021 supports IPv4/TCP/UDP checksumming in hardware. The **stge** driver supports this feature of the chip. See ifconfig(8) for information on how to enable this feature.

SEE ALSO

arp(4), ifmedia(4), mii(4), netintro(4), pci(4), ifconfig(8)

HISTORY

The **stge** driver first appeared in NetBSD 1.6.

AUTHORS

The **stge** driver was written by Jason R. Thorpe ⟨thorpej@NetBSD.org⟩.

BUGS

The **stge** driver does not support the VLAN tag insertion/removal feature of the TC9021 chip. This is primarily because the TC9021's VLAN do not have a useful programming interface.

The **stge** driver does not yet support jumbo Ethernet frames.

The **stge** driver does not yet function properly with 1000BASE-T fitted boards. Currently, only 1000BASE-SX boards work.

NAME

sti — HP Standard Text Interface

SYNOPSIS

```
sti*  at mainbus0 irq ?
sti*  at phantomas0 irq ?
wsdisplay*  at sti? console ?
```

DESCRIPTION

The **sti** was created by HP to provide uniform frame-buffer access operations for their 9000/300 and 9000/700 series of workstations.

The following models are supported (though not all features or frame buffer depths may be available):

Model	Bits	Mem	3D	Machines/Cards
GRX	8g	2		SGC
CRX	8	2		SGC
Tomcat	8	2		SGC
Stinger	8	2		HP9000/7[12]5/74[257]i
Artist	8	2		HP9000/712/7[12]5/74[38]i
CRX-24	24	16		SGC
HCRX-8	8	2		GSC
HCRX-24	24	16		GSC
Visualize EG	16	2		HP B/C-class, GSC/PCI

Implementation consists of a set of functions burnt in to the PROM on the card and providing the following set of functions (see below for PROM revision history on functions supported by particular PROM revision):

- Initialize graphics.
- State management.
- Print a character onto the screen using currently selected font.
- Copy a region of the frame-buffer to another location.
- Self testing.
- Exception handling.
- Frame-buffer configuration enquiry.
- Setting colour-map entry.
- DMA parameters.
- Flow control.
- User timing.
- Processing management.
- Miscellaneous utility functions.

There are two modes for accessing the PROM: “byte” and “word” mode. In “byte” mode each 4-byte word contains only the low-ordered big-endian byte of data; i.e., to compose one word of data 4 words should be read and low-ordered bytes of those should be shifted correspondingly. In “word” mode each word contains all 4 bytes of valid data.

PROM revision history:

8.02

Original release.

8.03

- OSF-extended self test (a.k.a fast).

- Restore display.

8.04
- Implement **curr_mon** function.
- Graphical boot screen.
- Implement “block move”.
- Implement “set colour-map entry”. **sti** Implement word mode.
- Support for multiple monitors.
- Support **user_data** **sti** space usage.
- Support for extra memory.
- Support for **Windows NT (tm)**.
- Monitor frequency reference.
- Early console.
- Support added for: PCXL, **GSC** bus, ROM-less operation.

8.05
- Interrupt support.
- Report card’s power usage.
- Birds of Prey.
- User interrupts.

8.06
- Multiple fonts.
- Monitor table descriptor strings.
- PCXL2 and PCXU monitor descriptors.

8.08
- HP-UX 10 support for Visualize FX
- **dma_ctrl** function added.
- **flow_ctrl** function added.
- **user_timing** function added.

8.09
- Addition changes for **Visualize FX** due to rearchitecture for performance.
- **process_mgr** function added.

8.0a
PCXL2 and PCXU dual **PCI** EPROM map mode, implemented on **Visualize EG**.

8.0b
Support for HP-UX non-implicit locking DMA, implemented on **Visualize FXE**.

8.0c
sti_util function added (flashing under HP-UX and other sideband traffic).

8.0d
Colour frame buffer support.

SEE ALSO

intro(4), phantomas(4), wsdisplay(4)

Standard Text Interface For Graphics Devices, *Hewlett-Packard*, Revision 8.13, March 1, 2000.

HISTORY

The **sti** driver was written by Michael Shalayeff ⟨mickey@openbsd.org⟩ for HPPA port for OpenBSD 2.7.

BUGS

Currently, neither scroll back nor screen blanking functions are implemented.

NAME

stpcide — STMicroelectronics STPC IDE disk controllers driver

SYNOPSIS

```
stpcide* at pci? dev ? function ? flags 0x0000
```

DESCRIPTION

The **stpcide** driver supports the STMicroelectronics STPC x86 SoC internal IDE controllers, and provides the interface with the hardware for the ata(4) driver. The driver features DMA mode 2 and PIO mode 4 transfer speeds.

The 0x0002 flag forces the **stpcide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

BUGS

The timings used for the DMA and PIO modes are for STPC Atlas and its siblings with their PCI clock configured at 33 MHz. Other speeds including the STPC Vega Ultra-IDE controller will need adjustments.

NAME

strip — Metricom Ricochet packet radio wireless network device

SYNOPSIS

```
pseudo-device strip
```

DESCRIPTION

The **strip** driver takes outbound network packets, encapsulates them using the Metricom "star mode" framing, and sends the packets out an RS-232 interface to a Metricom *Ricochet* packet radio. Packets arriving from the packet radio via the serial link are decapsulated and then passed up to the local host's networking stack.

strip is an acronym for **ST**armode **R**adio **IP**.

The **strip** interfaces can be created by using the `ifconfig`(8) **create** command. Each **strip** interface is a pseudo-device driver for the Metricom *Ricochet* packet radio, operating in peer-to-peer packet mode.

In many ways, the **strip** driver is very much like the `sl`(4) SLIP pseudo-device driver. A **strip** device is attached to a tty line with `slattach`(8). Once attached, the interface is configured via `ifconfig`(8). The major difference between the `sl`(4) SLIP pseudo-device driver and the **strip** driver is that SLIP works only between two hosts over a dedicated point-to-point connection.

In contrast, **strip** sends packets to a frequency-hopping packet radio, which can address packets to any peer Metricom *Ricochet* packet radio, rather than just to a single host at the other end of a point-to-point line. Thus, one **strip** pseudo-device is usually sufficient for any kernel.

In other respects, a **strip** interface is rather like an Ethernet interface. Packets are individually addressed, and subsequent packets can be sent independently to different MAC addresses. However, the "star mode" framing and MAC addressing are not in any way compatible with Ethernet. Broadcast or multicast to more than one packet radio is not possible, due to the independent frequency-hopping operation of the packet radios. The interface flags `IFF_POINTOPOINT` and `IFF_BROADCAST` are not supported on the **strip** interface.

In other words, **strip** implements a multiple-access, non-broadcast device, accessed via an RS-232 serial line, using a proprietary packet framing scheme.

This version of the **strip** driver maps IP addresses to Metricom *Ricochet* packet radio addresses using statically configured entries in the normal routing table. These entries map IP addresses of peer packet radios to the MAC-level addresses. The exact syntax of this mapping and an example are discussed below. The Internet Assigned Numbers Authority (IANA) has allocated an ARP type code for use with STRIP. A future version of this driver will support `arp`(4) to obtain the IP address of reachable peer packet radios dynamically.

ADDRESS CONFIGURATION

This version of the STRIP driver requires static pre-configuration of the mapping from IP addresses to packet radio MAC addresses. The `route`(8) command should be used to bind a peer STRIP host's packet radio IP address to the peer's link-level packet radio address.

Radio addresses are encoded using the hex equivalent of the packet radio's decimal ASCII address. For example, the following route command will configure a routing entry to a packet radio with a MAC address of 1234-5678, and an IP address 10.11.12.13, reachable via the *strip0* interface:

```
route add -host 10.11.12.13 -link strip0:1:2:3:4:5:6:7:8
```

Generalising from this example to other IP addresses and to other 8-digit MAC addresses should be clear.

RADIO CONFIGURATION

The Metricom *Ricochet* packet radios can auto-baud at speeds up to 38.4K baud. At higher speeds -- 57600 or 115200 -- the packet radio cannot autobaud. When running at high speeds, the packet radio's serial port should be manually configured to permanently run at the desired speed. Use a terminal emulator and the Hayes command **ATS304=115200** to set the serial baudrate to the specified number (or 0 for autobaud). The command **AT&W** will then save the current packet radio state in non-volatile memory.

Metricom *Ricochet* packet radios can operate in either "modem-emulation" mode or in packet mode (i.e. "star mode"). The **strip** driver automatically detects if the packet radio has fallen out of "star mode", and resets it back into "star mode", if the baud rate was set correctly by `slattach`(8).

SEE ALSO

`arp`(4), `inet`(4), `sl`(4), `ifconfig`(8), `route`(8), `slattach`(8)

HISTORY

strip was originally developed for the Linux kernel by Stuart Cheshire of Stanford's Operating Systems and Networking group, as part of Mary Baker's MosquitoNet **http://mosquitonet.stanford.edu/mosquitonet.html** project.

This **strip** driver was ported to NetBSD by Jonathan Stone at Stanford's Distributed Systems Group and first distributed with NetBSD 1.2.

BUGS

Currently, **strip** is IP-only. Encapsulations for AppleTalk and ARP have been defined, but are not yet implemented in this driver.

strip has not been widely tested on a variety of lower-level serial drivers.

The detection and resetting of packet radios that crash out of "star mode" does not always work in this version of the driver. One workaround is to kill the `slattach`(8) process, `ifconfig`(8) the **strip** interface down, and then start a new slattach and rerun ifconfig to enable the interface again.

NAME

stuirda — Sigmaltel 4116/4220 USB-IrDA bridge support

SYNOPSIS

```
stuirda*   at uhub?
irframe* at uirda?
```

DESCRIPTION

The **stuirda** driver provides support for SigmaTels USB-IrDA bridges that nearly follow the bridge specification:

SigmaTel ST4116
SigmaTel ST4210
SigmaTel ST4220

Access to the IrDA device is through the irframe(4) driver. Before accessing it, a firmware patch must be downloaded. It can currently be found at: http://sigmatel.oak9.com/documents/stir4210_4220_4116_patch_files.tar.gz

Unpack the archive with

tar -C /libdata/firmware/uirda/ **-xzf** *stir4210_4220_4116_patch_files.tar.gz*

BUGS

stuirda needs the firmware for configuration, so it can't be loaded at kernel startup time. As a workaround, you can plug in the USB-IrDA-bridge after booting.

SEE ALSO

irframe(4), usb(4)

HISTORY

The **stuirda** driver appeared in NetBSD 5.0.

NAME

sv — S3 SonicVibes audio device driver

SYNOPSIS

```
sv*    at pci? dev ? function ?
audio* at audiobus?
opl*   at sv?
```

DESCRIPTION

The **sv** driver provides support sound cards based on the S3 SonicVibes chip, e.g. the Turtle Beach Daytona card.

SEE ALSO

audio(4), opl(4), pci(4)

HISTORY

The **sv** device driver appeared in NetBSD 1.4.

BUGS

The waveform synth and MIDI port are not supported.

NAME

svwsata — Serverworks Serial ATA disk controller driver

SYNOPSIS

```
svwsata* at pci? dev ? function ? flags 0x0000
```

DESCRIPTION

The **svwsata** driver supports the Serverworks K2, Frodo4, Frodo8 and HT-1000 Serial ATA controllers, and provides the interface with the hardware for the ata(4) driver.

The 0x0002 flag forces the **svwsata** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the SATA controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

NAME

swwdog — software watchdog timer

SYNOPSIS

pseudo-device swwdog [*count*]

DESCRIPTION

The **swwdog** driver provides a software watchdog timer that works with `wdogctl`(8). If the timer expires, the system reboots unless the variable *swwdog_panic* is non-zero; if it is, the system will panic instead.

The default period of **swwdog** is 60 seconds.

SEE ALSO

`wdogctl`(8)

HISTORY

The **swwdog** driver was written by Steven M. Bellovin.

BUGS

Although more than one **swwdog** timer can be configured, it's a pointless thing to do, since only one watchdog timer can be active at any given time. Arguably, this is a bug in the watchdog timer framework.

Kernel tickle mode is useless with **swwdog** and arguably should be rejected, since both it and this driver rely on the same callout mechanism; if one is blocked, almost certainly the other is as well.

The alarm option to `wdogctl`(8) isn't implemented.

NAME

tap — virtual Ethernet device

SYNOPSIS

```
pseudo-device tap
```

DESCRIPTION

The **tap** driver allows the creation and use of virtual Ethernet devices. Those interfaces appear just as any real Ethernet NIC to the kernel, but can also be accessed by userland through a character device node in order to read frames being sent by the system or to inject frames.

In that respect it is very similar to what tun(4) provides, but the added Ethernet layer allows easy integration with machine emulators or virtual Ethernet networks through the use of bridge(4) with tunneling.

INTERFACE CREATION

Interfaces may be created in two different ways: using the ifconfig(8) **create** command with a specified device number, or its ioctl(2) equivalent, SIOCIFCREATE, or using the special cloning device /dev/tap.

The former works the same as any other cloning network interface: the administrator can create and destroy interfaces at any time, notably at boot time. This is the easiest way of combining **tap** and bridge(4). Later, userland will actually access the interfaces through the specific device nodes /dev/tapN.

The latter is aimed at applications that need a virtual Ethernet device for the duration of their execution. A new interface is created at the opening of /dev/tap, and is later destroyed when the last process using the file descriptor closes it.

CHARACTER DEVICES

Whether the **tap** devices are accessed through the special cloning device /dev/tap or through the specific devices /dev/tapN, the possible actions to control the matching interface are the same.

When using /dev/tap though, as the interface is created on-the-fly, its name is not known immediately by the application. Therefore the TAPGIFNAME ioctl is provided. It should be the first action an application using the special cloning device will do. It takes a pointer to a *struct ifreq* as an argument.

Ethernet frames sent out by the kernel on a **tap** interface can be obtained by the controlling application with read(2). It can also inject frames in the kernel with write(2). There is absolutely no validation of the content of the injected frame, it can be any data, of any length.

One call of write(2) will inject a single frame in the kernel, as one call of read(2) will retrieve a single frame from the queue, to the extent of the provided buffer. If the buffer is not large enough, the frame will be truncated.

tap character devices support the FIONREAD ioctl which returns the size of the next available frame, or 0 if there is no available frame in the queue.

They also support non-blocking I/O through the FIONBIO ioctl. In that mode, EWOULDBLOCK is returned by read(2) when no data is available.

Asynchronous I/O is supported through the FIOASYNC, FIOSETOWN, and FIOGETOWN ioctls. The first will enable SIGIO generation, while the two other configure the process group that will receive the signal when data is ready.

Synchronisation may also be achieved through the use of select(2), poll(2), or kevent(2).

ETHERNET ADDRESS

When a **tap** device is created, it is assigned an Ethernet address of the form f2:0b:a4:xx:xx:xx. This address can later be changed in two ways: through a sysctl node, or an ioctl call.

The sysctl node is net.link.tap.<iface>. Any string of six colon-separated hexadecimal numbers will be accepted. Reading that node will provide a string representation of the current Ethernet address.

The address can also be changed with the `SIOCSIFPHYADDR` ioctl, which is used the same way as with `gif`(4). The difference is in the family of the address which is passed inside the *`struct ifreqalias`* argument, which should be set to `AF_LINK`. This ioctl call should be made on a socket, as it is not available on the ioctl handler of the character device interface.

FILES

`/dev/tap`	cloning device
`/dev/tap[0-9]*`	individual character device nodes

SEE ALSO

`bridge`(4), `etherip`(4), `gif`(4), `tun`(4), `ifconfig`(8)

HISTORY

The **tap** driver first appeared in NetBSD 3.0.

NAME

tc — TURBOchannel expansion bus driver

SYNOPSIS

alpha

```
tc* at tcasic?
```

pmax

```
tc* at mainbus0
```

DESCRIPTION

The **tc** driver provides machine-independent support for the DEC TURBOchannel expansion bus found on all DEC 5000-series machines with MIPS and DEC 3000-series with Alpha processors.

Your system may support additional TURBOchannel devices. Drivers for TURBOchannel devices not listed here are machine-dependent. Consult your system's intro(4) for additional information.

HARDWARE

NetBSD includes machine-independent TURBOchannel drivers, sorted by device type and driver name:

SCSI interfaces

asc	PMAZ-A single-channel SCSI adapter
tcds	PMAZ-DS, PMAZ-FS, PMAZB-AA and PMAZC-AA dual-channel SCSI adapters

Network interfaces

fta	PMAF-F DEFTA FDDI controller
le	LANCE Ethernet interface

Frame buffers

cfb	PMAG-B CX colour unaccelerated 2-D framebuffer
mfb	PMAG-A MX monochrome framebuffer
px	PMAG-C PX accelerated graphics boards
pxg	PMAG-D, PMAG-E and PMAG-F PXG accelerated graphics boards
sfb	PMAGB-BA HX colour unaccelerated 2-D framebuffer
tfb	PMAG-J TX 24-bit colour unaccelerated 2-D framebuffer

Miscellaneous

ioasic	baseboard IO control ASIC for DEC TURBOchannel systems

SEE ALSO

asc(4), cfb(4), fta(4), intro(4), ioasic(4), le(4), mfb(4), px(4), pxg(4), sfb(4), tcds(4), tfb(4)

HISTORY

The **tc** driver first appeared in NetBSD 1.1.

BUGS

The **tc** driver makes poor use of interrupt priority on the 5000/1xx series systems.

NAME

tcasic — TURBOchannel host bus support

SYNOPSIS

```
tcasic* at mainbus0
tc* at tcasic?
```

DESCRIPTION

The **tcasic** driver provides support for the TURBOchannel host bus on the baseboard of the DEC 3000/300 and 3000/500 systems.

SEE ALSO

intro(4), mainbus(4), tc(4)

NAME

tcds — TURBOchannel dual-channel SCSI adapters

SYNOPSIS

tcds* **at tc? slot ? offset ?**

DESCRIPTION

The **tcds** driver provides support for the DEC TURBOchannel PMAZ-DS, PMAZ-FS, PMAZB-AA and PMAZC-AA dual-channel SCSI adapters. Each channel is driven by the asc(4) driver. The PMAZ-FS (Alpha onboard only) and PMAZC-AA (option board) provide two Fast Narrow SCSI busses, while the PMAX-DS (Alpha onboard only) and PMAZB-AA only provide Narrow SCSI. For the onboard Alpha controllers, one bus is for internal devices and one bus for external devices.

SEE ALSO

asc(4)

NAME

tcic — Databook PCMCIA controller driver

SYNOPSIS

```
tcic0   at isa? port 0x240 iomem 0xd0000 iosiz 0x4000
pcmcia* at tcic? controller ? socket ?
```

DESCRIPTION

NetBSD provides support for the Databook DB86082, DB86084, DB86184, and DB86072 PCMCIA controllers.

SEE ALSO

isa(4), pcic(4), pcmcia(4)

HISTORY

The **tcic** driver appeared in NetBSD 1.4.

NAME

tcom — multiplexing serial communications interface

SYNOPSIS

For 4-port TC-400 series boards:

```
tcom0 at isa? port 0x100 irq 5
com2 at tcom? slave ?
com3 at tcom? slave ?
com4 at tcom? slave ?
com5 at tcom? slave ?
```

For 8-port TC-800 series boards:

```
tcom0 at isa? port 0x100 irq 5
com2 at tcom? slave ?
com3 at tcom? slave ?
com4 at tcom? slave ?
com5 at tcom? slave ?
com6 at tcom? slave ?
com7 at tcom? slave ?
com8 at tcom? slave ?
com9 at tcom? slave ?
```

DESCRIPTION

The **tcom** driver provides support for the Byte Runner Technologies TC-400 and TC-800 series boards that multiplex together up to four or eight EIA RS-232C (CCITT V.28) communications interfaces.

Each **tcom** device is the master device for up to eight **com** devices. The kernel configuration specifies these **com** devices as slave devices of the **tcom** device, as shown in the synopsis. The slave ID given for each **com** device determines which bit in the interrupt multiplexing register is tested to find interrupts for that device. The port specification for the **tcom** device is used to compute the base addresses for the **com** subdevices and the port for the interrupt multiplexing register.

Not all possible configuration options are currently supported (for example, speeds beyond 115200 baud are not currently supported).

FILES

`/dev/tty??`

SEE ALSO

com(4)

HISTORY

The **tcom** driver was written by Jukka Marin.

NAME

tcp — Internet Transmission Control Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

int
socket(AF_INET, SOCK_STREAM, 0);

int
socket(AF_INET6, SOCK_STREAM, 0);
```

DESCRIPTION

The TCP provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the SOCK_STREAM abstraction. TCP uses the standard Internet address format and, in addition, provides a per-host collection of "port addresses". Thus, each address is composed of an Internet address specifying the host and network, with a specific TCP port on the host identifying the peer entity.

Sockets using TCP are either "active" or "passive". Active sockets initiate connections to passive sockets. By default TCP sockets are created active; to create a passive socket the listen(2) system call must be used after binding the socket with the bind(2) system call. Only passive sockets may use the accept(2) call to accept incoming connections. Only active sockets may use the connect(2) call to initiate connections.

Passive sockets may "underspecify" their location to match incoming connection requests from multiple networks. This technique, termed "wildcard addressing", allows a single server to provide service to clients on multiple networks. To create a socket which listens on all networks, the Internet address INADDR_ANY must be bound. The TCP port may still be specified at this time; if the port is not specified the system will assign one. Once a connection has been established the socket's address is fixed by the peer entity's location. The address assigned the socket is the address associated with the network interface through which packets are being transmitted and received. Normally this address corresponds to the peer entity's network.

TCP supports a number of socket options which can be set with setsockopt(2) and tested with getsockopt(2):

TCP_NODELAY Under most circumstances, TCP sends data when it is presented; when outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an acknowledgement is received. For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this packetization may cause significant delays. Therefore, TCP provides a boolean option, TCP_NODELAY (from ⟨netinet/tcp.h⟩, to defeat this algorithm.

TCP_MAXSEG By default, a sender- and receiver-TCP will negotiate among themselves to determine the maximum segment size to be used for each connection. The TCP_MAXSEG option allows the user to determine the result of this negotiation, and to reduce it if desired.

TCP_MD5SIG This option enables the use of MD5 digests (also known as TCP-MD5) on writes to the specified socket. In the current release, only outgoing traffic is digested; digests on incoming traffic are not verified. The current default behavior for the system is to respond to a system advertising this option with TCP-MD5; this may change.

One common use for this in a NetBSD router deployment is to enable based routers to interwork with Cisco equipment at peering points. Support for this feature conforms to RFC 2385. Only IPv4 (AF_INET) sessions are supported.

In order for this option to function correctly, it is necessary for the administrator to add a tcp-md5 key entry to the system's security associations database (SADB) using the `setkey`(8) utility. This entry must have an SPI of 0x1000 and can therefore only be specified on a per-host basis at this time.

If an SADB entry cannot be found for the destination, the outgoing traffic will have an invalid digest option prepended, and the following error message will be visible on the system console: *tcp_signature_compute: SADB lookup failed for %d.%d.%d.%d*.

`TCP_KEEPIDLE` TCP probes a connection that has been idle for some amount of time. The default value for this idle period is 4 hours. The `TCP_KEEPIDLE` option can be used to affect this value for a given socket, and specifies the number of seconds of idle time between keepalive probes. This option takes an *`unsigned int`* value, with a value greater than 0.

`TCP_KEEPINTVL` When the `SO_KEEPALIVE` option is enabled, TCP probes a connection that has been idle for some amount of time. If the remote system does not respond to a keepalive probe, TCP retransmits the probe after some amount of time. The default value for this retransmit interval is 150 seconds. The `TCP_KEEPINTVL` option can be used to affect this value for a given socket, and specifies the number of seconds to wait before retransmitting a keepalive probe. This option takes an *`unsigned int`* value, with a value greater than 0.

`TCP_KEEPCNT` When the `SO_KEEPALIVE` option is enabled, TCP probes a connection that has been idle for some amount of time. If the remote system does not respond to a keepalive probe, TCP retransmits the probe a certain number of times before a connection is considered to be broken. The default value for this keepalive probe retransmit limit is 8. The `TCP_KEEPCNT` option can be used to affect this value for a given socket, and specifies the maximum number of keepalive probes to be sent. This option takes an *`unsigned int`* value, with a value greater than 0.

`TCP_KEEPINIT` If a TCP connection cannot be established within some amount of time, TCP will time out the connect attempt. The default value for this initial connection establishment timeout is 150 seconds. The `TCP_KEEPINIT`
option can be used to affect this initial timeout period for a given socket, and specifies the number of seconds to wait before the connect attempt is timed out. For passive connections, the `TCP_KEEPINIT` option value is inherited from the listening socket. This option takes an *`unsigned int`* value, with a value greater than 0.

The option level for the `setsockopt`(2) call is the protocol number for TCP, available from `getprotobyname`(3).

In the historical BSD TCP implementation, if the `TCP_NODELAY` option was set on a passive socket, the sockets returned by `accept`(2) erroneously did not have the `TCP_NODELAY` option set; the behavior was corrected to inherit `TCP_NODELAY` in NetBSD 1.6.

Options at the IP network level may be used with TCP; see `ip`(4) or `ip6`(4). Incoming connection requests that are source-routed are noted, and the reverse source route is used in responding.

There are many adjustable parameters that control various aspects of the NetBSD TCP behavior; these parameters are documented in `sysctl`(7), and they include:

- RFC 1323 extensions for high performance
- Send/receive buffer sizes
- Default maximum segment size (MSS)

- SYN cache parameters
- Initial window size
- Hughes/Touch/Heidemann Congestion Window Monitoring algorithm
- Keepalive parameters
- newReno algorithm for congestion control
- Logging of connection refusals
- RST packet rate limits
- SACK (Selective Acknowledgment)
- ECN (Explicit Congestion Notification)
- Congestion window increase methods; the traditional packet counting or RFC 3465 Appropriate Byte Counting

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one;
[ENOBUFS]	when the system runs out of memory for an internal data structure;
[ETIMEDOUT]	when a connection was dropped due to excessive retransmissions;
[ECONNRESET]	when the remote peer forces the connection to be closed;
[ECONNREFUSED]	when the remote peer actively refuses connection establishment (usually because no process is listening to the port);
[EADDRINUSE]	when an attempt is made to create a socket with a port which has already been allocated;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

`getsockopt`(2), `socket`(2), `sysctl`(7), `inet`(4), `inet6`(4), `intro`(4), `ip`(4), `ip6`(4)

Transmission Control Protocol, RFC, 793, September 1981.

Requirements for Internet Hosts -- Communication Layers, RFC, 1122, October 1989.

HISTORY

The **tcp** protocol stack appeared in 4.2BSD.

NAME

tctrl — Tadpole Microcontroller Interface

SYNOPSIS

```
tctrl0 at obio0
```

DESCRIPTION

The **tctrl** driver provides control over many functions on the Tadpole SPARCbook 3 series laptops, via their TS102 chip. The microcontroller is used to power the TFT display down when the laptop lid is closed and when the screen is blanked by the pnozz(4) driver. The **tctrl** is also used to power the laptop off when the reboot(2) system call is used with the *RB_POWERDOWN* flag is set. Power button events are forwarded to powerd(8). The PCMCIA part of the controller is supported by tslot(4).

SEE ALSO

reboot(2), intro(4), pnozz(4), tslot(4), powerd(8)

NAME

tcx — Sun accelerated 8/24-bit color frame buffer

SYNOPSIS

tcx* at sbus? slot ? offset ?

DESCRIPTION

The **tcx** is a memory based color frame buffer, with graphics acceleration and overlay capabilities. Its control registers, colour lookup table and pixel memory can be mapped into a user process address space by using the mmap(2) system call. The **tcx** driver supports the minimal ioctl's needed to run X(1), and can be operated in cgthree(4) emulation mode.

SEE ALSO

bwtwo(4), cgtwo(4), cgthree(4), cgfour(4), cgsix(4), cgeight(4)

NAME

teliosio — driver for the Sharp Telios LCD screen and Battery unit

SYNOPSIS

```
teliosio* at txcsbus1 regcs 13 regcswidth 16
```

DESCRIPTION

The **teliosio** driver provides support for controlling brightness and power of the LCD screen in Sharp Telios series machines.

The default keymap binds ⟨Ctrl⟩ + ⟨Alt⟩ + ⟨*arrowkey*⟩ to control brightness with left and right arrow keys, and ⟨Ctrl⟩ + ⟨Alt⟩ + ⟨space⟩ to toggle power of the LCD screen.

The current status of the battery unit can be displayed via apm(8).

SEE ALSO

screenblank(1), apm(8)

HISTORY

The **teliosio** driver first appeared in NetBSD 3.0.

NAME

termios — general terminal line discipline

SYNOPSIS

```
#include <termios.h>
```

DESCRIPTION

This describes a general terminal line discipline that is supported on tty asynchronous communication ports.

Opening a Terminal Device File

When a terminal file is opened, it normally causes the process to wait until a connection is established. For most hardware, the presence of a connection is indicated by the assertion of the hardware *CARRIER DETECT* (CD) line. If the **termios** structure associated with the terminal file has the CLOCAL flag set in the cflag, or if the O_NONBLOCK flag is set in the open(2) call, then the open will succeed even without a connection being present.

In practice, applications seldom open these files; they are opened by special programs, such as getty(8) or rlogind(8), and become an application's standard input, output, and error files.

Job Control in a Nutshell

Every process is associated with a particular process group and session. The grouping is hierarchical: every member of a particular process group is a member of the same session. This structuring is used in managing groups of related processes for purposes of *job control*; that is, the ability from the keyboard (or from program control) to simultaneously stop or restart a complex command (a command composed of one or more related processes).

The grouping into process groups allows delivering of signals that stop or start the group as a whole, along with arbitrating which process group has access to the single controlling terminal. The grouping at a higher layer into sessions is to restrict the job control related signals and system calls to within processes resulting from a particular instance of a "login".

Typically, a session is created when a user logs in, and the login terminal is set up to be the controlling terminal; all processes spawned from that login shell are in the same session, and inherit the controlling terminal. A job control shell operating interactively (that is, reading commands from a terminal) normally groups related processes together by placing them into the same process group. A set of processes in the same process group is collectively referred to as a "job".

When the foreground process group of the terminal is the same as the process group of a particular job, that job is said to be in the *foreground*. When the process group of the terminal is different than the process group of a job (but is still the controlling terminal), that job is said to be in the *background*.

Normally the shell reads a command and starts the job that implements that command. If the command is to be started in the foreground (typical), it sets the process group of the terminal to the process group of the started job, waits for the job to complete, and then sets the process group of the terminal back to its own process group (it puts itself into the foreground).

If the job is to be started in the background (as denoted by the shell operator "&"), it never changes the process group of the terminal and doesn't wait for the job to complete (that is, it immediately attempts to read the next command).

If the job is started in the foreground, the user may type a character (usually '^Z') which generates the terminal stop signal (SIGTSTP) and has the affect of stopping the entire job. The shell will notice that the job stopped (see wait(2)), and will resume running after placing itself in the foreground.

The shell also has commands for placing stopped jobs in the background, and for placing stopped or background jobs into the foreground.

Orphaned Process Groups

An orphaned process group is a process group that has no process whose parent is in a different process group, yet is in the same session. Conceptually it means a process group that doesn't have a parent that could do anything if it were to be stopped. For example, the initial login shell is typically in an orphaned process group. Orphaned process groups are immune to keyboard generated stop signals and job control signals resulting from reads or writes to the controlling terminal.

The Controlling Terminal

A terminal may belong to a process as its controlling terminal. Each process of a session that has a controlling terminal has the same controlling terminal. A terminal may be the controlling terminal for at most one session. The controlling terminal for a session is allocated by the session leader by issuing the `TIOCSCTTY` ioctl. A controlling terminal is never acquired by merely opening a terminal device file. When a controlling terminal becomes associated with a session, its foreground process group is set to the process group of the session leader.

The controlling terminal is inherited by a child process during a `fork`(2) function call. A process relinquishes its controlling terminal when it creates a new session with the `setsid`(2) function; other processes remaining in the old session that had this terminal as their controlling terminal continue to have it. A process does not relinquish its controlling terminal simply by closing all of its file descriptors associated with the controlling terminal if other processes continue to have it open.

When a controlling process terminates, the controlling terminal is disassociated from the current session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by other processes in the earlier session will be denied, with attempts to access the terminal treated as if modem disconnect had been sensed.

Terminal Access Control

If a process is in the foreground process group of its controlling terminal, read operations are allowed. Any attempts by a process in a background process group to read from its controlling terminal causes a `SIGTTIN` signal to be sent to the process's group unless one of the following special cases apply: If the reading process is ignoring or blocking the `SIGTTIN` signal, or if the process group of the reading process is orphaned, the `read`(2) returns -1 with *errno set to* `EIO` and no signal is sent. The default action of the `SIGTTIN` signal is to stop the process to which it is sent.

If a process is in the foreground process group of its controlling terminal, write operations are allowed. Attempts by a process in a background process group to write to its controlling terminal will cause the process group to be sent a `SIGTTOU` signal unless one of the following special cases apply: If `TOSTOP` is not set, or if `TOSTOP` is set and the process is ignoring or blocking the `SIGTTOU` signal, the process is allowed to write to the terminal and the `SIGTTOU` signal is not sent. If `TOSTOP` is set, and the process group of the writing process is orphaned, and the writing process is not ignoring or blocking `SIGTTOU`, the `write`(2) returns -1 with errno set to `EIO` and no signal is sent.

Certain calls that set terminal parameters are treated in the same fashion as write, except that `TOSTOP` is ignored; that is, the effect is identical to that of terminal writes when `TOSTOP` is set.

Input Processing and Reading Data

A terminal device associated with a terminal device file may operate in full-duplex mode, so that data may arrive even while output is occurring. Each terminal device file has associated with it an input queue, into which incoming data is stored by the system before being read by a process. The system imposes a limit, {`MAX_INPUT`}, on the number of bytes that may be stored in the input queue. The behavior of the system when this limit is exceeded depends on the setting of the `IMAXBEL` flag in the **`termios`** *`c_iflag`*. If this

flag is set, the terminal is sent an ASCII BEL character each time a character is received while the input queue is full. Otherwise, the input queue is flushed upon receiving the character.

Two general kinds of input processing are available, determined by whether the terminal device file is in canonical mode or noncanonical mode. Additionally, input characters are processed according to the *c_iflag* and *c_lflag* fields. Such processing can include echoing, which in general means transmitting input characters immediately back to the terminal when they are received from the terminal. This is useful for terminals that can operate in full-duplex mode.

The manner in which data is provided to a process reading from a terminal device file is dependent on whether the terminal device file is in canonical or noncanonical mode.

Another dependency is whether the O_NONBLOCK flag is set by open(2) or fcntl(2). If the O_NONBLOCK flag is clear, then the read request is blocked until data is available or a signal has been received. If the O_NONBLOCK flag is set, then the read request is completed, without blocking, in one of three ways:

1. If there is enough data available to satisfy the entire request, and the read completes successfully the number of bytes read is returned.
2. If there is not enough data available to satisfy the entire request, and the read completes successfully, having read as much data as possible, the number of bytes read is returned.
3. If there is no data available, the read returns -1, with errno set to EAGAIN.

When data is available depends on whether the input processing mode is canonical or noncanonical.

Canonical Mode Input Processing

In canonical mode input processing, terminal input is processed in units of lines. A line is delimited by a newline '\n' character, an end-of-file (EOF) character, or an end-of-line (EOL) character. See the **Special Characters** section for more information on EOF and EOL. This means that a read request will not return until an entire line has been typed, or a signal has been received. Also, no matter how many bytes are requested in the read call, at most one line is returned. It is not, however, necessary to read a whole line at once; any number of bytes, even one, may be requested in a read without losing information.

{MAX_CANON} is a limit on the number of bytes in a line. The behavior of the system when this limit is exceeded is the same as when the input queue limit {MAX_INPUT}, is exceeded.

Erase and kill processing occur when either of two special characters, the ERASE and KILL characters (see the **Special Characters section**), is received. This processing affects data in the input queue that has not yet been delimited by a newline NL, EOF, or EOL character. This un-delimited data makes up the current line. The ERASE character deletes the last character in the current line, if there is any. The KILL character deletes all data in the current line, if there is any. The ERASE and KILL characters have no effect if there is no data in the current line. The ERASE and KILL characters themselves are not placed in the input queue.

Noncanonical Mode Input Processing

In noncanonical mode input processing, input bytes are not assembled into lines, and erase and kill processing does not occur. The values of the VMIN and VTIME members of the *c_cc* array are used to determine how to process the bytes received.

VMIN represents the minimum number of bytes that should be received when the read(2) system call successfully returns. VTIME is a timer of 0.1 second granularity that is used to time out bursty and short term data transmissions. If VMIN is greater than { MAX_INPUT}, the response to the request is undefined. The four possible values for VMIN and VTIME and their interactions are described below.

Case A: VMIN > 0, VTIME > 0

In this case VTIME serves as an inter-byte timer and is activated after the first byte is received. Since it is an inter-byte timer, it is reset after a byte is received. The interaction between VMIN and VTIME is as follows: as soon as one byte is received, the inter-byte timer is started. If VMIN bytes are received before the inter-byte timer expires (remember that the timer is reset upon receipt of each byte), the read is satisfied. If the timer expires before VMIN bytes are received, the characters received to that point are returned to the user. Note that if VTIME expires at least one byte is returned because the timer would not have been enabled unless a byte was received. In this case (VMIN > 0, VTIME > 0) the read blocks until the VMIN and VTIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is in the buffer at the time of the read(2), the result is as if data had been received immediately after the read(2).

Case B: VMIN > 0, VTIME = 0

In this case, since the value of VTIME is zero, the timer plays no role and only VMIN is significant. A pending read is not satisfied until VMIN bytes are received (i.e., the pending read blocks until VMIN bytes are received), or a signal is received. A program that uses this case to read record-based terminal I/O may block indefinitely in the read operation.

Case C: VMIN = 0, VTIME > 0

In this case, since VMIN = 0, VTIME no longer represents an inter-byte timer. It now serves as a read timer that is activated as soon as the read function is processed. A read is satisfied as soon as a single byte is received or the read timer expires. Note that in this case if the timer expires, no bytes are returned. If the timer does not expire, the only way the read can be satisfied is if a byte is received. In this case the read will not block indefinitely waiting for a byte; if no byte is received within VTIME∗0.1 seconds after the read is initiated, the read returns a value of zero, having read no data. If data is in the buffer at the time of the read, the timer is started as if data had been received immediately after the read.

Case D: VMIN = 0, VTIME = 0

The minimum of either the number of bytes requested or the number of bytes currently available is returned without waiting for more bytes to be input. If no characters are available, read returns a value of zero, having read no data.

Writing Data and Output Processing

When a process writes one or more bytes to a terminal device file, they are processed according to the *c_oflag* field (see the **Output Modes** section). The implementation may provide a buffering mechanism; as such, when a call to write(2) completes, all of the bytes written have been scheduled for transmission to the device, but the transmission will not necessarily have been completed.

Special Characters

Certain characters have special functions on input or output or both. These functions are summarized as follows:

INTR Special character on input and is recognized if the ISIG flag (see the **Local Modes** section) is enabled. Generates a SIGINT signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the INTR character is discarded when processed.

QUIT Special character on input and is recognized if the ISIG flag is enabled. Generates a SIGQUIT signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the QUIT character is discarded when processed.

ERASE Special character on input and is recognized if the ICANON flag is set. Erases the last character in the current line; see **Canonical Mode Input Processing**. It does not erase beyond the start of a line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the ERASE character is dis-

carded when processed.

KILL Special character on input and is recognized if the ICANON flag is set. Deletes the entire line, as delimited by a NL, EOF, or EOL character. If ICANON is set, the KILL character is discarded when processed.

EOF Special character on input and is recognized if the ICANON flag is set. When received, all the bytes waiting to be read are immediately passed to the process, without waiting for a newline, and the EOF is discarded. Thus, if there are no bytes waiting (that is, the EOF occurred at the beginning of a line), a byte count of zero is returned from the read(2), representing an end-of-file indication. If ICANON is set, the EOF character is discarded when processed.

NL Special character on input and is recognized if the ICANON flag is set. It is the line delimiter '\n'.

EOL Special character on input and is recognized if the ICANON flag is set. Is an additional line delimiter, like NL.

SUSP If the ISIG flag is enabled, receipt of the SUSP character causes a SIGTSTP signal to be sent to all processes in the foreground process group for which the terminal is the controlling terminal, and the SUSP character is discarded when processed.

STOP Special character on both input and output and is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to temporarily suspend output. It is useful with fast terminals to prevent output from disappearing before it can be read. If IXON is set, the STOP character is discarded when processed.

START Special character on both input and output and is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to resume output that has been suspended by a STOP character. If IXON is set, the START character is discarded when processed.

CR Special character on input and is recognized if the ICANON flag is set; it is the '\r', as denoted in the C Standard {2}. When ICANON and ICRNL are set and IGNCR is not set, this character is translated into a NL, and has the same effect as a NL character.

The following special characters are extensions defined by this system and are not a part of IEEE Std 1003.1 ("POSIX.1") **termios**.

EOL2 Secondary EOL character. Same function as EOL.

WERASE Special character on input and is recognized if the ICANON flag is set. Erases the last word in the current line according to one of two algorithms. If the ALTWERASE flag is not set, first any preceding whitespace is erased, and then the maximal sequence of non-whitespace characters. If ALTWERASE is set, first any preceding whitespace is erased, and then the maximal sequence of alphabetic/underscores or non alphabetic/underscores. As a special case in this second algorithm, the first previous non-whitespace character is skipped in determining whether the preceding word is a sequence of alphabetic/underscores. This sounds confusing but turns out to be quite practical.

REPRINT
Special character on input and is recognized if the ICANON flag is set. Causes the current input edit line to be retyped.

DSUSP Has similar actions to the SUSP character, except that the SIGTSTP signal is delivered when one of the processes in the foreground process group issues a read(2) to the controlling terminal.

LNEXT Special character on input and is recognized if the IEXTEN flag is set. Receipt of this character causes the next character to be taken literally.

DISCARD
Special character on input and is recognized if the IEXTEN flag is set. Receipt of this character toggles the flushing of terminal output.

STATUS Special character on input and is recognized if the ICANON flag is set. Receipt of this character causes a SIGINFO signal to be sent to the foreground process group of the terminal. Also, if the NOKERNINFO flag is not set, it causes the kernel to write a status message to the terminal that displays the current load average, the name of the command in the foreground, its process ID, the symbolic wait channel, the number of user and system seconds used, the percentage of CPU the process is getting, and the resident set size of the process.

The NL and CR characters cannot be changed. The values for all the remaining characters can be set and are described later in the document under Special Control Characters.

Special character functions associated with changeable special control characters can be disabled individually by setting their value to {_POSIX_VDISABLE}; see **Special Control Characters**.

If two or more special characters have the same value, the function performed when that character is received is undefined.

Modem Disconnect

If a modem disconnect is detected by the terminal interface for a controlling terminal, and if CLOCAL is not set in the *c_cflag* field for the terminal, the SIGHUP signal is sent to the controlling process associated with the terminal. Unless other arrangements have been made, this causes the controlling process to terminate. Any subsequent call to the read(2) function returns the value zero, indicating end of file. Thus, processes that read a terminal file and test for end-of-file can terminate appropriately after a disconnect. Any subsequent write(2) to the terminal device returns -1, with *errno* set to EIO, until the device is closed.

GENERAL TERMINAL INTERFACE

Closing a Terminal Device File

The last process to close a terminal device file causes any output to be sent to the device and any input to be discarded. Then, if HUPCL is set in the control modes, and the communications port supports a disconnect function, the terminal device performs a disconnect.

Parameters That Can Be Set

Routines that need to control certain terminal I/O characteristics do so by using the **termios** structure as defined in the header ⟨termios.h⟩. This structure contains minimally four scalar elements of bit flags and one array of special characters. The scalar flag elements are named: *c_iflag*, *c_oflag*, *c_cflag*, and *c_lflag*. The character array is named *c_cc*, and its maximum index is NCCS.

Input Modes

Values of the *c_iflag* field describe the basic terminal input control, and are composed of following masks:

```
IGNBRK  /* ignore BREAK condition */
BRKINT  /* map BREAK to SIGINTR */
IGNPAR  /* ignore (discard) parity errors */
PARMRK  /* mark parity and framing errors */
INPCK   /* enable checking of parity errors */
ISTRIP  /* strip 8th bit off chars */
INLCR   /* map NL into CR */
IGNCR   /* ignore CR */
```

```
ICRNL    /* map CR to NL (ala CRMOD) */
IXON     /* enable output flow control */
IXOFF    /* enable input flow control */
IXANY    /* any char will restart after stop */
IMAXBEL  /* ring bell on input queue full */
```

In the context of asynchronous serial data transmission, a break condition is defined as a sequence of zero-valued bits that continues for more than the time to send one byte. The entire sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a time equivalent to more than one byte. In contexts other than asynchronous serial data transmission the definition of a break condition is implementation defined.

If IGNBRK is set, a break condition detected on input is ignored, that is, not put on the input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break condition flushes the input and output queues and if the terminal is the controlling terminal of a foreground process group, the break condition generates a single SIGINT signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break condition is read as a single '\0', or if PARMRK is set, as \377, '\0', '\0'.

If IGNPAR is set, a byte with a framing or parity error (other than break) is ignored.

If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than break) is given to the application as the three-character sequence \377, '\0', X, where \377, '\0' is a two-character flag preceding each sequence and X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of \377 is given to the application as \377, \377. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) is given to the application as a single character '\0'.

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled, allowing output parity generation without input parity errors. Note that whether input parity checking is enabled or disabled is independent of whether parity detection is enabled or disabled (see **Control Modes**). If parity detection is enabled but input parity checking is disabled, the hardware to which the terminal is connected recognizes the parity bit, but the terminal special file does not check whether this bit is set correctly or not.

If ISTRIP is set, valid input bytes are first stripped to seven bits, otherwise all eight bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). If IGNCR is not set and ICRNL is set, a received CR character is translated into a NL character.

If IXON is set, start/stop output control is enabled. A received STOP character suspends output and a received START character restarts output. If IXANY is also set, then any character may restart output. When IXON is set, START and STOP characters are not read, but merely perform flow control functions. When IXON is not set, the START and STOP characters are read.

If IXOFF is set, start/stop input control is enabled. The system shall transmit one or more STOP characters, which are intended to cause the terminal device to stop transmitting data, as needed to prevent the input queue from overflowing and causing the undefined behavior described in **Input Processing and Reading Data**, and shall transmit one or more START characters, which are intended to cause the terminal device to resume transmitting data, as soon as the device can continue transmitting data without risk of overflowing the input queue. The precise conditions under which STOP and START characters are transmitted are implementation defined.

If IMAXBEL is set and the input queue is full, subsequent input shall cause an ASCII BEL character to be transmitted to the output queue.

The initial input control value after open(2) is implementation defined.

Output Modes

Values of the *c_oflag* field describe the basic terminal output control, and are composed of the following masks:

```
OPOST   /* enable following output processing */
ONLCR   /* map NL to CR-NL (ala CRMOD) */
OCRNL   /* map CR to NL */
OXTABS  /* expand tabs to spaces */
ONOEOT  /* discard EOT's (^D) on output */
ONOCR   /* do not transmit CRs on column 0 */
ONLRET  /* on the terminal NL performs the CR function */
```

If OPOST is set, the remaining flag masks are interpreted as follows; otherwise characters are transmitted without change.

If ONLCR is set, newlines are translated to carriage return, linefeeds.

If OCRNL is set, carriage returns are translated to newlines.

If OXTABS is set, tabs are expanded to the appropriate number of spaces (assuming 8 column tab stops).

If ONOEOT is set, ASCII EOT's are discarded on output.

If ONOCR is set, no CR character is transmitted when at column 0 (first position).

If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0.

Control Modes

Values of the *c_cflag* field describe the basic terminal hardware control, and are composed of the following masks. Not all values specified are supported by all hardware.

```
CSIZE       /* character size mask */
CS5         /* 5 bits (pseudo) */
CS6         /* 6 bits */
CS7         /* 7 bits */
CS8         /* 8 bits */
CSTOPB      /* send 2 stop bits */
CREAD       /* enable receiver */
PARENB      /* parity enable */
PARODD      /* odd parity, else even */
HUPCL       /* hang up on last close */
CLOCAL      /* ignore modem status lines */
CCTS_OFLOW  /* CTS flow control of output */
CRTSCTS     /* logically the same as CCTS_OFLOW | CCTS_IFLOW */
CRTS_IFLOW  /* RTS flow control of input */
MDMBUF      /* flow control output via Carrier */
```

The CSIZE bits specify the byte size in bits for both transmission and reception. The *c_cflag* is masked with CSIZE and compared with the values CS5, CS6, CS7, or CS8. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stop bits are normally used.

If CREAD is set, the receiver is enabled. Otherwise, no character is received. Not all hardware supports this bit. In fact, this flag is pretty silly and if it were not part of the **termios** specification it would be omitted.

If PARENB is set, parity generation and detection are enabled and a parity bit is added to each character. If parity is enabled, PARODD specifies odd parity if set, otherwise even parity is used.

If HUPCL is set, the modem control lines for the port are lowered when the last process with the port open closes the port or the process terminates. The modem connection is broken.

If CLOCAL is set, a connection does not depend on the state of the modem status lines. If CLOCAL is clear, the modem status lines are monitored.

Under normal circumstances, a call to the open(2) function waits for the modem connection to complete. However, if the O_NONBLOCK flag is set or if CLOCAL has been set, the open(2) function returns immediately without waiting for the connection.

If the tty(4) TIOCFLAG_CLOCAL flag has been set on the port then the CLOCAL flag will automatically be set on every open.

The CCTS_OFLOW and CRTS_IFLOW flags are currently unused. Only CRTSCTS, which has the combined effect, is implemented. Note that CRTSCTS support is hardware and driver dependent. Check the specific port driver manual page to see if hardware flow control is supported on the port you are using.

If the tty(4) TIOCFLAG_CRTSCTS flag has been set on the port then the CRTSCTS flag will automatically be set on every open.

If MDMBUF is set then output flow control is controlled by the state of Carrier Detect.

If the tty(4) TIOCFLAG_MDMBUF flag has been set on the port then the MDMBUF flag will automatically be set on every open.

If the object for which the control modes are set is not an asynchronous serial connection, some of the modes may be ignored; for example, if an attempt is made to set the baud rate on a network connection to a terminal on another host, the baud rate may or may not be set on the connection between that terminal and the machine it is directly connected to.

Local Modes

Values of the *c_lflag* field describe the control of various functions, and are composed of the following masks.

```
ECHOKE      /* visual erase for line kill */
ECHOE       /* visually erase chars */
ECHO        /* enable echoing */
ECHONL      /* echo NL even if ECHO is off */
ECHOPRT     /* visual erase mode for hardcopy */
ECHOCTL     /* echo control chars as ^(Char) */
ISIG        /* enable signals INTR, QUIT, [D]SUSP */
ICANON      /* canonicalize input lines */
ALTWERASE   /* use alternative WERASE algorithm */
IEXTEN      /* enable DISCARD and LNEXT */
EXTPROC     /* external processing */
TOSTOP      /* stop background jobs from output */
FLUSHO      /* output being flushed (state) */
NOKERNINFO  /* no kernel output from VSTATUS */
PENDIN      /* re-echo input buffer at next read */
NOFLSH      /* don't flush output on signal */
```

If ECHO is set, input characters are echoed back to the terminal. If ECHO is not set, input characters are not echoed.

If ECHOE and ICANON are set, the ERASE character causes the terminal to erase the last character in the current line from the display, if possible. If there is no character to erase, an implementation may echo an indication that this was the case or do nothing.

If ECHOK and ICANON are set, the KILL character causes the current line to be discarded and the system echoes the '\n' character after the KILL character.

If ECHOKE and ICANON are set, the KILL character causes the current line to be discarded and the system causes the terminal to erase the line from the display.

If ECHOPRT and ICANON are set, the system assumes that the display is a printing device and prints a backslash and the erased characters when processing ERASE characters, followed by a forward slash.

If ECHOCTL is set, the system echoes control characters in a visible fashion using a caret followed by the control character.

If ALTWERASE is set, the system uses an alternative algorithm for determining what constitutes a word when processing WERASE characters (see WERASE).

If ECHONL and ICANON are set, the '\n' character echoes even if ECHO is not set.

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as described in **Canonical Mode Input Processing**.

If ICANON is not set, read requests are satisfied directly from the input queue. A read is not satisfied until at least VMIN bytes have been received or the timeout value VTIME expired between bytes. The time value represents tenths of seconds. See **Noncanonical Mode Input Processing** for more details.

If ISIG is set, each input character is checked against the special control characters INTR, QUIT, and SUSP (job control only). If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set.

If IEXTEN is set, implementation-defined functions are recognized from the input data. How IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF is implementation defined. If IEXTEN is not set, then implementation-defined functions are not recognized, and the corresponding input characters are not processed as described for ICANON, ISIG, IXON, and IXOFF.

If NOFLSH is set, the normal flush of the input and output queues associated with the INTR, QUIT, and SUSP characters are not be done.

If TOSTOP is set, the signal SIGTTOU is sent to the process group of a process that tries to write to its controlling terminal if it is not in the foreground process group for that terminal. This signal, by default, stops the members of the process group. Otherwise, the output generated by that process is output to the current output stream. Processes that are blocking or ignoring SIGTTOU signals are excepted and allowed to produce output and the SIGTTOU signal is not sent.

If NOKERNINFO is set, the kernel does not produce a status message when processing STATUS characters (see STATUS).

Special Control Characters

The special control characters values are defined by the array *c_cc*. This table lists the array index, the corresponding special character, and the system default value. For an accurate list of the system defaults, consult the header file ⟨ttydefaults.h⟩.

Index Name	*Special Character*	*Default Value*
VEOF	EOF	^D
VEOL	EOL	_POSIX_VDISABLE
VEOL2	EOL2	_POSIX_VDISABLE
VERASE	ERASE	^? \177
VWERASE	WERASE	^W
VKILL	KILL	^U
VREPRINT	REPRINT	^R
VINTR	INTR	^C
VQUIT	QUIT	^\ \34
VSUSP	SUSP	^Z
VDSUSP	DSUSP	^Y
VSTART	START	^Q
VSTOP	STOP	^S
VLNEXT	LNEXT	^V
VDISCARD	DISCARD	^O
VMIN	---	1
VTIME	---	0
VSTATUS	STATUS	^T

If the value of one of the changeable special control characters (see **Special Characters**) is {_POSIX_VDISABLE}, that function is disabled; that is, no input data is recognized as the disabled special character. If ICANON is not set, the value of {_POSIX_VDISABLE} has no special meaning for the VMIN and VTIME entries of the *c_cc* array.

The initial values of the flags and control characters after open(2) is set according to the values in the header ⟨sys/ttydefaults.h⟩.

SEE ALSO

tcsendbreak(3), tcsetattr(3)

NAME

tfb — PMAG-J and PMAGB-J TX colour unaccelerated 2-D framebuffer

SYNOPSIS

```
tfb* at tc? slot ? offset ?
wsdisplay* at tfb?
```

DESCRIPTION

The **tfb** driver provides support for the PMAG-J and PMAGB-J TX colour framebuffer for the TURBOchannel bus. The PMAG-J is an 8 bpp or 24 bpp colour framebuffer capable of running at a resolution of 1280-by-1024 at 66 Hz. The PMAGB-J is an 8 bpp or 24 bpp colour framebuffer capable of running at a resolution of 1280-by-1024 at 72 Hz.

SEE ALSO

cfb(4), mfb(4), px(4), pxg(4), sfb(4), tc(4), wscons(4)

BUGS

NetBSD/pmax does not currently support the machine-independent wscons(4) interface and uses a machine-dependent version.

NAME

ti — Alteon Networks Tigon I and Tigon II gigabit Ethernet driver

SYNOPSIS

```
ti* at pci? dev ? function ?
```

DESCRIPTION

The **ti** driver provides support for PCI gigabit Ethernet adapters based on the Alteon Networks Tigon gigabit Ethernet controller chip. The Tigon contains an embedded R4000 CPU, gigabit MAC, dual DMA channels and a PCI interface unit. The Tigon II contains two R4000 CPUs and other refinements. Either chip can be used in either a 32-bit or 64-bit PCI slot. Communication with the chip is achieved via PCI shared memory and bus master DMA. The Tigon I and II support hardware multicast address filtering, VLAN tag extraction and insertion, and jumbo Ethernet frames sizes up to 9000 bytes. Note that the Tigon I chipset is no longer in active production: all new adapters should come equipped with Tigon II chipsets.

There are several PCI boards available from both Alteon and other vendors that use the Tigon chipset under OEM contract. The **ti** driver has been tested with the following Tigon-based adapters:

- The Alteon AceNIC V gigabit (1000BASE-SX and 1000BASE-T variants) Ethernet adapter
- The 3Com 3c985-SX gigabit Ethernet adapter
- The Netgear GA620 gigabit (1000BASE-SX and 1000BASE-T variants) Ethernet adapter
- The Digital EtherWORKS 1000SX PCI Gigabit Adapter (DEGPA)

The following should also be supported but have not yet been tested:

- Silicon Graphics PCI gigabit Ethernet adapter

While the Tigon chipset supports 10, 100 and 1000Mbps speeds, support for 10 and 100Mbps speeds is only available on boards with the proper transceivers. Most adapters are only designed to work at 1000Mbps, however the driver should support those NICs that work at lower speeds as well.

Support for jumbo frames is provided via the interface MTU setting. Selecting an MTU larger than 1500 bytes with the `ifconfig`(8) utility configures the adapter to receive and transmit jumbo frames. Using jumbo frames can greatly improve performance for certain tasks, such as file transfers and data streaming.

The **ti** driver supports the following media types:

autoselect	Enable autoselection of the media type and options.
10baseT/UTP	Set 10Mbps operation. The *mediaopt* option can also be used to select either *full-duplex* or *half-duplex modes*.
100baseTX	Set 100Mbps (fast Ethernet) operation. The *mediaopt* option can also be used to select either *full-duplex* or *half-duplex* modes.
1000baseSX	Set 1000Mbps (gigabit Ethernet over multimode fiber) operation. Only full *full-duplex* mode is supported at this speed.
1000baseT	Set 1000Mbps (gigabit Ethernet over twisted pair) operation. Only full *full-duplex* mode is supported at this speed.

The **ti** driver supports the following media options:

full-duplex	Force full duplex operation.
half-duplex	Force half duplex operation.

The Alteon Tigon and Tigon II support IPv4/TCP/UDP checksumming in hardware. The **ti** supports this feature of the chip's firmware. See `ifconfig`(8) for information on how to enable this feature.

For more information on configuring this device, see `ifconfig`(8).

DIAGNOSTICS

ti%d: can't map memory space A fatal initialization error has occurred.

ti%d: couldn't map / establish interrupt A fatal initialization error has occurred.

ti%d: jumbo buffer allocation failed The driver failed to allocate memory for jumbo frames during initialization.

ti%d: bios thinks we're in a 64 bit slot, but we aren't The BIOS has programmed the NIC as though it had been installed in a 64-bit PCI slot, but in fact the NIC is in a 32-bit slot. This happens as a result of a bug in some BIOSes. This can be worked around on the Tigon II, but on the Tigon I initialization will fail.

ti%d: board self-diagnostics failed! The ROMFAIL bit in the CPU state register was set after system startup, indicating that the on-board NIC diagnostics failed.

ti%d: unknown hwrev The driver detected a board with an unsupported hardware revision. The **ti** driver supports revision 4 (Tigon 1) and revision 6 (Tigon 2) chips and has firmware only for those devices.

ti%d: watchdog timeout The device has stopped responding to the network, or there is a problem with the network connection (cable).

SEE ALSO

`netintro`(4), `pci`(4), `ifconfig`(8)

HISTORY

The **ti** device driver first appeared in NetBSD 1.4.2.

AUTHORS

The **ti** driver was written by Bill Paul ⟨wpaul@ctr.columbia.edu⟩.

BUGS

The driver currently tries to access some on-board memory transparently. This mapping (BUS_SPACE_MAP_LINEAR) fails on systems where the corresponding PCI memory range is located in "sparse" space only.

This driver currently does not work on big-endian systems.

NAME

timer — Timer driver for Sun SPARC computers

SYNOPSIS

```
timer0 at mainbus0                # sun4c
timer0 at obio0                   # sun4m
timer0 at obio0 addr 0xef000000   # sun4/300
```

DESCRIPTION

The **timer** device is used to generate clocks for the NetBSD kernel.

The hardclock is provided by the timer register and the statistics clock by CPU counter registers.

SEE ALSO

sparc/clock(4)

HISTORY

The **timer** appeared in NetBSD 1.0.

NAME

tl — Ethernet driver for Texas Instruments ThunderLAN based board

SYNOPSIS

```
tl* at pci? dev ? function ?
```

Configuration of PHYs are necessary. See mii(4).

DESCRIPTION

The **tl** device driver supports network adapters based on the Texas Instruments ThunderLAN chip.

HARDWARE

Supported cards include:

Compaq Netelligent
: in baseboard and PCI variants (10BASE-T-only variant untested).

Compaq NetFlex 3/P
: in baseboard variant only (the PCI variant doesn't use the same chip !).

It Baseboard Compaq Deskpro 4000 5233MMX Ethernet
: (This has been tested on the Deskpro 4000M only).

TI TravelMate 5000 series laptop docking station's Ethernet board.

MEDIA SELECTION

The different models of the supported boards come with some subset of RJ-45, BNC and AUI connectors. Media selection is done using ifconfig(8) using the standard ifmedia(4) mechanism. Refer to those manual pages for more information.

The **tl** driver don't have full automatic media selection. By default it will do an NWay on the UTP port for negotiation of the speed and duplex mode with the link partner. If the AUI or BNC port is used, an explicit medium must be specified to ifconfig(8) or in your /etc/ifconfig.tl? file.

SEE ALSO

ifmedia(4), mii(4), netintro(4), pci(4), ifconfig(8)

BUGS

The board marked as untested will always claim having an AUI connector, where it may be a BNC one.

NAME

tlp — DECchip 21x4x and clone Ethernet interfaces device driver

SYNOPSIS

```
tlp* at eisa? slot ?
tlp* at pci? dev ? function ?
tlp* at cardbus? function ?
```

Configuration of PHYs may also be necessary. See mii(4).

DESCRIPTION

The **tlp** device driver supports Ethernet interfaces based on the DECchip 21x4x "Tulip" (DEC fourth generation Ethernet controller) and a variety of clone chips. The Tulip has several features designed to make it flexible and reduce CPU usage:

- Flexible receive filter allowing for 16 perfect matches, 16 perfect inverse matches, 512-bit hash table plus 1 perfect match, or 512-bit hash table only.
- Uniform transmit descriptor architecture, configurable as a ring (allowing 2 buffers per descriptor) or a chain (allowing 1 buffer per descriptor).
- Uniform receive descriptor architecture, configurable as a ring (allowing 2 buffers per descriptor) or a chain (allowing 1 buffer per descriptor).
- Interrupt pacing; host may choose whether or not completion of processing of an individual descriptor causes an interrupt.
- Support for jumbo packets (by disabling transmit and receive watchdog timers).
- A patented transmit backoff algorithm which solves the Ethernet capture effect problem.
- Flexible bus modes to optimize DMA cycles for various cache sizes and bus implementations.
- Programmable transmit FIFO drain threshold to allow DMA overlap and reduce time to transmit.
- Flexible media attachment facilities.

The **tlp** driver supports the following chips:

- *DECchip 21040* -- This is the original Tulip Ethernet chip. It supports 10Mb/s speeds over a built-in serial interface. The serial interface has support for 10BASE-T and AUI media. The AUI port may be connected to 10BASE5 AUI or 10BASE2 BNC connectors, or both, selected by a gang jumper on the board. Some boards connect the BNC connector to an external serial interface. The driver has no way of knowing this, but the external serial interface may be selected with the "manual" media setting.

  Boards that include this chip include the DEC DE-435, on-board Ethernet on many DEC AlphaStation and AlphaServer systems, ZNYX ZX312, ZX312T, ZX314, ZX315, SMC 8432, SMC 8434, ACCTON EN1203, and some Cogent multi-port boards.

  This chip also appears on the DEC DE-425 EISA Ethernet board. This board is a DECchip 21040 and a PLX PCI glue chip, which provides the interface to the EISA bus, and special address decoding so that the PCI configuration space registers of the 21040 are accessible in normal EISA I/O space.

  The very first versions of this chip were labeled "DC1003" and "DC1003 Prototype".

- *DECchip 21041* -- This is the second chip in the Tulip family, dubbed "Tulip Plus". It supports 10Mb/s speeds over a built-in serial interface. The serial interface has support for 10BASE-T, 10BASE5 AUI, and 10BASE2 BNC media. The serial interface also includes support for IEEE

802.3u NWay over the 10BASE-T interface, for negotiation of duplex mode with the link partner.

Boards that include this chip include the DEC DE-450 and some SMC boards.

- *DECchip 21140 and 21140A* -- This is the third chip in the Tulip family, dubbed “FasterNet”. It supports 10Mb/s speeds with a built-in 10BASE-T encoder/decoder, and 100Mb/s speeds with a built-in 100BASE PCS function. Support for 100BASE-TX and 100BASE-T4 is provided by a built-in scrambler. Support for 100BASE-FX is possible with an appropriate PMD connected to the 100BASE PCS. The 21140 and 21140A also support 10Mb/s and 100Mb/s speeds over an MII interface connected to one or more PHYs.

  The 21140 and 21140A include a general purpose I/O facility, which may be used to toggle relays on the board. This facility is often used to reset individual board modules (e.g. the MII bus), select the output path of the chip (e.g. connect the UTP port on the board to the PHY, built-in 10BASE-T ENDEC, or built-in 100BASE-T PMD), or detect link status (by reading an output pin on the 100BASE-T magnetics).

  The 21140 and 21140A use a standardized data structure located in the SROM to describe how the chip should be programmed for various media settings, including the internal chip pathway, and GPIO settings. If the SROM data is not in the standardized format, the device driver must know specific programming information for that particular board.

  Boards that include the 21140 and 21140A include the DEC EB140, DE-500XA, DE-500AA, Asante EtherFast, DaynaPORT BlueStreak, Cogent EM100TX, EM110TX, EM440T4 multi-port, Kingston KNE100TX, older versions of the NetGear FA-310TX, SMC 9332, SMC 9334, ZNYX ZX34x multi-port, and Adaptec ANA-6944A/TX multi-port.

- *DECchip 21142 and 21143* -- These are the fourth and fifth chips in the Tulip family. While they have two different chip numbers, the 21142 and 21143 are essentially identical, with only minor differences related to available technology at time of manufacture. Both chips include support for 10Mb/s speeds over a built-in serial interface, and support for 10Mb/s and 100Mb/s speeds over an MII interface connected to one or more PHYs. The serial interface includes support for 10BASE-T, 10BASE5 AUI, and 10BASE2 BNC media, as well as support for IEEE 802.3u NWay over the 10BASE-T interface, for negotiation of duplex mode and link speed with the link partner.

  The 21143 adds support for 100Mb/s speeds with a built-in PCS function. Support for 100BASE-TX and 100BASE-T4 is provided by a built-in scrambler. Support for 100BASE-FX is possible with an appropriate PMD connected to the 100BASE PCS.

  The 21142 and 21143 include a general purpose I/O facility, which may be used to toggle relays on the board. This facility is often used to reset individual board modules (e.g. the MII bus), select the output path of the chip (e.g. connect the UTP port on the board to the PHY, built-in serial interface, or built-in 100BASE-T PMD), or detect link status (by reading an output pin on the 100BASE-T magnetics).

  The 21142 and 21143 use a standardized data structure located in the SROM to describe how the chip should be programmed for various media settings, including the internal chip pathway, and GPIO settings. If the SROM data is not in the standardized format, the device driver must know specific programming information for that particular board.

  Boards that include the 21142 include the DEC EB142, and on-board Ethernet on the Digital Personal Workstation (Alpha “Miata” and x86 models) and several Digital PCs.

  Boards that include the 21143 include the DEC EB143, DE-500BA, several commonly-available 100BASE-FX boards, the NetGear FA-510c CardBus card, and the Compu-Shack FASTline-II PCI boards.

- *Lite-On 82C168 and 82C169* -- These chips, dubbed “PNIC”, were some of the first commonly-available Tulip clones, appearing on low-cost boards when it became difficult for board vendors to obtain DECchip 21140A parts. They include support for 10Mb/s speeds over a built-in 10BASE-T encoder/decoder, and 100Mb/s speeds over a built-in PCS function. Support for 100BASE-TX and 100BASE-T4 is provided by a built-in scrambler and transceiver module. The transceiver module also includes support for NWay, for negotiating duplex mode and link speed with the link partner. These chips also include support for 10Mb/s and 100Mb/s speeds over and MII interface connected to one or more PHYs.

  These chips also include a GPIO facility, although it is programmed differently than the 21140’s.

  Unfortunately, these chips seem to be plagued by two unfortunate hardware bugs: in some situations, the receive logic incorrectly dumps the entire transmit FIFO into the receive chain, rather than a single Ethernet frame, and the DMA engines appear to be substandard; they must be run in store-and-forward mode, and occasionally fail to upload the filter setup frame.

  Boards that include the 82C168 and 82C169 include the newer NetGear FA-310TX, the Kingston KNE110TX, and some older LinkSys LNE100TX boards.

- *Macronix 98713, 98713A, 98715, 98715A, and 98725* -- Of all the clones, these chips, dubbed “PMAC”, are the best. They are very close clones of their respective originals, with the exception of some slight programming magic necessary to work around an apparent hardware bug.

  The 98713 is a DECchip 21140A clone. It includes all of the 21140A’s features, and uses the same SROM data format.

  The 98713A is a half-clone of the DECchip 21143. It has support for serial, PCS, and MII media. The serial interface has a built-in NWay function. However, the 98713A does not have a GPIO facility, and, as a result, usually does not use the same SROM format as the 21143 (no need for GPIO programming information).

  The 98715, 98715A, and 98725 are more 21143-like, but lack the GPIO facility and MII. These chips also support ACPI power management.

  Boards that include the Macronix chips include some SVEC boards, some SOHOWare boards, and the Compex RL100TX.

- *Lite-On/Macronix 82C115* -- This chip, dubbed the “PNIC-II”, was co-designed by Lite-On and Macronix. It is almost identical to the Macronix 98725, with a few exceptions: it has Wake-On-LAN support, uses a 128-bit receive filter hash table, and supports IEEE 802.3x flow control.

  Boards that include the 82C115 include the newer LinkSys (Version 2) LNE100TX boards.

- *Winbond 89C840F* -- This chip is a very low-end barely-a-clone of the 21140. It supports 10Mb/s and 100Mb/s speeds over an MII interface only, and has several programming differences from the 21140.

  The receive filter is completely different: it supports only a single perfect match, and has only a 64-bit multicast filter hash table. The receive filter is programmed using special registers rather than the standard Tulip setup frame.

  This chip is also plagued by a terrible DMA engine. The chip must be run in store-and-forward mode or it will often transmit garbage onto the wire.

  Interrupt pacing is also less flexible on the chip.

  Boards that include the 89C940F include the Complex RL100ATX, some Unicom 10/100 boards, and several no-name 10/100 boards.

- *ADMtek AL981* -- This chip is a low cost, single-chip (sans magnetics) 10/100 Ethernet implementation. It supports 10Mb/s and 100Mb/s speeds over an internal PHY. There is no generic MII bus; instead the IEEE 802.3u-compliant PHY is accessed via special registers on the chip. This chip also supports Wake-On-LAN and IEEE 802.3x flow control.

  The receive filter on the AL981 is completely different: it supports only a single perfect match, and has only a 64-bit multicast filter hash table. The receive filter is programmed using special registers rather than the standard Tulip setup frame.

  This chip also supports ACPI power management.

  A list of boards which include the AL981 is not yet available.

  Support for the AL981 has not yet been tested. If you have a board which uses this chip, please contact the author (listed below).

- *Xircom X3201-3* -- This chip is a CardBus 21143 clone with a loosely-coupled modem function (the modem is on a separate CardBus function, but the MAC portion includes a shadow of its interrupt status). Media is provided by an IEEE 802.3u-compliant PHY connected to an MII interface. These chips have no SROM; instead, the MAC address must be obtained from the card's CIS information. Unlike most other Tulip-like chips, the X3201-3 requires that transmit buffers be aligned to a 4-byte boundary. This virtually ensures that each outgoing packet must be copied into an aligned buffer, since the Ethernet header is 14 bytes long.

  This chip also supports ACPI power management.

  This chip is found in Xircom RealPort(tm) 10/100 CardBus Ethernet/Modem cards, as well as some Intel OEM'd RealPort(tm) and IBM Etherjet cards.

- *Davicom DM9102 and DM9102A* -- These chips are 21104A-like with a few minor exceptions. Media is provided by an internal IEEE 802.3u-compliant PHY accessed as if it were connected to a normal MII interface. The DM9102A also provides an external MII interface, to which a HomePNA 1 PHY is typically connected. The DM9102A also includes support for CardBus.

  This chip also supports ACPI power management and Wake-On-LAN.

  A complete list of boards with the DM9102 and DM9102A is not available. However, the DM9102 is often found on PC motherboards that include a built-in Ethernet interface.

- *ASIX AX88140A and AX88141* -- These chips are 21143-like with some exceptions. Media is proved by an internal IEEE 802.3u-compliant PHY connected to an MII interface. Unlike most other Tulip-like chips, AX88140A and AX88141 both require that the transmit buffers be aligned to a 4-byte boundary.

  It has a specific broadcast bit.

  This chip also supports ACPI power management.

  A list of boards which include the AX88140A or the AX88141 is not yet available.

- *Conexant RS7112 (LANfinity)* -- These chips are 21143 clones with coupled modem function. Media is provided by an IEEE 802.3u-compliant PHY connected to an MII interface.

  A list of boards which include the RS7112 is not yet available.

MEDIA SELECTION

Media selection done using `ifconfig`(8) using the standard `ifmedia`(4) mechanism. Refer to those manual pages for more information.

SEE ALSO

`arp`(4), `eisa`(4), `ifmedia`(4), `mii`(4), `netintro`(4), `pci`(4), `ifconfig`(8)

Digital Equipment Corporation, *DECchip 21040 Ethernet LAN Controller for PCI Hardware Reference Manual*, May 1994, Order Number EC-N0752-72.

Digital Equipment Corporation, *DECchip 21041 PCI Ethernet LAN Controller Hardware Reference Manual*, Preliminary, April 1995, Order Number EC-QAWXA-TE.

Digital Equipment Corporation, *DECchip 21041 DC1017-BA Errata*, Revision 1.0, April 27, 1995, Order Number EC-QD2MA-TE.

Digital Equipment Corporation, *DECchip 21140 PCI Fast Ethernet LAN Controller Hardware Reference Manual*, Supersedes EC-Q0CA-TE, May 1995, Order Number EC-Q0CB-TE.

Digital Equipment Corporation, *DECchip 21140A PCI Fast Ethernet LAN Controller Hardware Reference Manual*, Supersedes EC-QN7NA-TE, EC-QN7NB-TE, January 1996, Order Number EC-QN7NC-TE.

Intel Corporation, *21143 PCI/CardBus 10/100Mb/s Ethernet LAN Controller Hardware Reference Manual*, Revision 1.0, October 1998, Document Number 278074-001.

Digital Equipment Corporation, *Ethernet Address ROM Programming: An Application Note*, April 1994, Order Number EC-N3214-72.

Digital Equipment Corporation, *Using the DECchip 21041 with Boot ROM, Serial ROM, and External Register: An Application Note*, April 1995, Order Number EC-QJLGA-TE.

Digital Equipment Corporation, *Connecting the DECchip 21140 PCI Fast Ethernet LAN Controller to the Network: An Application Note*, Preliminary, December 1994, Order Number EC-QAR2A-TE.

Macronix International Co., Ltd., *MXIC MX98713 PMAC 100/10BASE PCI MAC Controller*, Revision 1.1, November 8, 1996, Part Number: PM0386.

Macronix International Co., Ltd., *MXIC MX98713A Fast Ethernet MAC Controller*, Revision 1.0, August 28, 1997, Part Number: PM0489.

Macronix International Co., Ltd., *MXIC MX98715A Single Chip Fast Ethernet NIC Controller*, Revision 1.2, February 24, 1999, Part Number: PM0537.

Macronix International Co., Ltd., *MXIC MX98725 Single Chip Fast Ethernet NIC Controller*, Revision 1.7, September 15, 1998, Part Number: PM0468.

Macronix International Co., Ltd., *MXIC MX98715 Application Note*, Revision 1.5, October 9, 1998, Part Number: PM0498.

Macronix International Co., Ltd., *MXIC MX98715A Application Note*, Revision 1.2, October 9, 1998, Part Number: PM0541.

Macronix International Co., Ltd., *MXIC MX98725 Application Note*, Revision 1.1, July 10, 1998, Part Number: PM0525.

Macronix International Co., Ltd., *MXIC LC82C115 Single Chip Fast Ethernet NIC Controller*, Revision 0.2, February 12, 1999, Part Number: PM0572.

LITE ON, Inc., *PNIC Hardware Specification*, Revision 1.0, December 1, 1994.

ADMtek Incorporated, *Comet: AL981 PCI 10/100 Fast Ethernet Controller with Integrated PHY*, Revision 0.93, January, 1999.

Winbond Electronics Corporation, *Winbond LAN W89C840F 100/10Mbps Ethernet Controller*, Revision A1, April 1997.

Xircom X3201-3 CardBus 10/100 Mbps Ethernet Controller Software Developer's Specification, Revision B, April 7, 1999, Reference number: 103-0548-001.

Davicom DM9102 10/100 Mbps Single Chip LAN Controller, Version DM9102-DS-F01, July 22, 1999.

Davicom DM9102A Single Chip Fast Ethernet NIC Controller, Version DM9102A-DS-F01, January 20, 2000.

ASIX Electronics Co., *ASIX AX88140A 100BaseTX/FX PCI Bus Fast Ethernet MAC Controller*, Preliminary, March 11, 1997, Document Number AX140D2.DOC.

Conexant Systems, Inc., *LANfinity - Home Networking Physical Layer Device with Integrated Analog Front End Circuitry*, Revision A, March 12, 1999.

HISTORY

The **tlp** driver first appeared in NetBSD 1.5.

AUTHORS

The **tlp** driver was written by Jason R. Thorpe while employed at the Numerical Aerospace Simulation Facility, NASA Ames Research Center. The author may be contacted at ⟨thorpej@NetBSD.org⟩.

ASIX AX88140A and AX881401 support was added by Rui Paulo ⟨rpaulo@NetBSD.org⟩.

Conexant RS7112 support was contributed by Frank Wille ⟨frank@phoenix.owl.de⟩.

BUGS

Media autosense is not yet supported for any serial or PCS function media. It is, however, supported for IEEE 802.3u-compliant PHY media.

NAME

tlphy — Driver for TI ThunderLAN internal Ethernet PHYs

SYNOPSIS

tlphy* **at mii? phy ?**

DESCRIPTION

The **tlphy** driver supports the internal PHY on TI ThunderLAN Ethernet interfaces. The ThunderLAN PHY supports 10BASE5 (AUI) and 10BASE2 (BNC) on some ThunderLAN implementations.

SEE ALSO

ifmedia(4), intro(4), mii(4), tl(4), ifconfig(8)

NAME

tlsb — AlphaServer 8x00 TurboLaser System bus

SYNOPSIS

```
tlsb* at mainbus0
tlsbmem* at tlsb? node ? offset ?
```

DESCRIPTION

The **tlsb** driver provides support for the TurboLaser System Bus found on AlphaServer 8200 and 8400 systems.

The following devices are supported by the **tlsb** driver:

gbus	internal bus on AlphaServer CPU modules
kft	KFTIA and KFTHA Bus Adapter Node for I/O hoses
tlsbmem	TurboLaser system memory

SEE ALSO

gbus(4), intro(4), kft(4), mainbus(4)

NAME

tm — TM-11/TE-10 mag tape device interface

SYNOPSIS

controller tm0 at uba? csr 0172520 vector tmintr tape te0 at tm0 drive 0

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The TM-11/TE-10 combination provides a standard tape drive interface as described in mtio(4). Hardware implementing this on the VAX is typified by the Emulex TC-11 controller operating with a Kennedy model 9300 tape transport, providing 800 and 1600 BPI operation at 125 IPS.

DIAGNOSTICS

te%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

te%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

te%d: can't change density in mid-tape. An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.

te%d: hard error bn%d er=%b. A tape error occurred at block *bn*; the tm error register is printed in octal with the bits symbolically decoded. Any error is fatal on non-raw tape; when possible the driver will have retried the operation which failed several times before reporting the error.

te%d: lost interrupt. A tape operation did not complete within a reasonable time, most likely because the tape was taken off-line during rewind or lost vacuum. The controller should, but does not, give an interrupt in these cases. The device will be made available again after this message, but any current open reference to the device will return an error as the operation in progress aborts.

SEE ALSO

mt(1), tar(1), ht(4), mt(4), mtio(4), ts(4), ut(4)

HISTORY

A **tm** driver appeared in Version 6 AT&T UNIX.

BUGS

May hang if a physical (non-data) error occurs.

NAME

tm121temp — Texas Instruments TMP121 temperature sensor

SYNOPSIS

```
tm121temp0 at spi? slave 0
```

DESCRIPTION

The **tm121temp** driver provides support for the Texas Instruments (formerly Burr-Brown) TMP121 temperature sensor with the envsys(4) API. These devices are connected to the host system with spi(4) or Microwire 4-wire serial busses. The **tm121temp** is capable of measuring temperatures with a 2 degree accuracy over the frame from -40 to 125 Celsius. Temperatures are reported with a granularity of 625 microKelvins.

tm121temp devices can transfer information at up to 10MHz, and support SPI mode 0.

The device reports itself with envsys(4) using the generic device name. This name can be overridden, by specifying an alternate name in the device property “envsys-description”.

SEE ALSO

envsys(4), spi(4), envstat(8)

HISTORY

The **tm121temp** driver was written by Garrett D’Amore and first appeared in NetBSD 4.0.

NAME

topcat — HP98544 98550 ‘‘Topcat’’ and ‘‘Catseye’’ device interface

SYNOPSIS

```
topcat* at intio?
topcat* at dio? scode ?
```

DESCRIPTION

This driver is for the HP98544, 98545 and 98547 ‘‘Topcat’’ and HP98548, 98549, and 98550 ‘‘Catseye’’ display cards. This driver merely checks for the existence of the device and does minimal set up, as it is expected the applications will initialize the device to their requirements. The Topcat and Catseye are nearly identical in common usage and only the Topcat will be referred to from now on.

The Topcat display cards are not user configurable. If one is present on a system, it will always have a frame buffer address of 0x200000 and a control register address of 0x560000. These are the HP series 300 ITE (Internal Terminal Emulator) defaults. The device can also be used as a graphics output device.

The `ioctl`(2) calls supported by the BSD system for the Topcat are:

`GRFIOCGINFO` Get Graphics Info

Get info about device, setting the entries in the *grfinfo* structure, as defined in ⟨`hpdev/grfioctl.h`⟩. For the 98544 or 98549, the number of planes should be 1, as they are monochrome devices. The number of planes for a 98545 is 4, translating to 15 colors, excluding black. The 98547 and 98548 cards have 6 planes, yielding 63 colors and black. The 98550 has 8 planes, yielding 255 colors and black. The displayed frame buffer size for the 98549 and 98550 is 2048 x 1024, for the others it is 1024 x 768.

`GRFIOCON` Graphics On

Turn graphics on by enabling CRT output. The screen will come on, displaying whatever is in the frame buffer, using whatever colormap is in place.

`GRFIOCOFF` Graphics Off

Turn graphics off by disabling output to the CRT. The frame buffer contents are not affected.

`GRFIOCMAP` Map Device to user space

Map in control registers and framebuffer space. Once the device file is mapped, the frame buffer structure is accessible. The frame buffer structure describing Topcat/Catseye devices is defined in ⟨`hpdev/grf_tcreg.h`⟩.

For further information about the use of `ioctl`(2) see the man page.

FILES

`/dev/grf?`	BSD special file
`/dev/crt9837`	
`/dev/crt98550`	HP-UX *starbase* special files
`/dev/MAKEDEV.hpux`	script for creating HP-UX special files

EXAMPLES

A small example of opening, mapping and using the device is given below. For more examples of the details on the behavior of the device, see the device dependent source files for the X Window System, in the `/usr/src/new/X/libhp` directory.

```
struct tcboxfb *tc;
u_char *Addr, frame_buffer;
struct grfinfo gi;
int disp_fd;

disp_fd = open("/dev/grf0",1);

if (ioctl (disp_fd, GRFIOCGINFO, &gi) < 0) return -1;

(void) ioctl (disp_fd, GRFIOCON, 0);

Addr = (u_char *) 0;
if (ioctl (disp_fd, GRFIOCMAP, &Addr) < 0) {
       (void) ioctl (disp_fd, GRFIOCOFF, 0);
       return -1;
}
tc = (tcboxfb *) Addr;                          /* Control Registers   */
frame_buffer = (u_char *) Addr + gi.gd_regsize; /* Frame buffer memory */
```

DIAGNOSTICS

None under BSD. HP-UX /usr/CE.utilities/Crtadjust programs must be used.

ERRORS

[ENODEV] no such device.

[EBUSY] Another process has the device open.

[EINVAL] Invalid `ioctl`(2) specification.

SEE ALSO

`ioctl`(2), `grf`(4), `ite`(4)

NAME

tp — ISO Transport Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netiso/iso_errno.h>
#include <netiso/tp_param.h>
#include <netiso/tp_user.h>

int
socket([AF_INET, AF_ISO], SOCK_SEQPACKET, 0);
```

DESCRIPTION

The TP protocol provides reliable, flow-controlled, two-way transmission of data and record boundaries. It is a byte-stream protocol and is accessed according to the SOCK_SEQPACKET abstraction. The TP protocol makes use of a standard ISO address format, including a Network Service Access Point, and a Transport Service Entity Selector. Subclass 4 may make use of the Internet address format.

Sockets using the TP protocol are either “active” or “passive”. Active sockets initiate connections to passive sockets. By default TCP sockets are created active; to create a passive socket the listen(2) system call must be used after binding the socket with the bind(2) system call. Only passive sockets may use the accept(2) call to accept incoming connections. Only active sockets may use the connect(2) call to initiate connections.

Passive sockets may “underspecify” their location to match incoming connection requests from multiple networks. This technique, termed “wildcard addressing”, allows a single server to provide service to clients on multiple networks. To create a socket which listens on all networks, the NSAP portion of the bound address must be void (of length zero). The Transport Selector may still be specified at this time; if the port is not specified the system will assign one. Once a connection has been established the socket’s address is fixed by the peer entity’s location. The address assigned the socket is the address associated with the network interface through which packets are being transmitted and received.

The ISO Transport Protocol implemented for AOS R2 at the University of Wisconsin - Madison, and modified for inclusion in the Berkeley Software Distribution, includes classes 0 and 4 of the ISO transport protocols as specified in the June 1986 version of IS 8073. Class 4 of the protocol provides reliable, sequenced, flow-controlled, two-way transmission of data packets with an alternative stop-and-wait data path called the "expedited data" service. Class 0 is essentially a null transport protocol, which is used when the underlying network service provides reliable, sequenced, flow-controlled, two-way data transmission. Class 0 does not provide the expedited data service. The protocols are implemented as a single transport layer entity that coexists with the Internet protocol suite. Class 0 may be used only in the ISO domain. Class 4 may be used in the Internet domain as well as in the ISO domain.

Two system calls were modified from the previous release of the Berkeley Software Distribution to permit the support of the end-of-transport-service-data-unit (EOTSDU) indication, and for the receipt and transmission of user connect, confirm, and disconnect data. See sendmsg(2) and recvmsg(2), and further discussion below for the formats of the data in the ancillary data buffer. If the EOTSDU is not needed, the normal read(2) and write(2) system calls may be used.

Through the getsockopt(2) and setsockopt(2) system calls, TP supports several options to control such things as negotiable options in the protocol and protocol strategies. The options are defined in ⟨netiso/tp_user.h⟩, and are described below.

In the tables below, the options marked with a pound sign ‘#’ may be used with setsockopt(2) after a connection is established. Others must be used before the connection is established, in other words, before calling connect(2) or accept(2). All options may be used with getsockopt(2) before or after a con-

nection is established.

`TPOPT_CONN_DATA` (char *) [none]
Data to send on `connect`(2). The passive user may issue a `getsockopt`(2) call to retrieve a connection request's user data, after having done the `accept`(2) system call without implying confirmation of the connection.

The data may also be retrieved by issuing a `recvmsg`(2) request for ancillary data only, without implying confirmation of the connection. The returned *cmsghdr* will contain `SOL_TRANSPORT` for the *csmg_level* and `TPOPT_CONN_DATA` for *cmsg_type*.

`TPOPT_DISC_DATA #` (char *) [none]
Data to send on `close`(2). Disconnect data may be sent by the side initiating the close but not by the passive side ("passive" with respect to the closing of the connection), so there is no need to read disconnect data after calling `close`(2). This may be sent by a `setsockopt`(2) system call, or by issuing a `sendmsg`(2) request specifying ancillary data only. The user-provided *cmsghdr* must contain `SOL_TRANSPORT` for *csmg_level* and `TPOPT_DISC_DATA` for *cmsg_type*. Sending of disconnect data will in of itself tear down (or reject) the connection.

`TPOPT_CFRM_DATA #` (char *) [none]
Data to send when confirming a connection. This may also be sent by a `setsockopt`(2) system call, or by issuing a `sendmsg`(2) request, as above. Sending of connect confirm data will cause the connection to be confirmed rather than rejected.

`TPOPT_PERF_MEAS #` Boolean.
When true, performance measurements will be kept for this connection. When set before a connection is established, the active side will use a locally defined parameter on the connect request packet; if the peer is another ARGO implementation, this will cause performance measurement to be turned on on the passive side as well.

`TPOPT_PSTATISTICS` No associated value on input. On output, *`struct tp_pmeas`*.

This command is used to read the performance statistics accumulated during a connection's lifetime. It can only be used with `getsockopt`(2). The structure it returns is described in ⟨`netiso/tp_stat.h`⟩.

`TPOPT_FLAGS` unsigned integer. [0x0]
This command can only be used with `getsockopt`(2). See the description of the flags below.

`TPOPT_PARAMS` *`struct tp_conn_param`*
Used to get or set a group parameters for a connection. The *`struct tp_conn_param`* is the argument used with the `getsockopt`(2) or `setsockopt`(2) system call. It is described in ⟨`netiso/tp_user.h`⟩.

The fields of the *`tp_conn_param`* structure are described below.

Values for TPOPT_PARAMS:

`p_Nretrans` nonzero short integer [1]
Number of times a TPDU will be retransmitted before the local TP entity closes a connection.

p_dr_ticks nonzero short integer [various]
Number of clock ticks between retransmissions of disconnect request TPDUs.

p_dt_ticks nonzero short integer [various]
Number of clock ticks between retransmissions of data TPDUs. This parameter applies only to class 4.

p_cr_ticks nonzero short integer [various]
Number of clock ticks between retransmissions of connection request TPDUs.

p_cc_ticks nonzero short integer [various]
Number of clock ticks between retransmissions of connection confirm TPDUs. This parameter applies only to class 4.

p_x_ticks nonzero short integer [various]
Number of clock ticks between retransmissions of expedited data TPDUs. This parameter applies only to class 4.

p_sendack_ticks nonzero short integer [various]
Number of clock ticks that the local TP entity will wait before sending an acknowledgment for normal data (not applicable if the acknowledgement strategy is TPACK_EACH). This parameter applies only to class 4.

p_ref_ticks nonzero short integer [various]
Number of clock ticks for which a reference will be considered frozen after the connection to which it applied is closed. This parameter applies to classes 4 and 0 in the ARGO implementation, despite the fact that the frozen reference function is required only for class 4.

p_inact_ticks nonzero short integer [various]
Number of clock ticks without an incoming packet from the peer after which TP close the connection. This parameter applies only to class 4.

p_keepalive_ticks
nonzero short integer [various]
Number of clock ticks between acknowledgments that are sent to keep an inactive connection open (to prevent the peer's inactivity control function from closing the connection). This parameter applies only to class 4.

p_winsize short integer between 128 and 16384. [4096 bytes]
The buffer space limits in bytes for incoming and outgoing data. There is no way to specify different limits for incoming and outgoing paths. The actual window size at any time during the lifetime of a connection is a function of the buffer size limit, the negotiated maximum TPDU size, and the rate at which the user program receives data. This parameter applies only to class 4.

p_tpdusize unsigned char between 0x7 and 0xd. [0xc for class 4] [0xb for class 0]
Log 2 of the maximum TPDU size to be negotiated. The TP standard (ISO 8473) gives an upper bound of 0xd for class 4 and 0xb for class 0. The ARGO implementation places upper bounds of 0xc on class 4 and 0xb on class 0.

p_ack_strat TPACK_EACH or TPACK_WINDOW. [TPACK_WINDOW]
This parameter applies only to class 4. Two acknowledgment strategies are supported:

TPACK_EACH means that each data TPDU is acknowledged with an AK TPDU.

TPACK_WINDOW means that upon receipt of the packet that represents the high edge of the last window advertised, an AK TPDU is generated.

p_rx_strat 4 bit mask [TPRX_USE_CW | TPRX_FASTSTART] over connectionless network protocols] [TPRX_USE_CW over connection-oriented network protocols]
This parameter applies only to class 4. The bit mask may include the following values:

TPRX_EACH: When a retransmission timer expires, retransmit each packet in the send window rather than just the first unacknowledged packet.

TPRX_USE_CW: Use a "congestion window" strategy borrowed from Van Jacobson's congestion window strategy for TCP. The congestion window size is set to one whenever a retransmission occurs.

TPRX_FASTSTART: Begin sending the maximum amount of data permitted by the peer (subject to availability). The alternative is to start sending slowly by pretending the peer's window is smaller than it is, and letting it slowly grow up to the peer window's real size. This is to smooth the effect of new connections on a congested network by preventing a transport connection from suddenly overloading the network with a burst of packets. This strategy is also due to Van Jacobson.

p_class 5 bit mask [TP_CLASS_4 | TP_CLASS_0]
Bit mask including one or both of the values TP_CLASS_4 and TP_CLASS_0. The higher class indicated is the preferred class. If only one class is indicated, negotiation will not occur during connection establishment.

p_xtd_format Boolean. [false]
Boolean indicating that extended format is negotiated. This parameter applies only to class 4.

p_xpd_service Boolean. [true]
Boolean indicating that the expedited data transport service will be negotiated. This parameter applies only to class 4.

p_use_checksum Boolean. [true]
Boolean indicating the use of checksums will be negotiated. This parameter applies only to class 4.

p_use_nxpd Reserved for future use.

p_use_rcc Reserved for future use.

p_use_efc Reserved for future use.

p_no_disc_indications
Boolean. [false]

Boolean indicating that the local TP entity will not issue indications (signals) when a TP connection is disconnected.

p_dont_change_params
Boolean. [false]
If *true* the TP entity will not override any of the other values given in this structure. If the values cannot be used, the TP entity will drop, disconnect, or refuse to establish the connection to which this structure pertains.

p_netservice One of { ISO_CLNS, ISO_CONS, ISO_COSNS, IN_CLNS }. [ISO_CLNS] Indicates which network service is to be used.

ISO_CLNS indicates the connectionless network service provided by CLNP (ISO 8473).

ISO_CONS indicates the connection-oriented network service provided by X.25 (ISO 8208) and ISO 8878.

ISO_COSNS indicates the connectionless network service running over a connection-oriented subnetwork service: CLNP (ISO 8473) over X.25 (ISO 8208).

IN_CLNS indicates the DARPA Internet connectionless network service provided by IP (RFC 791).

p_dummy Reserved for future use.

The TPOPT_FLAGS option is used for obtaining various boolean-valued options. Its meaning is as follows. The bit numbering used is that of the RT PC, which means that bit 0 is the most significant bit, while bit 8 is the least significant bit.

Values for TPOPT_FLAGS:

Bits	**Description [Default]**
0	TPFLAG_NLQOS_PDN: set when the quality of the network service is similar to that of a public data network.
1	TPFLAG_PEER_ON_SAMENET: set when the peer TP entity is considered to be on the same network as the local TP entity.
2	Not used.
3	TPFLAG_XPD_PRES: set when expedited data are present [0]
4..7	Reserved.

ERRORS

The TP entity returns *errno* error values as defined in ⟨sys/errno.h⟩ and ⟨netiso/iso_errno.h⟩.

If the TP entity encounters asynchronous events that will cause a transport connection to be closed, such as timing out while retransmitting a connect request TPDU, or receiving a DR TPDU, the TP entity issues a SIGURG signal, indicating that disconnection has occurred. If the signal is issued during a system call, the system call may be interrupted, in which case the *errno* value upon return from the system call is EINTR. If the signal SIGURG is being handled by reading from the socket, and it was an accept(2) that timed out, the read may result in ENOTSOCK, because the accept(2) call had not yet returned a legitimate socket descriptor when the signal was handled. ETIMEDOUT (or a some other errno value appropriate to the type of error) is returned if SIGURG is blocked for the duration of the system call. A user program should take one of the following approaches:

Block SIGURG
: If the program is servicing only one connection, it can block or ignore SIGURG during connection establishment. The advantage of this is that the *errno* value returned is somewhat meaningful. The disadvantage of this is that if ignored, disconnection and expedited data indications could be missed. For some programs this is not a problem.

Handle SIGURG
: If the program is servicing more than one connection at a time or expedited data may arrive or both, the program may elect to service SIGURG. It can use the **getsockopt**(*...TPOPT_FLAGS...*)

system call to see if the signal was due to the arrival of expedited data or due to a disconnection. In the latter case, `getsockopt`(2) will return `ENOTCONN`.

SEE ALSO

`netstat`(1), `clnp`(4), `cltp`(4), `iso`(4), `tcp`(4), `ifconfig`(8)

BUGS

The protocol definition of expedited data is slightly problematic, in a way that renders expedited data almost useless, if two or more packets of expedited data are sent within time epsilon, where epsilon depends on the application. The problem is not of major significance since most applications do not use transport expedited data. The problem is this: the expedited data acknowledgment TPDU has no field for conveying credit, thus it is not possible for a TP entity to inform its peer that "I received your expedited data but have no room to receive more." The TP entity has the choice of acknowledging receipt of the XPD TPDU:

when the user receives the XPD TSDU

which may be a fairly long time, which may cause the sending TP entity to retransmit the packet, and possibly to close the connection after retransmission, or

when the TP entity receives it

so the sending entity does not retransmit or close the connection. If the sending user then tries to send more expedited data “soon”, the expedited data will not be acknowledged (until the receiving user receives the first XPD TSDU).

The ARGO implementation acknowledges XPD TPDUs immediately, in the hope that most users will not use expedited data frequently enough for this to be a problem.

NAME

tqphy — Driver for TDK Semiconductor 78Q2120 10/100 Ethernet PHYs

SYNOPSIS

```
tqphy* at mii? phy ?
```

DESCRIPTION

The **tqphy** driver supports the TDK Semiconductor 78Q2120 PHY found in some CardBus and Mini-PCI Ethernet cards.

SEE ALSO

ifmedia(4), intro(4), mii(4), ifconfig(8)

BUGS

Due to hardware bugs, the tqphy driver refuses to match chips with revision 3 or lower. For these chips, use ukphy(4) instead.

NAME

tr — TROPIC based shared memory Token-Ring cards device driver

SYNOPSIS

```
tr0 at isa? port 0xa20 iomem 0xd800 irq ?
tr* at isa? port ? irq ?
tr* at isapnp?
tr* at mca? slot ?
tr* at pcmcia? function ?
```

DESCRIPTION

The **tr** device driver supports TROPIC I based shared-memory Token-Ring cards.

HARDWARE

Supported cards include the following IBM and 3Com models:

IBM Token-Ring Network PC Adapter

IBM Token-Ring Network PC Adapter II

IBM Token-Ring Network Adapter/A

IBM Token-Ring Network 16/4 Adapter

IBM Token-Ring Network 16/4 Adapter/A

IBM Token-Ring 16/4 Credit Card Adapter

IBM Token Ring Auto 16/4 Credit Card Adapter

IBM Turbo 16/4 Token Ring PC Card

IBM 16/4 ISA Adapter

IBM Auto 16/4 Token-Ring ISA Adapter

IBM Token Ring 16/4 Credit Card Adapter

IBM Token Ring Auto 16/4 Credit Card Adapter

IBM Turbo 16/4 Token Ring PC Card

3Com
3C619 TokenLink

3Com
3C319 TokenLink Velocity

3Com
3C389 TokenLink Velocity PC Card

SOURCE ROUTING

Setting IFF_LINK0 enables Token-Ring source routing. Setting IFF_LINK1 uses all-routes broadcasts otherwise single-route broadcasts are used.

NOTES

The MCA attachment has been only tested on IBM Token Ring 16/4 Adapter/A so far. It doesn't support `ifmedia`(4) yet, too.

SEE ALSO

`ifmedia`(4), `intro`(4), `isa`(4), `isapnp`(4), `mca`(4), `pcmcia`(4), `ifconfig`(8)

HISTORY

The **`tr`** driver appeared in NetBSD 1.4.

BUGS

The PCMCIA attachment does not work with the `cbb`(4) CardBus driver.

NAME

tra — Fujitsu MB86950 based Tiara Ethernet cards driver

SYNOPSIS

tra* **at mca? slot ?**

DESCRIPTION

The **tra** driver supports Tiara MCA bus Ethernet adapters and clones, based on the Fujitsu MB86950 Ethernet controller. Supported boards include:

Tiara LANCard/E Ethernet Adapter

Standard MicroSystems 3016/MC Ethernet

SEE ALSO

ifmedia(4), intro(4), mca(4), ifconfig(8)

HISTORY

The **tra** driver appeared in NetBSD 4.0.

AUTHORS

The **tra** driver has been written by Dave Barnes ⟨djb_pizza@ieee.org⟩.

BUGS

Multicast is not supported. There is no hardware support for multicast, and the driver does not implement a software-only solution.

NAME

trm — Tekram TRM-S1040 ASIC based PCI SCSI host adapter driver

SYNOPSIS

```
trm* at pci? dev ? function ?
scsibus* at trm?
```

DESCRIPTION

The **trm** driver supports PCI SCSI host adapters based on the Tekram TRM-S1040 SCSI ASIC.

HARDWARE

Supported SCSI controllers include:

Tekram DC-315 PCI Ultra SCSI adapter without flash BIOS and internal SCSI connector

Tekram DC-315U PCI Ultra SCSI adapter without flash BIOS

Tekram DC-395U PCI Ultra SCSI adapter with flash BIOS

Tekram DC-395UW PCI Ultra-Wide SCSI adapter with flash BIOS

Tekram DC-395F PCI Ultra-Wide SCSI adapter with flash BIOS and 68-pin external SCSI connector

For Tekram DC-390 PCI SCSI host adapter, use pcscp(4) driver.

For Tekram DC-310/U and DC-390U/UW/F PCI SCSI host adapters, use siop(4) driver.

SEE ALSO

cd(4), ch(4), intro(4), pci(4), scsi(4), sd(4), ss(4), st(4), uk(4), scsipi(9)

http://www.tekram.com/

AUTHORS

The **trm** driver was originally written for NetBSD 1.4/i386 by Erich Chen of Tekram Technology, and Rui-Xiang Guo rewrote the driver to use bus_space(9) and bus_dma(9) for NetBSD 1.6.

NAME

ts — TS-11/TSV-05 mag tape interface

SYNOPSIS

```
ts0 at uba? csr 0172520
```

DESCRIPTION

The TS-11/TSV-05 combination provides a standard tape drive interface as described in mtio(4). The TS-11 operates only at 1600 BPI, and only one transport is possible per controller.

The TS-11 is attached to an UNIBUS, and the TSV-05 is attached to a Q22 bus.

DIAGNOSTICS

ts%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

ts%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

ts%d: error at bn%d. An error occurred on the tape at block *bn*; status register and controller state is printed in hex and decimal.

SEE ALSO

mt(1), tar(1), ht(4), mt(4), mtio(4), tm(4), ut(4)

HISTORY

The **ts** driver appeared in 4.1BSD.
A new **ts** driver showed up in NetBSD 1.2.

BUGS

Lots of them.

NAME

tsc — DECchip 21272 Core Logic chipset

SYNOPSIS

```
tsc* at mainbus?
```

DESCRIPTION

The **tsc** driver provides support for the DECchip 21272 Core Logic chipset (Tsunami) found on the 264DP OEM Board.

SEE ALSO

intro(4), mainbus(4), tsp(4)

NAME

tslot — sbus pcmcia bridge

SYNOPSIS

```
tslot* at sbus? slot ? offset ?
pcmcia* at tslot? controller ? socket ?
```

DESCRIPTION

tslot is a driver for the PCMCIA part of the TS102 chip found in Tadpole SPARCbooks. The microcontroller interface is supported by tctrl(4).

KERNEL THREAD

Each **tslot** instance creates a kernel thread, named like the instance. This thread is used to watch for card insertions and removals, and handling the attachment and initialization of the card's driver.

SEE ALSO

pcmcia(4), sbus(4), tctrl(4)

NAME

tsp — DECchip 21272 Core Logic chipset PCI controller

SYNOPSIS

```
tsp* at tsc?
pci* at tsp?
```

DESCRIPTION

The **tsp** driver provides support for the PCI controller on the DECchip 21272 Core Logic chipset.

SEE ALSO

intro(4), pci(4), tsc(4)

NAME

ttwoga — DEC T2 Gate Array

SYNOPSIS

```
ttwoga* at mainbus?
```

DESCRIPTION

The **ttwoga** driver provides support for the Digital Equipment Corp. T2 Gate Array, the CBUS->PCI bridge found in the AlphaServer 2100/2100A.

SEE ALSO

intro(4), mainbus(4), ttwopci(4)

NAME

ttwopci — DEC T2 Gate Array PCI controller

SYNOPSIS

```
ttwopci* at ttwoga?
pci* at ttwopci?
```

DESCRIPTION

The **ttwopci** driver provides support for CBUS->PCI bridge on the DEC T2 Gate Array found in the AlphaServer 2100/2100A.

SEE ALSO

intro(4), pci(4), ttwoga(4)

NAME

tty — general terminal interface

SYNOPSIS

```
#include <sys/ioctl.h>
```

DESCRIPTION

This section describes the interface to the terminal drivers in the system.

Terminal Special Files

Each hardware terminal port on the system usually has two terminal special device files associated with it in the directory `/dev/` (for example, `/dev/tty03` and `/dev/dty03`).

The `/dev/ttyXX` special file is used for dial-in modems and terminals. When a user logs into the system on one of these hardware terminal ports, the system has already opened the associated device and prepared the line for normal interactive use (see `getty`(8)).

The `/dev/dtyXX` special file is a SunOS-compatible dial-out device. Unlike the dial-in device, opening the dial-out device never blocks. If the corresponding dial-in device is already opened (not blocked in the open waiting for carrier), then the dial-out open will fail immediately; otherwise it will succeed immediately. While the dial-out device is open, the dial-in device may not be opened. If the dial-in open is blocking, it will wait until the dial-out device is closed (and carrier is detected); otherwise it will fail immediately.

There is also a special case of a terminal file that connects not to a hardware terminal port, but to another program on the other side. These special terminal devices are called *ptys* (pseudo terminals) and provide the mechanism necessary to give users the same interface to the system when logging in over a network (using `rlogin`(1), or `telnet`(1) for example.) Even in these cases the details of how the terminal file was opened and set up is already handled by special software in the system. Thus, users do not normally need to worry about the details of how these lines are opened or used. Also, these lines are often used for dialing out of a system (through an out-calling modem), but again the system provides programs that hide the details of accessing these terminal special files (see `tip`(1)).

When an interactive user logs in, the system prepares the line to behave in a certain way (called a *line discipline*), the particular details of which is described in `stty`(1) at the command level, and in `termios`(4) at the programming level. A user may be concerned with changing settings associated with his particular login terminal and should refer to the preceding man pages for the common cases. The remainder of this man page is concerned with describing details of using and controlling terminal devices at a low level, such as that possibly required by a program wishing to provide features similar to those provided by the system.

Line disciplines

A terminal file is used like any other file in the system in that it can be opened, read, and written to using standard system calls. For each existing terminal file, there is a software processing module called a *line discipline* associated with it. The *line discipline* essentially glues the low level device driver code with the high level generic interface routines (such as `read`(2) and `write`(2)), and is responsible for implementing the semantics associated with the device. When a terminal file is first opened by a program, the default *line discipline* called the `termios` line discipline is associated with the file. This is the primary line discipline that is used in most cases and provides the semantics that users normally associate with a terminal. When the `termios` line discipline is in effect, the terminal file behaves and is operated according to the rules described in `termios`(4). Please refer to that man page for a full description of the terminal semantics. The operations described here generally represent features common across all *line disciplines*, however some of these calls may not make sense in conjunction with a line discipline other than `termios`, and some may not be supported by the underlying hardware (or lack thereof, as in the case of ptys).

Terminal File Operations

All of the following operations are invoked using the `ioctl`(2) system call. Refer to that man page for a description of the *request* and *argp* parameters. In addition to the ioctl *requests* defined here, the specific line discipline in effect will define other *requests* specific to it (actually `termios`(4) defines them as function calls, not ioctl *requests*.) The following section lists the available ioctl requests. The name of the request, a description of its purpose, and the typed *argp* parameter (if any) are listed. For example, the first entry says

TIOCSLINED char name[32]

and would be called on the terminal associated with file descriptor zero by the following code fragment:

```
ioctl(0, TIOCSLINED, "termios");
```

Terminal File Request Descriptions

`TIOCSLINED` *`char name[32]`*
: Change to the new line discipline called *`name`*.

`TIOCGLINED` *`char name[32]`*
: Return the current line discipline in the string pointed to by *`name`*.

`TIOCSBRK` *`void`*
: Set the terminal hardware into BREAK condition.

`TIOCCBRK` *`void`*
: Clear the terminal hardware BREAK condition.

`TIOCSDTR` *`void`*
: Assert data terminal ready (DTR).

`TIOCCDTR` *`void`*
: Clear data terminal ready (DTR).

`TIOCGPGRP` *`int *tpgrp`*
: Return the current process group the terminal is associated with in the integer pointed to by *`tpgrp`*. This is the underlying call that implements the `tcgetpgrp`(3) call.

`TIOCSPGRP` *`int *tpgrp`*
: Associate the terminal with the process group (as an integer) pointed to by *`tpgrp`*. This is the underlying call that implements the `tcsetpgrp`(3) call.

`TIOCGETA` *`struct termios *term`*
: Place the current value of the termios state associated with the device in the termios structure pointed to by *`term`*. This is the underlying call that implements the `tcgetattr`(3) call.

`TIOCSETA` *`struct termios *term`*
: Set the termios state associated with the device immediately. This is the underlying call that implements the `tcsetattr`(3) call with the `TCSANOW` option.

`TIOCSETAW` *`struct termios *term`*
: First wait for any output to complete, then set the termios state associated with the device. This is the underlying call that implements the `tcsetattr`(3) call with the `TCSADRAIN` option.

`TIOCSETAF` *`struct termios *term`*
: First wait for any output to complete, clear any pending input, then set the termios state associated with the device. This is the underlying call that implements the `tcsetattr`(3) call with the `TCSAFLUSH` option.

`TIOCOUTQ` *`int *num`*
Place the current number of characters in the output queue in the integer pointed to by *`num`*.

`TIOCSTI` *`char *cp`*
Simulate typed input. Pretend as if the terminal received the character pointed to by *`cp`*.

`TIOCNOTTY` *`void`*
This call is obsolete but left for compatibility. In the past, when a process that didn't have a controlling terminal (see *The Controlling Terminal* in `termios`(4)) first opened a terminal device, it acquired that terminal as its controlling terminal. For some programs this was a hazard as they didn't want a controlling terminal in the first place, and this provided a mechanism to disassociate the controlling terminal from the calling process. It *must* be called by opening the file `/dev/tty` and calling `TIOCNOTTY` on that file descriptor.

The current system does not allocate a controlling terminal to a process on an **`open`**() call: there is a specific ioctl called `TIOCSCTTY` to make a terminal the controlling terminal. In addition, a program can **`fork`**() and call the **`setsid`**() system call which will place the process into its own session - which has the effect of disassociating it from the controlling terminal. This is the new and preferred method for programs to lose their controlling terminal.

`TIOCSTOP` *`void`*
Stop output on the terminal (like typing ^S at the keyboard).

`TIOCSTART` *`void`*
Start output on the terminal (like typing ^Q at the keyboard).

`TIOCSCTTY` *`void`*
Make the terminal the controlling terminal for the process (the process must not currently have a controlling terminal).

`TIOCDRAIN` *`void`*
Wait until all output is drained.

`TIOCEXCL` *`void`*
Set exclusive use on the terminal. No further opens are permitted except by root. Of course, this means that programs that are run by root (or setuid) will not obey the exclusive setting - which limits the usefulness of this feature.

`TIOCNXCL` *`void`*
Clear exclusive use of the terminal. Further opens are permitted.

`TIOCFLUSH` *`int *what`*
If the value of the int pointed to by *`what`* contains the `FREAD` bit as defined in ⟨`sys/fcntl.h`⟩, then all characters in the input queue are cleared. If it contains the `FWRITE` bit, then all characters in the output queue are cleared. If the value of the integer is zero, then it behaves as if both the `FREAD` and `FWRITE` bits were set (i.e. clears both queues).

`TIOCGWINSZ` *`struct winsize *ws`*
Put the window size information associated with the terminal in the *winsize* structure pointed to by *`ws`*. The window size structure contains the number of rows and columns (and pixels if appropriate) of the devices attached to the terminal. It is set by user software and is the means by which most full-screen oriented programs determine the screen size. The *winsize* structure is defined in ⟨`sys/ioctl.h`⟩.

`TIOCSWINSZ` *`struct winsize *ws`*
Set the window size associated with the terminal to be the value in the *winsize* structure pointed to by *`ws`* (see above).

`TIOCCONS` *`int *on`*
If *`on`* points to a non-zero integer, redirect kernel console output (kernel printf's) to this terminal. If *`on`* points to a zero integer, redirect kernel console output back to the normal console. This is usually used on workstations to redirect kernel messages to a particular window.

`TIOCMSET` *`int *state`*
The integer pointed to by *`state`* contains bits that correspond to modem state. Following is a list of defined variables and the modem state they represent:

TIOCM_LE Line Enable.
TIOCM_DTR
Data Terminal Ready.
TIOCM_RTS
Request To Send.
TIOCM_ST Secondary Transmit.
TIOCM_SR Secondary Receive.
TIOCM_CTS
Clear To Send.
TIOCM_CAR
Carrier Detect.
TIOCM_CD Carrier Detect (synonym).
TIOCM_RNG
Ring Indication.
TIOCM_RI Ring Indication (synonym).
TIOCM_DSR
Data Set Ready.

This call sets the terminal modem state to that represented by *`state`*. Not all terminals may support this.

`TIOCMGET` *`int *state`*
Return the current state of the terminal modem lines as represented above in the integer pointed to by *`state`*.

`TIOCMBIS` *`int *state`*
The bits in the integer pointed to by *`state`* represent modem state as described above, however the state is OR-ed in with the current state.

`TIOCMBIC` *`int *state`*
The bits in the integer pointed to by *`state`* represent modem state as described above, however each bit which is on in *`state`* is cleared in the terminal.

`TIOCSFLAGS` *`int *state`*
The bits in the integer pointed to by *`state`* contain bits that correspond to serial port state. Following is a list of defined flag values and the serial port state they represent:

TIOCFLAG_SOFTCAR
Ignore hardware carrier.
TIOCFLAG_CLOCAL Set the `termios`(4) `CLOCAL` flag on open.

TIOCFLAG_CRTSCTS
Set the `termios`(4) `CRTSCTS` flag on open.
TIOCFLAG_MDMBUF
Set the `termios`(4) `MDMBUF` flag on open.

This call sets the serial port state to that represented by *`state`*. Not all serial ports may support this.

`TIOCGFLAGS` *`int *state`*
Return the current state of the serial port as represented above in the integer pointed to by *`state`*.

COMPATIBILITY

Two ioctls are maintained for backwards compatibility. They provide methods to get and set the current line discipline, but are not extensible.

`TIOCSETD` *`int *ldisc`*
Change to the new line discipline pointed to by *`ldisc`*. The old list of available line disciplines are listed in ⟨`sys/ttycom.h`⟩ and are:

TTYDISC	Termios interactive line discipline.
TABLDISC	Tablet line discipline.
SLIPDISC	Serial IP line discipline.
PPPDISC	Point to Point Protocol line discipline.
STRIPDISC	Starmode Radio IP line discipline.

`TIOCGETD` *`int *ldisc`*
Return the current line discipline in the integer pointed to by *`ldisc`*.

SEE ALSO

`stty`(1), `ioctl`(2), `tcgetattr`(3), `tcsetattr`(3), `ttyaction`(3), `pty`(4), `termios`(4), `ttys`(5), `getty`(8), `linedisc`(9)

HISTORY

Separate dial-out device files were implemented in SunOS 4. They were cloned by Charles M. Hannum for NetBSD 1.4.

NAME

tu — VAX-11/750 TU-58 console cassette interface

DESCRIPTION

The **tu** interface provides access to the VAX-11/750 TU-58 console cassette drive.

The interface supports only block I/O to the TU-58 cassettes. The devices are normally manipulated with the dd(1) program.

The device driver is automatically included when a system is configured to run on an 11/750.

The TU-58 on an 11/750 uses the Radial Serial Protocol (RSP) to communicate with the cpu over a serial line. This protocol is inherently unreliable as it has no flow control measures built in.

FILES

/dev/tu0

DIAGNOSTICS

tu%d: lost recv interrupt.
A timer watching the controller detected no interrupt for an extended period while an operation was outstanding. This usually indicates that one or more receiver interrupts were lost and the transfer is restarted.

HISTORY

The **tu** driver appeared in 4.1BSD.
A new driver showed up in NetBSD 1.2.

BUGS

The VAX-11/750 console interface is unusable while the system is multi-user, because interrupts occur at a very high priority level (spl7) and cannot be blocked with splbio. Use this driver only when the system is running single-user and, even then, with caution.

NAME

tun — tunnel software network interface

SYNOPSIS

```
pseudo-device tun
```

DESCRIPTION

The **tun** interface is a software loopback mechanism that can be loosely described as the network interface analog of the pty(4), that is, **tun** does for network interfaces what the **pty** driver does for terminals.

The **tun** driver, like the **pty** driver, provides two interfaces: an interface like the usual facility it is simulating (a network interface in the case of **tun**, or a terminal for **pty**), and a character-special device "control" interface.

To use a **tun** device, the administrator must first create the interface. This can be done by using the ifconfig(8) **create** command, or via the SIOCIFCREATE ioctl. An **open**() call on /dev/tun**N**, will also create a network interface with the same unit number of that device if it doesn't exists yet.

The network interfaces should be named **tun***0*, **tun***1*, etc. Each interface supports the usual network-interface ioctl(2)s, such as SIOCSIFADDR and SIOCSIFNETMASK, and thus can be used with ifconfig(8) like any other interface. At boot time, they are POINTOPOINT interfaces, but this can be changed; see the description of the control device, below. When the system chooses to transmit a packet on the network interface, the packet can be read from the control device (it appears there as "output"); writing a packet to the control device generates an input packet on the network interface, as if the (non-existent) hardware had just received it.

The tunnel device, normally /dev/tun**N**, is exclusive-open (it cannot be opened if it is already open) and is restricted to the super-user (regardless of file system permissions). A **read**() call will return an error (EHOSTDOWN) if the interface is not "ready" (which means that the interface address has not been set). Once the interface is ready, **read**() will return a packet if one is available; if not, it will either block until one is or return EAGAIN, depending on whether non-blocking I/O has been enabled. If the packet is longer than is allowed for in the buffer passed to **read**(), the extra data will be silently dropped.

Packets can be optionally prepended with the destination address as presented to the network interface output routine ('tunoutput'). The destination address is in 'struct sockaddr' format. The actual length of the prepended address is in the member 'sa_len'. The packet data follows immediately. A write(2) call passes a packet in to be "received" on the pseudo-interface. Each **write**() call supplies exactly one packet; the packet length is taken from the amount of data provided to **write**(). Writes will not block; if the packet cannot be accepted for a transient reason (e.g., no buffer space available), it is silently dropped; if the reason is not transient (e.g., packet too large), an error is returned. If "link-layer mode" is on (see TUNSLMODE below), the actual packet data must be preceded by a 'struct sockaddr'. The driver currently only inspects the 'sa_family' field. The following ioctl(2) calls are supported (defined in ⟨net/if_tun.h⟩):

TUNSDEBUG The argument should be a pointer to an *int*; this sets the internal debugging variable to that value. What, if anything, this variable controls is not documented here; see the source code.

TUNGDEBUG The argument should be a pointer to an *int*; this stores the internal debugging variable's value into it.

TUNSIFMODE The argument should be a pointer to an *int*; its value must be either IFF_POINTOPOINT or IFF_BROADCAST (optionally IFF_MULTICAST may be or'ed into the value). The type of the corresponding *tun***n** interface is set to the supplied type. If the value is anything else, an EINVAL error occurs. The interface must be down at the time; if it is up, an EBUSY error occurs.

`TUNSLMODE` The argument should be a pointer to an *int*; a non-zero value turns off "multi-af" mode and turns on "link-layer" mode, causing packets read from the tunnel device to be prepended with network destination address.

`TUNGIFHEAD` The argument should be a pointer to an *int*; the ioctl sets the value to one if the device is in "multi-af" mode, and zero otherwise.

`TUNSIFHEAD` The argument should be a pointer to an *int*; a non-zero value turns off "link-layer" mode, and enables "multi-af" mode, where every packet is preceded with a four byte address family.

`FIONBIO` Turn non-blocking I/O for reads off or on, according as the argument *int*'s value is or isn't zero (Writes are always nonblocking).

`FIOASYNC` Turn asynchronous I/O for reads (i.e., generation of `SIGIO` when data is available to be read) off or on, according as the argument *int*'s value is or isn't zero.

`FIONREAD` If any packets are queued to be read, store the size of the first one into the argument *int*; otherwise, store zero.

`TIOCSPGRP` Set the process group to receive `SIGIO` signals, when asynchronous I/O is enabled, to the argument *int* value.

`TIOCGPGRP` Retrieve the process group value for `SIGIO` signals into the argument *int* value.

The control device also supports `select`(2) for read; selecting for write is pointless, and always succeeds, since writes are always non-blocking.

On the last close of the data device, by default, the interface is brought down (as if with "ifconfig tun**n** down"). All queued packets are thrown away. If the interface is up when the data device is not open output packets are always thrown away rather than letting them pile up.

SEE ALSO

`inet`(4), `intro`(4)

HISTORY

IPv6 support comes mostly from FreeBSD and was added in NetBSD 4.0 by
Rui Paulo ⟨rpaulo@NetBSD.org⟩.

NAME

twa — 3ware Apache RAID controller driver

SYNOPSIS

twa* **at pci? dev ? function ?**

DESCRIPTION

The **twa** driver provides support for the 3ware Apache family of RAID controllers. Attached disk arrays are supported by the **ld** driver.

SEE ALSO

intro(4), ld(4)

HISTORY

The **twa** driver first appeared in NetBSD 3.1.

AUTHORS

The **twa** driver was written by Vinod Kashyap and was modified for inclusion in NetBSD by Jordan Rhody ⟨jordanr@spam.wasabisystems.com⟩.

NAME

twe — 3ware Escalade RAID controller driver

SYNOPSIS

```
twe* at pci? dev ? function ?
```

DESCRIPTION

The **twe** driver provides support for the 3ware Escalade family of RAID controllers. Supported models include series 5000, 6000, 7000, and 8000.

Attached disk arrays are supported by the **ld** driver.

SEE ALSO

intro(4), ld(4)

HISTORY

The **twe** driver first appeared in NetBSD 1.5.3, and was based on the FreeBSD driver of the same name.

NAME

txp — 3Com 3XP Typhoon/Sidewinder (3CR990) Ethernet interface

SYNOPSIS

txp* **at pci? dev ? function ?**

DESCRIPTION

The **txp** interface provides access to the 10Mb/s and 100Mb/s Ethernet networks via the 3Com Typhoon/Sidewinder chipset. This driver supports the following cards:

- 3Com 3CR990-TX-95
- 3Com 3CR990-TX-97
- 3Com 3CR990SVR95
- 3Com 3CR990SVR97

Basic Ethernet functions are provided as well as support for `vlan`(4) tag removal and insertion assistance, receive `ip`(4), `tcp`(4), and `udp`(4) checksum offloading, and transmit `ip`(4) checksum offloading. There is currently no support for transmit `tcp`(4) or `udp`(4) checksum offloading, `tcp`(4) segmentation, nor `ipsec`(4) acceleration. Note that hardware checksumming is only used when the interface is not in `bridge`(4) mode.

Each of the host's network addresses is specified at boot time with an `SIOCSIFADDR` `ioctl`(2). The **txp** interface employs the address resolution protocol described in `arp`(4) to dynamically map between Internet and Ethernet addresses on the local network.

When a **txp** interface is brought up, by default, it will attempt to auto-negotiate the link speed and duplex mode. The speeds, in order of attempt, are: 100Mb/s Full Duplex, 100Mb/s Half Duplex, 10 Mb/s Full Duplex, and 10 Mb/s Half Duplex.

The **txp** supports several media types, which are selected via the `ifconfig`(8) command. The supported media types are:

media autoselect — Attempt to autoselect the media type (default)

media 100baseTX mediaopt full-duplex
: Use 100baseTX, full duplex

media 100baseTX [mediaopt half-duplex]
: Use 100baseTX, half duplex

media 10baseT mediaopt full-duplex
: Use 10baseT, full duplex

media 10baseT [mediaopt half-duplex]
: Use 10baseT, half duplex

SEE ALSO

`arp`(4), `ifmedia`(4), `inet`(4), `intro`(4), `ip`(4), `netintro`(4), `pci`(4), `tcp`(4), `udp`(4), `vlan`(4), `ifconfig`(8)

HISTORY

The **txp** driver first appeared in NetBSD 2.0.

NAME

uaudio — USB audio device driver

SYNOPSIS

```
uaudio* at uhub?
audio*  at audiobus?
```

DESCRIPTION

The **uaudio** driver provides support for USB audio class devices.

A USB audio device consists of a number of components: input terminals (e.g. USB digital input), output terminals (e.g. speakers), and a number of units in between (e.g. volume control). The following types of units are handled by the **uaudio** driver and are accessible via the mixer (see audio(4)) interface:

- mixer — A mixer has a number of inputs and one output. Each input has a control that determines its volume in the output. The name of the control is *mixN-S*, where *N* is a number that identifies which mixer it is and *S* which input.
- selector — A selector unit selects one of multiple audio sources such as mic-in and line-in. The name of the control is *selN-S1S2S3...*, where *N* is a number that identifies which selector unit it is and the sequence of *Sn* indicates candidate units for the audio source.
- feature — A feature unit changes the sound in some way, like bass, treble, mute, or volume. The name of the control is determined in a heuristic way. If the unit changes the sound to a speaker output terminal, the names of the controls may be outputs.speaker.bass, outputs.speaker.treble, outputs.speaker.mute, outputs.speaker, or likewise.
- processing — A processing unit does one of a number of audio processing functions (e.g., channel up-down mixing, Dolby ProLogic, or chorus effects). The name of the on-off control is *proN.M-enable*, where *N* is a number that identifies which processing unit it is and *M* which kind. Depending on the type of processing unit there may be other controls as well.
- extension — An extension unit performs some unspecified audio processing The name of the on-off control is *extN-enable*, where *N* is a number that identifies which processing unit it is.

For more information the USB Audio class specification is indispensable reading.

SEE ALSO

The USB specifications can be found at: http://www.usb.org/developers/docs.html

audio(4), usb(4)

HISTORY

The **uaudio** driver appeared in NetBSD 1.5.

BUGS

There is no support for multiple-endpoints audio stream, adaptive recording, async playback, and TYPE-II/III formats.

There is the possibility that a device has multiple mixer items which have the same name.

NAME

uberry — USB driver for the RIM BlackBerry phones

SYNOPSIS

uberry* **at uhub? port ?**

DESCRIPTION

The **uberry** driver provides support for the original BlackBerry, the BlackBerry Pearl, the BlackBerry Pearl Duo, and other models.

This is a stub driver that only allows the device to be charged from the USB port. If the **uberry** driver attaches to a device that has support for memory cards (such as the BlackBerry Pearl series), the **uberry** driver will detach and the device will be re-attached using the umass(4) driver.

SEE ALSO

umass(4), usb(4)

HISTORY

The **uberry** driver appeared in NetBSD 5.0.

NAME

ubsa — USB support for Belkin serial adapters

SYNOPSIS

```
ubsa*  at uhub?
ucom*  at ubsa?
```

HARDWARE

The **ubsa** driver supports the following adapters:

Belkin F5U103
Belkin F5U120
e-Tek Labs Kwik232
GoHubs GoCOM232
Option N.V. Mobile Connect 3G datacard
Option N.V. GlobeTrotter Fusion Quad Lite UMTS/GPRS
Option N.V. GlobeTrotter Fusion Quad Lite 3D
Peracom single port serial adapter

DESCRIPTION

The **ubsa** driver provides support for the USB-to-RS-232 Bridge chip used by a variety of serial adapters from Belkin and other vendors. The bridge is also embedded into several UMTS/GPRS cards.

The device is accessed through the ucom(4) driver which makes it behave like a tty(4).

SEE ALSO

tty(4), ucom(4), usb(4)

HISTORY

The **ubsa** driver first appeared in FreeBSD 5.0 and was ported to NetBSD 2.0.

NAME

ubsec — Broadcom and BlueSteel uBsec 5x0x crypto accelerator

SYNOPSIS

ubsec* **at pci? dev ? function ?**

DESCRIPTION

The **ubsec** driver supports cards containing any of the following chips:

Bluesteel 5501	The original chipset, no longer made. This extremely rare unit was not very fast, lacked an RNG, and had a number of other bugs.
Bluesteel 5601	A faster and fixed version of the original, with a random number unit and large number engine added.
Broadcom BCM5801	A BCM5805 without public key engine or random number generator.
Broadcom BCM5802	A slower version of the BCM5805.
Broadcom BCM5805	Faster version of Bluesteel 5601.
Broadcom BCM5820	64 bit version of the chip, and significantly more advanced.
Broadcom BCM5821	Faster version of the BCM5820. (This is the chip found on the Sun Crypto Accelerator 1000.)
Broadcom BCM5822	Faster version of the BCM5820.
Broadcom BCM5823	Faster version of the BCM5822.
Broadcom BCM5823	Faster version of the BCM5821, with AES hardware.

The **ubsec** driver registers itself to accelerate DES, Triple-DES, MD5, SHA1, MD5-HMAC, and SHA1-HMAC operations for `opencrypto`(9), and thus for `fast_ipsec`(4) and `crypto`(4).

On those models which contain a public key engine (almost all of the more recent ones), this feature is registered with the `crypto`(4) subsystem.

On all models except the Bluesteel 5501 and Broadcom 5801, the driver registers itself to provide random data to the `rnd`(4) subsystem.

SEE ALSO

`crypto`(4), `fast_ipsec`(4), `intro`(4), `rnd`(4), `opencrypto`(9)

HISTORY

The **ubsec** device driver appeared in OpenBSD 2.8. The **ubsec** device driver was imported to FreeBSD 5.0, back-ported to FreeBSD 4.8, and subsequently imported to NetBSD 2.0.

BUGS

The BCM5801 and BCM5802 have not actually been tested.

Whilst some of the newer chips support AES, AES is not supported by the driver.

NAME

ubt — USB Bluetooth driver

SYNOPSIS

ubt* **at uhub? port ? configuration ? interface ?**

DESCRIPTION

The **ubt** driver provides support for USB Bluetooth dongles to the Bluetooth protocol stack.

USB Bluetooth dongles provide two interfaces, both of which the **ubt** driver claims. The second interface is used for Isochronous data and will have several alternate configurations regarding bandwidth consumption, which can be set using the hw.ubtN.config `sysctl`(8) variable. The number of alternate configurations is indicated by the value in the hw.ubtN.alt_config variable, and the isoc frame size for the current configuration is shown in the hw.ubtN.sco_rxsize and hw.ubtN.sco_txsize variables.

By default, configuration 0 is selected, which means that no bandwidth is used on the Isochronous interface and no SCO data can be sent. Consult the Bluetooth USB specification at https://www.bluetooth.org/ for complete instructions on setting bandwidth consumption. The following extract may be useful as a general guidance though details may differ between manufacturers.

0 No active voice channels
1 One voice channel with 8-bit encoding
2 Two voice channels with 8-bit encoding, or one voice channel with 16-bit encoding.
3 Three voice channels with 8-bit encoding
4 Two voice channels with 16-bit encoding
5 Three voice channels with 16-bit encoding

SEE ALSO

`bluetooth`(4), `uhub`(4), `sysctl`(8)

HISTORY

This **ubt** device driver was originally a character device written by David Sainty and Lennart Augustsson. It was rewritten to support socket based Bluetooth access for NetBSD 4.0 by Iain Hibbert.

CAVEATS

Isochronous data is seemingly not well supported over USB in the current system and to get SCO working, you may have to calculate the SCO packet size that the stack will use. This is the sco_mtu value reported by the `btconfig`(8) command, and when combined with the SCO header (3 bytes) should fit exactly into an integer number of Isochronous data frames where the frame size is indicated by the 'hw.ubtN.sco_txsize' sysctl variable.

For example: I want one voice channel (which is all that is supported, for now) so am using configuration #2, with a frame length of 17 bytes. This gives possible values of:

```
(17 * 1) - 3 = 14
(17 * 2) - 3 = 31
(17 * 3) - 3 = 48
(17 * 4) - 3 = 65
(17 * 5) - 3 = 82
etc.
```

`btconfig`(8) shows the maximum SCO payload as 64 bytes, so I am using the next smaller size of 48, to minimize the overhead of the 3 header bytes.

The SCO packet size can be changed using the ‘scomtu’ option to `btconfig`(8).

The failure mode is that the USB Bluetooth dongle locks up though generally removal/reinsertion will clear the problem.

BUGS

The Isochronous configuration can only be changed when the device is not marked up.

NAME

uchcom — USB support for WinChipHead CH341/CH340 serial adapters driver

SYNOPSIS

```
uchcom* at uhub?
ucom*   at uchcom?
```

HARDWARE

The **uchcom** driver supports the following adapters:

HL USB-RS232

DESCRIPTION

The **uchcom** driver provides support for the WinChipHead CH341/CH340 USB-to-RS-232 Bridge chip.

The device is accessed through the ucom(4) driver which makes it behave like a tty(4).

SEE ALSO

tty(4), ucom(4), usb(4)

HISTORY

The **uchcom** driver appeared in NetBSD 5.0.

BUGS

Actually, this chip seems unable to drive other than 8 data bits and 1 stop bit line.

NAME

ucom — USB tty support

SYNOPSIS

```
ucom* at ubsa?
ucom* at uchcom?
ucom* at uftdi?
ucom* at ugensa?
ucom* at uhmodem?
ucom* at uipaq?
ucom* at ukyopon?
ucom* at umct?
ucom* at umodem?
ucom* at uplcom?
ucom* at uslsa?
ucom* at uvisor? portno ?
ucom* at uvscom?
```

DESCRIPTION

The **ucom** driver attaches to USB modems, serial ports, and other devices that need to look like a tty. The **ucom** driver shows a behaviour like a tty(4). This means that normal programs such as tip(1) or pppd(8) can be used to access the device.

The *portno* locator can be used to decide which port to use for device that have multiple external ports.

FILES

/dev/ttyU?

SEE ALSO

tty(4), ubsa(4), uchcom(4), uftdi(4), ugensa(4), uhmodem(4), uipaq(4), ukyopon(4), umct(4), umodem(4), uplcom(4), usb(4), uslsa(4), uvisor(4), uvscom(4)

HISTORY

The **ucom** driver appeared in NetBSD 1.5.

NAME

ucycom — USB support for Cypress microcontroller based serial adapters

SYNOPSIS

```
ucycom   at uhidev? reportid ?
```

HARDWARE

The **ucycom** driver supports the following adapters:

Various low-cost USB-RS232 adapters
Delmore Earthmate GPS receiver

DESCRIPTION

The **ucycom** driver provides support for Cypress microcontroller based USB-to-RS-232 adapter. The **ucycom** driver shows a behaviour like a tty(4). This means that normal programs such as tip(1) or pppd(8) can be used to access the device.

FILES

/dev/ttyY?

SEE ALSO

uhidev(4), usb(4)

HISTORY

The **ucycom** driver appeared in NetBSD 4.0.

NAME

uda — UNIBUS MSCP disk controller driver

SYNOPSIS

```
uda0 at uba? csr 0172150
uda1 at uba? csr 0160334
mscpbus* at uda?
```

DESCRIPTION

The **uda** driver is for UNIBUS disk controllers that use MSCP. Among these controllers are:

DEC UDA50 UNIBUS ctlr
DEC KDA50 Q22 bus ctlr
DEC RQDX3 Q22 bus ctlr

The **uda** communicates with the host through a packet protocol known as the Mass Storage Control Protocol (MSCP). Consult the file ⟨mscp/mscp.h⟩ for a detailed description of this protocol.

SEE ALSO

intro(4)

HISTORY

The **uda** driver appeared in 4.2BSD.
A new **uda** driver approach showed up in NetBSD 1.2.

NAME

udav — Davicom DM9601 USB Ethernet driver

SYNOPSIS

```
udav* at uhub?
ukphy* at mii?
```

HARDWARE

The **udav** driver supports the following adapters:

Corega FEther USB-TXC
ShanTou ST268 USB NIC

DESCRIPTION

The **udav** driver provides support for USB Ethernet adapters based on the Davicom DM9601 chipset.

For more information on configuring this device, see `ifconfig`(8).

SEE ALSO

`arp`(4), `netintro`(4), `usb`(4), `ifconfig`(8)

Davicom DM9601 data sheet, http://www.davicom.com.tw/big5/download/Data%20Sheet/DM9601-DS-F01-062202s.pdf.

HISTORY

The **udav** device driver first appeared in NetBSD 2.0.

AUTHORS

The **udav** driver was written by Shingo WATANABE ⟨nabe@nabechan.org⟩.

NAME

udp — Internet User Datagram Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

int
socket(AF_INET, SOCK_DGRAM, 0);

int
socket(AF_INET6, SOCK_DGRAM, 0);
```

DESCRIPTION

UDP is a simple, unreliable datagram protocol which is used to support the SOCK_DGRAM abstraction for the Internet protocol family. UDP sockets are connectionless, and are normally used with the sendto(2) and recvfrom(2) calls, though the connect(2) call may also be used to fix the destination for future packets (in which case the recv(2) or read(2) and send(2) or write(2) system calls may be used).

UDP address formats are identical to those used by TCP. In particular UDP provides a port identifier in addition to the normal Internet address format. Note that the UDP port space is separate from the TCP port space (i.e. a UDP port may not be "connected" to a TCP port). In addition broadcast packets may be sent (assuming the underlying network supports this) by using a reserved "broadcast address"; this address is network interface dependent.

Options at the IP transport level may be used with UDP; see ip(4) or ip6(4).

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
[ENOTCONN]	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
[ENOBUFS]	when the system runs out of memory for an internal data structure;
[EADDRINUSE]	when an attempt is made to create a socket with a port which has already been allocated;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

getsockopt(2), recv(2), send(2), socket(2), inet(4), inet6(4), intro(4), ip(4), ip6(4)

User Datagram Protocol, RFC, 768, August 28, 1980.

Requirements for Internet Hosts -- Communication Layers, RFC, 1122, October 1989.

HISTORY

The **udp** protocol appeared in 4.2BSD.

NAME

udsbr — support for D-Link DSB-R100 USB radio

SYNOPSIS

```
udsbr* at uhub?
radio* at udsbr?
```

DESCRIPTION

The **udsbr** driver provides support for the D-Link DSB-R100 USB radio.

The device is accessed through the radio(4) driver.

The DSB-R100 is a rather poor radio and the only feedback that you can get from the radio is if it has a stereo signal or not.

SEE ALSO

radio(4), usb(4)

HISTORY

The **udsbr** driver appeared in NetBSD 1.6.

NAME

uep — USB eGalax Touchpanel

SYNOPSIS

```
uep*     at uhub? port ?
wsmouse* at uep?
```

DESCRIPTION

The **uep** driver provides support for USB touchpanel controllers by eGalax. Access to panel events is through the wsmouse(4) driver.

SEE ALSO

usb(4), wsmouse(4)

HISTORY

The **uep** driver was written by Tyler C. Sarna for NetBSD. The **uep** driver appeared in NetBSD 3.0.

BUGS

uep currently does not support calibration. Also, not all NetBSD-supplied X servers support the absolute position events it generates.

NAME

uftdi — USB support for serial adapters based on the FT8U100AX and FT8U232AM

SYNOPSIS

```
uftdi* at uhub?
ucom*  at uftdi?
```

HARDWARE

The **uftdi** driver supports the following adapters:

- B&B Electronics uLinks RS-422/485
- Coastal ChipWorks TNC-X
- Crystalfontz CFA-631 LCD
- Crystalfontz CFA-632 LCD
- Crystalfontz CFA-633 LCD
- Crystalfontz CFA-634 LCD
- Crystalfontz CFA-635 LCD
- Falcom Samba 55/56 GSM/GPRS modem
- Falcom Twist GSM/GPRS modem
- Future Technology Devices DB9
- Future Technology Devices IC
- Future Technology Devices KW
- Future Technology Devices RS232
- Future Technology Devices Y6
- Future Technology Devices Y8
- Future Technology Devices Y9
- Future Technology Devices YS
- HP USB-Serial adapter shipped with some HP laptops
- Inland UAS111
- Interpid Control Systems NeoVI Blue
- Interpid Control Systems ValueCAN
- Matrix Orbital LK/VK/PK202-24 LCD
- Matrix Orbital LK/VK204-24 LCD
- Matrix Orbital MX2/MX3/MX6 Series
- Matrix Orbital MX4/MX5 Series LCD
- QVS USC-1000
- SIIG US2308 Serial
- Sealevel Systems USB-Serial adapter

DESCRIPTION

The **uftdi** driver provides support for various serial adapters based on the FTDI FT8U100AX and FT8U232AM chips.

The device is accessed through the ucom(4) driver which makes it behave like a tty(4).

SEE ALSO

tty(4), ucom(4), usb(4)

HISTORY

The **uftdi** driver appeared in NetBSD 1.5.

NAME

ug — Abit uGuru system hardware monitor

SYNOPSIS

```
ug* at acpi?
ug0 at isa? port 0xe0
```

DESCRIPTION

The **ug** driver provides support for the Abit uGuru hardware monitor to be used with the envsys(4) interface.

The **ug** driver has 19 sensors:

Sensor	Units	Typical Use
CPU	uK	CPU Temp
SYS	uK	System Temp
PWN	uK	PWN Temp
VCORE	uV DC	Core voltage
DDR0	uV DC	DDRVdd
DDR1	uV DC	DDRVtt
NB	uV DC	NBVdd
SB	uV DC	SBVdd
HTV	uV DC	HTVdd
AGP	uV DC	AGPVdd
V5V	uV DC	Vdd5V
V33V	uV DC	Vdd3V3
V5SB	uV DC	Vdd5VSB
V33VD	uV DC	Vdd3VDual
CPUF	RPM	CPU Fan
NBF	RPM	NB Fan
SYSF	RPM	SYS Fan
AUXF	RPM	AUX Fan 1
AUXF2	RPM	AUX Fan 2

SEE ALSO

acpi(4), envsys(4), envstat(8)

HISTORY

The **ug** driver first appeared in NetBSD 4.0.

AUTHORS

The **ug** driver was written by Mihai Chelaru ⟨kefren@netbsd.ro⟩.

BUGS

Interrupt support is unimplemented.

NAME

ugen — USB generic device support

SYNOPSIS

```
ugen* at uhub? flags N
options UGEN_BULK_RA_WB
```

DESCRIPTION

The **ugen** driver provides support for all USB devices that do not have a special driver. It supports access to all parts of the device, but not in a way that is as convenient as a special purpose driver.

Normally the **ugen** driver is used when no other driver attaches to a device. If "flags 1" is specified, the **ugen** will instead attach with a very high priority and always be used. Together with the **vendor** and **product** locators this can be used to force the **ugen** driver to be used for a certain device.

There can be up to 127 USB devices connected to a USB bus. Each USB device can have up to 16 endpoints. Each of these endpoints will communicate in one of four different modes: control, isochronous, bulk, or interrupt. Each of the endpoints will have a different device node. The four least significant bits in the minor device number determines which endpoint the device accesses and the rest of the bits determines which USB device.

If an endpoint address is used both for input and output the device can be opened for both read or write.

To find out what endpoints exist there are a series of `ioctl`(2) operations on the control endpoint that return the USB descriptors of the device, configurations, interfaces, and endpoints.

The control transfer mode can only happen on the control endpoint which is always endpoint 0. The control endpoint accepts requests and may respond with an answer to such requests. Control requests are issued by `ioctl`(2) calls.

The bulk transfer mode can be in or out depending on the endpoint. To perform IO on a bulk endpoint `read`(2) and `write`(2) should be used. All IO operations on a bulk endpoint are normally unbuffered. On kernels built with the `UGEN_BULK_RA_WB` option, the `USB_SET_BULK_RA` and `USB_SET_BULK_WB` `ioctl`(2) calls are available, and enable read-ahead and write-behind buffering respectively. When read-ahead or write-behind are enabled, the file descriptor may be set to use non-blocking IO.

When in a `UGEN_BULK_RA_WB` mode, `select`(2) for read and write operates normally, returning true if there is data in the read buffer and space in the write buffer, respectively. When not in a `UGEN_BULK_RA_WB` mode, `select`(2) always returns true, because there is no way to predict how the device will respond to a read or write request.

The interrupt transfer mode can be in or out depending on the endpoint. To perform IO on an interrupt endpoint `read`(2) and `write`(2) should be used. A moderate amount of buffering is done by the driver.

All endpoints handle the following `ioctl`(2) calls:

`USB_SET_SHORT_XFER (int)`
: Allow short read transfer. Normally a transfer from the device which is shorter than the request specified is reported as an error.

`USB_SET_TIMEOUT (int)`
: Set the timeout on the device operations, the time is specified in milliseconds. The value 0 is used to indicate that there is no timeout.

The control endpoint (endpoint 0) handles the following `ioctl`(2) calls:

`USB_GET_CONFIG (int)`
: Get the device configuration number.

`USB_SET_CONFIG (int)`
: Set the device into the given configuration number.
This operation can only be performed when the control endpoint is the sole open endpoint.

`USB_GET_ALTINTERFACE (struct usb_alt_interface)`
: Get the alternative setting number for the interface with the given index. The `config_index` is ignored in this call.

```
struct usb_alt_interface {
        int     uai_config_index;
        int     uai_interface_index;
        int     uai_alt_no;
};
```

`USB_SET_ALTINTERFACE (struct usb_alt_interface)`
: Set the alternative setting to the given number in the interface with the given index. The `uai_config_index` is ignored in this call.
This operation can only be performed when no endpoints for the interface are open.

`USB_GET_NO_ALT (struct usb_alt_interface)`
: Return the number of different alternate settings in the `aui_alt_no` field.

`USB_GET_DEVICE_DESC (usb_device_descriptor_t)`
: Return the device descriptor.

`USB_GET_CONFIG_DESC (struct usb_config_desc)`
: Return the descriptor for the configuration with the given index. For convenience the current configuration can be specified by `USB_CURRENT_CONFIG_INDEX`.

```
struct usb_config_desc {
        int     ucd_config_index;
        usb_config_descriptor_t ucd_desc;
};
```

`USB_GET_INTERFACE_DESC (struct usb_interface_desc)`
: Return the interface descriptor for an interface specified by its configuration index, interface index, and alternative index. For convenience the current alternative can be specified by `USB_CURRENT_ALT_INDEX`.

```
struct usb_interface_desc {
        int     uid_config_index;
        int     uid_interface_index;
        int     uid_alt_index;
        usb_interface_descriptor_t uid_desc;
};
```

`USB_GET_ENDPOINT_DESC (struct usb_endpoint_desc)`
: Return the endpoint descriptor for the endpoint specified by its configuration index, interface index, alternative index, and endpoint index.

```
struct usb_endpoint_desc {
        int     ued_config_index;
        int     ued_interface_index;
        int     ued_alt_index;
        int     ued_endpoint_index;
        usb_endpoint_descriptor_t ued_desc;
};
```

`USB_GET_FULL_DESC (struct usb_full_desc)`
Return all the descriptors for the given configuration.

```
struct usb_full_desc {
        int     ufd_config_index;
        u_int   ufd_size;
        u_char  *ufd_data;
};
```

The `ufd_data` field should point to a memory area of the size given in the `ufd_size` field. The proper size can be determined by first issuing a `USB_GET_CONFIG_DESC` and inspecting the `wTotalLength` field.

`USB_GET_STRING_DESC (struct usb_string_desc)`
Get a string descriptor for the given language id and string index.

```
struct usb_string_desc {
        int     usd_string_index;
        int     usd_language_id;
        usb_string_descriptor_t usd_desc;
};
```

`USB_DO_REQUEST`
Send a USB request to the device on the control endpoint. Any data sent to/from the device is located at `data`. The size of the transferred data is determined from the `request`. The `ucr_addr` field is ignored in this call. The `ucr_flags` field can be used to flag that the request is allowed to be shorter than the requested size, and the `ucr_actlen` field will contain the actual size on completion.

```
struct usb_ctl_request {
        int     ucr_addr;
        usb_device_request_t ucr_request;
        void    *ucr_data;
        int     ucr_flags;
#define USBD_SHORT_XFER_OK    0x04    /* allow short reads */
        int     ucr_actlen;           /* actual length transferred */
};
```

This is a dangerous operation in that it can perform arbitrary operations on the device. Some of the most dangerous (e.g., changing the device address) are not allowed.

`USB_GET_DEVICEINFO (struct usb_device_info)`
Get an information summary for the device. This call will not issue any USB transactions.

Bulk endpoints handle the following `ioctl`(2) calls:

`USB_SET_BULK_RA (int)`
Enable or disable bulk read-ahead. When enabled, the driver will begin to read data from the device into a buffer, and will perform reads from the device whenever there is room in the buffer. The `read`(2) call will read data from this buffer, blocking if necessary until there is enough data to read the length of data requested. The buffer size and the read request length can be set by the `USB_SET_BULK_RA_OPT` `ioctl`(2) call.

`USB_SET_BULK_WB (int)`
Enable or disable bulk write-behind. When enabled, the driver will buffer data from the `write`(2) call before writing it to the device, enabling the `write`(2) call to return immediately. `write`(2) will block if there is not enough room in the buffer for all the data. The buffer size and the write request length can be set by the `USB_SET_BULK_WB_OPT` `ioctl`(2) call.

`USB_SET_BULK_RA_OPT (struct usb_bulk_ra_wb_opt)`
: Set the size of the buffer and the length of the read requests used by the driver when bulk read-ahead is enabled. The changes do not take effect until the next time bulk read-ahead is enabled. Read requests are made for the length specified, and the host controller driver (i.e., `ehci`(4), `ohci`(4), and `uhci`(4)) will perform as many bus transfers as required. If transfers from the device should be smaller than the maximum length, `ra_wb_request_size` must be set to the required length.

```
struct usb_bulk_ra_wb_opt {
        u_int   ra_wb_buffer_size;
        u_int   ra_wb_request_size;
};
```

`USB_SET_BULK_WB_OPT (struct usb_bulk_ra_wb_opt)`
: Set the size of the buffer and the length of the write requests used by the driver when bulk write-behind is enabled. The changes do not take effect until the next time bulk write-behind is enabled.

Note that there are two different ways of addressing configurations, interfaces, alternatives, and endpoints: by index or by number. The index is the ordinal number (starting from 0) of the descriptor as presented by the device. The number is the respective number of the entity as found in its descriptor. Enumeration of descriptors use the index, getting and setting typically uses numbers.

Example: All endpoints (except the control endpoint) for the current configuration can be found by iterating the `interface_index` from 0 to `config_desc->bNumInterface-1` and for each of these iterating the `endpoint_index` from 0 to `interface_desc->bNumEndpoints`. The `config_index` should set to `USB_CURRENT_CONFIG_INDEX` and `alt_index` should be set to `USB_CURRENT_ALT_INDEX`.

FILES

`/dev/ugenN.EE` Endpoint `EE` of device `N`.

SEE ALSO

`usb`(4)

HISTORY

The **ugen** driver appeared in NetBSD 1.4.

NAME

ugensa — USB generic serial adapter

SYNOPSIS

```
ugensa*  at uhub?
ucom*  at ugensa?
```

HARDWARE

The **ugensa** driver supports the following adapters (or their USB device portion, for adaptors that contain a USB host controller, hub and a USB device):

Airprime PC5220
Novatel FlexPak GPS receiver
Qualcom CDMA MSM (found in Kyocera KPC650 EVDO interface)
Sierra AirCard 580
Sierra AirCard 595
Sierra AirCard 875 [not tested]
Sierra Wireless EM5625 [not tested]
Sierra Wireless MC5720 [not tested]
Sierra Wireless MC5725 [not tested]
Sierra Wireless MC8755 [not tested]
Sierra Wireless MC8755 [not tested]
Sierra Wireless MC8755 [not tested]
Sierra Wireless MC8765 [not tested]
Sierra Wireless MC8775 [not tested]
Novatel Wireless Merlin CDMA (found in Verizon V620) [not tested]
Novatel ExpressCard [not tested]
Novatel Wireless S720 [not tested]
Novatel Ovation U720 [not tested]
Novatel Merlin XU870 HSDPA ExpressCard [not tested]
Novatel Merlin ES620 [not tested]
Dell/Novatel Wireless HDSPA modem
Dell W5500 HDSPA modem [not tested]
AnyDATA ADU-E500A [not tested]

DESCRIPTION

The **ugensa** driver provides support for the USB-to-RS-232 Bridge chip used by various vendors.

The device is accessed through the `ucom`(4) driver which makes it behave like a `tty`(4).

SEE ALSO

`tty`(4), `ucom`(4), `usb`(4)

HISTORY

The **ugensa** driver first appeared in NetBSD 3.0.

NAME

uha — Ultrastor SCSI bus adapter for ISA/EISA bus

SYNOPSIS

```
uha0 at isa? port 0x330 irq ? drq ?
uha* at eisa? slot ? irq ?
scsibus* at uha?
```

DESCRIPTION

The **uha** driver provides support for the following SCSI adapters:

Ultrastor ISA 14f
Ultrastor EISA 24f
Ultrastor ISA 34f

SEE ALSO

cd(4), ch(4), eisa(4), intro(4), isa(4), scsi(4), sd(4), st(4)

NAME

uhci — USB Universal Host Controller driver

SYNOPSIS

```
uhci*    at cardbus? function ?
uhci*    at pci? dev ? function ?
usb*     at uhci?
```

DESCRIPTION

The **uhci** driver provides support for USB Universal Host Controller Interface.

SEE ALSO

cardbus(4), pci(4), usb(4)

HISTORY

The **uhci** driver appeared in NetBSD 1.4.

NAME

uhid — USB generic HID support

SYNOPSIS

```
uhid* at uhidev? reportid ? flags N
```

DESCRIPTION

The **uhid** driver provides support for all HID (Human Interface Device) interfaces in USB devices that do not have a special driver.

Normally the **uhid** driver is used when no other HID driver attaches to a device. If "flags 1" is specified, the **uhid** driver will instead attach with a very high priority and always be used. Together with the **vendor** and **product** locators on the uhidev(4) driver this can be used to force the **uhid** driver to be used for a certain device.

The device handles the following ioctl(2) calls:

USB_GET_REPORT_ID (int)
: Get the report identifier used by this HID report.

USB_GET_REPORT_DESC (struct usb_ctl_report_desc)
: Get the HID report descriptor. Using this descriptor the exact layout and meaning of data to/from the device can be found. The report descriptor is delivered without any processing.

```
struct usb_ctl_report_desc {
    int     ucrd_size;
    u_char  ucrd_data[1024];  /* filled data size will vary */
};
```

USB_SET_IMMED (int)
: Sets the device in a mode where each read(2) will return the current value of the input report. Normally a read(2) will only return the data that the device reports on its interrupt pipe. This call may fail if the device does not support this feature.

USB_GET_REPORT (struct usb_ctl_report)
: Get a report from the device without waiting for data on the interrupt pipe. The report field indicates which report is requested. It should be UHID_INPUT_REPORT, UHID_OUTPUT_REPORT, or UHID_FEATURE_REPORT. This call may fail if the device does not support this feature.

```
struct usb_ctl_report {
        int     ucr_report;
        u_char  ucr_data[1024];       /* used data size will vary */
};
```

USB_SET_REPORT (struct usb_ctl_report)
: Set a report in the device. The report field indicates which report is to be set. It should be UHID_INPUT_REPORT, UHID_OUTPUT_REPORT, or UHID_FEATURE_REPORT. This call may fail if the device does not support this feature.

USB_GET_DEVICEINFO (struct usb_device_info)
: Get an information summary for the device. This call will not issue any USB transactions.

USB_GET_STRING_DESC (struct usb_string_desc)
: Get a string descriptor for the given language id and string index.

```
struct usb_string_desc {
        int     usd_string_index;
        int     usd_language_id;
        usb_string_descriptor_t usd_desc;
};
```

Use `read`(2) to get data from the device. Data should be read in chunks of the size prescribed by the report descriptor.

Use `write`(2) send data to the device. Data should be written in chunks of the size prescribed by the report descriptor.

FILES

`/dev/uhid?`

SEE ALSO

`usbhidaction`(1), `usbhidctl`(1), `uhidev`(4), `usb`(4)

HISTORY

The **uhid** driver appeared in NetBSD 1.4. Support for the `USB_GET_DEVICEINFO` and `USB_GET_STRING_DESC` ioctls appeared in NetBSD 2.0.

NAME

uhidev — USB Human Interface Device support

SYNOPSIS

```
uhidev* at uhub?
ucycom* at uhidev? reportid ?
uhid*   at uhidev? reportid ?
ukbd*   at uhidev? reportid ?
ums*    at uhidev? reportid ?
```

DESCRIPTION

The **uhidev** driver handles all Human Interface Devices. Each HID device can have several components, e.g., a keyboard and a mouse. These components use different report identifiers (a byte) to distinguish which one data is coming from. The **uhidev** driver has other drivers attached that handle particular kinds of devices and **uhidev** only dispatches data to them based on the report id.

SEE ALSO

ucycom(4), uhid(4), ukbd(4), ums(4), usb(4)

HISTORY

The **uhidev** driver appeared in NetBSD 1.6.

NAME

uhmodem — USB support for Huawei 3G wireless modem device

SYNOPSIS

```
uhmodem*  at uhub?
ucom*  at uhmodem?
```

HARDWARE

The **uhmodem** driver supports the following adapters:

Huawei E220
E-mobile D01HW
E-mobile D02HW
NTT DoCoMo a2502

DESCRIPTION

The **uhmodem** driver provides support for the Huawei 3G wireless modems and its variants. This type of device has multiple com ports. And some modems have own storage to contain its device driver (for Windows). The **uhmodem** driver can handle all of these functions on the device.

When the device attached, it will attach multiple ucom devices. The former one is main com device to use communication with your network provider, and others are sub com device to monitor the condition of data communication and/or network status. These com devices can be used simultaneously.

The device is accessed through the ucom(4) driver which makes it behave like a tty(4).

SEE ALSO

tty(4), ubsa(4), ucom(4), usb(4)

HISTORY

The **uhmodem** driver first appeared in NetBSD 5.0. Separate from ubsa driver.

NAME

uipaq — USB support for iPAQ units

SYNOPSIS

```
uipaq* at uhub?
ucom*  at uipaq?
```

HARDWARE

The **uipaq** driver supports the following adapters:

HP iPAQ 22xx/Jornada 548
HP Jornada 568
Compaq IPaq PocketPC
Casio BE300

DESCRIPTION

The **uipaq** driver provides support for the USB serial emulation provided by the iPAQ devices.

The device is accessed through the ucom(4) driver which makes it behave like a tty(4).

SEE ALSO

tty(4), ucom(4), uhub(4), usb(4)

HISTORY

The NetBSD support was added in NetBSD 4.0 and it was imported from OpenBSD 3.8.

NAME

uirda — USB-IrDA bridge support

SYNOPSIS

```
uirda*   at uhub?
irframe* at uirda?
```

DESCRIPTION

The **uirda** driver provides support for USB-IrDA bridges that follow the bridge specification and also the following adapters:

ACTiSYS ACT-IR2000U FIR-USB Adapter
Kawatsu KC-180 USB IrDA Device
Extended Systems XTNDAccess IrDA USB (ESI-9685)

Access to the IrDA device is through the irframe(4) driver.

SEE ALSO

irframe(4), usb(4)

HISTORY

The **uirda** driver appeared in NetBSD 1.6.

NAME

uk — SCSI user-level driver

SYNOPSIS

```
uk* at scsibus? target ? lun ?
```

DESCRIPTION

The **uk** driver provides support for a process to address devices on the SCSI bus for which there is no configured driver.

A SCSI adapter must also be separately configured into the system before this driver makes sense.

KERNEL CONFIGURATION

If a *count* is given, that number of **uk** devices will be configured into the NetBSD kernel.

IOCTLS

The **uk** driver has no ioctls of its own but rather acts as a medium for the generic `scsi`(4) ioctls. These are described in ⟨`sys/scsiio.h`⟩.

FILES

`/dev/uk[0-255]` unknown SCSI devices.

DIAGNOSTICS

All `scsi`(4) debug ioctls work on **uk** devices.

SEE ALSO

`ioctl`(2), `cd`(4), `ch`(4), `scsi`(4), `sd`(4), `ss`(4), `st`(4)

HISTORY

The **uk** driver appeared in 386BSD 0.1.

NAME

ukbd — USB keyboard support

SYNOPSIS

```
ukbd*  at uhidev? reportid ?
wskbd* at ukbd? console ?
```

DESCRIPTION

The **ukbd** driver provides support for USB keyboards (i.e., HID devices with reports in usage page 7). Access to the keyboard is through the wscons(4) driver.

SEE ALSO

uhidev(4), usb(4), wskbd(4)

HISTORY

The **ukbd** driver appeared in NetBSD 1.4.

BUGS

The **ukbd** driver is brought into action rather late in the boot process, so if it is used as the console driver then ddb(4) is not usable until late in the boot.

NAME

ukphy — Driver for generic/unknown IEEE 802.3u Ethernet PHYs

SYNOPSIS

ukphy* **at mii? phy ?**

DESCRIPTION

The **ukphy** driver supports the basic functionality of most Ethernet PHYs.

SEE ALSO

`ifmedia`(4), `intro`(4), `mii`(4), `ifconfig`(8)

BUGS

In areas where the programming interface for PHYs differ, such as detecting the currently active medium, this driver often does not work optimally. When available, it is best to use a PHY-specific driver.

NAME

ukyopon — Kyocera AIR-EDGE PHONE support

SYNOPSIS

```
ukyopon* at uhub?
ucom*    at ukyopon? portno ?

#include <dev/usb/ukyopon.h>
```

DESCRIPTION

The **ukyopon** driver provides support for Kyocera AIR-EDGE PHONE AH-K3001V.

Two units of this driver attach to an AIR-EDGE PHONE: the modem port and the data transfer port. The modem port is compatible to umodem(4), and can be used for dialup connections. The data transfer port is for reading and writing internal storage of AIR-EDGE PHONE.

Both devices are accessed through the ucom(4) driver which makes them behave like a tty(4).

The manipulation of the internal storage is through external programs, for example, the pkgsrc/comms/kyopon package.

IOCTLS

The following ioctl(2) calls apply to the **ukyopon** device:

UKYOPON_IDENTIFY *struct ukyopon_identify*

Read, from the kernel, the identification information of the device, useful to assure that the opened device node is a modem or a data transfer port of **ukyopon** device.

```
struct ukyopon_identify {
        char    ui_name[16];            /* driver name */

        int     ui_busno;               /* usb bus number */
        uint8_t ui_address;             /* device address */

        enum ukyopon_model {
                UKYOPON_MODEL_UNKNOWN
        } ui_model;                     /* possibly future use */
        enum ukyopon_port {
                UKYOPON_PORT_UNKNOWN,
                UKYOPON_PORT_MODEM,     /* modem port */
                UKYOPON_PORT_DATA       /* data transfer port */
        } ui_porttype;                  /* port type */
        int     ui_rsvd1, ui_rsvd2;
};
#define UKYOPON_NAME          "ukyopon"
```

The *ui_name* field contains the driver signature, and has the string UKYOPON_NAME.

The *ui_busno* field contains the usb(4) bus number to which the device is connected; the *ui_address* field contains the address of the device in the bus. These fields are useful to identify the physical device from the file descriptor.

The *ui_porttype* field contains the type of device: UKYOPON_PORT_MODEM means the device is associated to the modem port, and UKYOPON_PORT_DATA means the device is associated to the data transfer port.

Other fields are reserved for future extension and cleared to zeros.

In addition, **ukyopon** devices accept all `ioctl`(2) calls that `umodem`(4) accepts.

SEE ALSO

`tty`(4), `ucom`(4), `umodem`(4), `usb`(4), `pkgsrc/comms/kyopon`

HISTORY

The **ukyopon** driver appeared in NetBSD 3.0.

NOTES

“Kyopon” is a widely-used nickname of Kyocera AIR-EDGE PHONE.

NAME

ulpt — USB printer support

SYNOPSIS

ulpt* **at uhub?**

DESCRIPTION

The **ulpt** driver provides support for USB printers that follow the printer bi- or uni-directional protocol. The bits in the minor number select various features of the driver.

Minor Bit	Function
64	Do not initialize (reset) the device on the port.

Some printers cannot handle the reset on open; in case of problems try the `ulpn` device.

FILES

`/dev/ulpt?` device with reset

`/dev/ulpn?` device without reset

SEE ALSO

`lpt`(4), `usb`(4)

HISTORY

The **ulpt** driver appeared in NetBSD 1.4.

NAME

umass — USB mass storage support

SYNOPSIS

```
umass*      at uhub?
atapibus*   at umass?
scsibus*    at umass? channel ?
wd*         at umass?
```

DESCRIPTION

The **umass** driver provides support for USB mass storage class devices. The driver supports subclasses SCSI, 8020i (ATAPI), 8070i (ATAPI), QIC157 (ATAPI), RBC (subset of SCSI), and UFI (USB Floppy). The driver handles the protocols Bulk-Only, CBI, and CBI-with-CCI.

HARDWARE

Devices known to work with this driver include:

Creative Tech NOMAD MuVo TX
FujiFilm FinePix1300 Digital Camera
Imation SuperDisk
IOmega Zip 100
IOmega Zip 250
Kingston DataTraveler 2.0
LaCie CD R/W
Lexar Media JumpDrive Flash Disks
Neodio Multi-format Flash Controller
Olympus C-5050 ZOOM Digital Camera
SanDisk ImageMate SDDR-31
Siemens MP3-Player USB
Sony DSC Digital Cameras
STMicroelectronics ST92163 Mass Storage library Tester
Y-E Data FlashBuster floppy

The driver also supports certain adapters from In-System Design that use the (non-standardized) ATA protocol over Bulk-Only.

SEE ALSO

atapi(4), scsi(4), usb(4), wd(4)

HISTORY

The **umass** driver is a port of the FreeBSD driver. The **umass** driver appeared in NetBSD 1.5.

NAME

umct — USB support for MCT USB RS-232 serial adapters driver

SYNOPSIS

```
umct*  at uhub?
ucom*  at umct?
```

HARDWARE

The **umct** driver supports the following adapters:

Belkin F5U109
D-Link DU-H3SP USB BAY
MCT USB RS-232 Converter 25 pin Model U232-P25
MCT USB RS-232 Converter 9 pin Model U232-P9
Sitecom USB RS-232 Converter

DESCRIPTION

The **umct** driver provides support for MCT USB-to-RS-232 Converter and their family products.

The device is accessed through the ucom(4) driver which makes it behave like a tty(4).

SEE ALSO

tty(4), ucom(4), usb(4)

HISTORY

The **umct** driver appeared in NetBSD 1.6.

NAME

umidi — USB support for MIDI devices

SYNOPSIS

```
umidi* at uhub?
midi*  at umidi?
```

DESCRIPTION

The **umidi** driver supports USB MIDI devices that conform to the *Universal Serial Bus Device Class Definition for MIDI Devices*. Vendor-specific support is also included for the following:

Midiman devices

MidiSport 2x4

Roland and Edirol devices

Fantom-X
PC300
PCR
SC8820
SC8850
SCD70
SD20
SD80
SD90
SK500
SonicCell
U8
UA25
UA100
UA101
UA10F
UA4FX
UA700
UA1000
UM1
UM2
UM3
UM4
UM550
UM880N
XV5050

Yamaha devices

UX256	(product-specific support)
Others	Other Yamaha MIDI devices will be attached and are expected to work also.

Devices supported by the **umidi** driver will appear as midi(4) devices.

SEE ALSO

midi(4), usb(4)

HISTORY

The **umidi** driver appeared in NetBSD 1.6.

NAME

umodem — USB modem support

SYNOPSIS

```
umodem* at uhub?
ucom*   at umodem?
```

DESCRIPTION

The **umodem** driver provides support for USB modems in the Communication Device Class using the Abstract Control Model. These modems are basically standard serial line modems, but they are accessed via USB instead. They support a regular AT command set. The commands can either be multiplexed with the data stream or handled through separate pipes. In the latter case the AT commands have to be given on a device separate from the data device.

The device is accessed through the ucom(4) driver which makes it behave like a tty(4).

SEE ALSO

tty(4), ucom(4), usb(4)

HISTORY

The **umodem** driver appeared in NetBSD 1.5.

BUGS

Only modems with multiplexed commands and data are supported at the moment.

NAME

ums — USB mouse support

SYNOPSIS

```
ums*     at uhidev? reportid ?
wsmouse* at ums?
```

DESCRIPTION

The **ums** driver provides support for USB mice. Access to the mouse is through the wscons(4) driver.

SEE ALSO

uhidev(4), usb(4), wsmouse(4)

HISTORY

The **ums** driver appeared in NetBSD 1.4.

NAME

unix — UNIX-domain protocol family

SYNOPSIS

```
#include <sys/types.h>
#include <sys/un.h>
```

DESCRIPTION

The UNIX-domain protocol family is a collection of protocols that provides local (on-machine) interprocess communication through the normal socket(2) mechanisms. The UNIX-domain family supports the SOCK_STREAM and SOCK_DGRAM socket types and uses filesystem pathnames for addressing.

ADDRESSING

UNIX-domain addresses are variable-length filesystem pathnames of at most 104 characters. The include file ⟨sys/un.h⟩ defines this address:

```
struct sockaddr_un {
        u_char  sun_len;
        u_char  sun_family;
        char    sun_path[104];
};
```

Binding a name to a UNIX-domain socket with bind(2) causes a socket file to be created in the filesystem. This file is *not* removed when the socket is closed—unlink(2) must be used to remove the file.

The length of UNIX-domain address, required by bind(2) and connect(2), can be calculated by the macro **SUN_LEN**() defined in ⟨sys/un.h⟩. The *sun_path* field must be terminated by a NUL character to be used with **SUN_LEN**(), but the terminating NUL is *not* part of the address. The NetBSD kernel ignores any user-set value in the *sun_len* member of the structure.

The UNIX-domain protocol family does not support broadcast addressing or any form of "wildcard" matching on incoming messages. All addresses are absolute- or relative-pathnames of other UNIX-domain sockets. Normal filesystem access-control mechanisms are also applied when referencing pathnames; e.g., the destination of a connect(2) or sendto(2) must be writable.

PROTOCOLS

The UNIX-domain protocol family comprises simple transport protocols that support the SOCK_STREAM and SOCK_DGRAM abstractions. SOCK_STREAM sockets also support the communication of UNIX file descriptors through the use of the *msg_control* field in the *msg* argument to sendmsg(2) and recvmsg(2).

Any valid descriptor may be sent in a message. The file descriptor(s) to be passed are described using a *struct cmsghdr* that is defined in the include file ⟨sys/socket.h⟩. The type of the message is SCM_RIGHTS, and the data portion of the messages is an array of integers representing the file descriptors to be passed. The number of descriptors being passed is defined by the length field of the message; the length field is the sum of the size of the header plus the size of the array of file descriptors.

The received descriptor is a *duplicate* of the sender's descriptor, as if it were created with a call to dup(2). Per-process descriptor flags, set with fcntl(2), are *not* passed to a receiver. Descriptors that are awaiting delivery, or that are purposely not received, are automatically closed by the system when the destination socket is closed.

There are two socket-level setsockopt(2) /getsockopt(2) option available in the **unix** domain:

The LOCAL_CREDS option may be enabled on a SOCK_DGRAM or a SOCK_STREAM socket. This option provides a mechanism for the receiver to receive the credentials of the process as a recvmsg(2) control message. The msg_control field in the msghdr structure points to a buffer that contains a cmsghdr structure followed by a variable length sockcred structure, defined in ⟨sys/socket.h⟩ as follows:

```
struct sockcred {
        uid_t   sc_uid;                 /* real user id */
        uid_t   sc_euid;                /* effective user id */
        gid_t   sc_gid;                 /* real group id */
        gid_t   sc_egid;                /* effective group id */
        int     sc_ngroups;             /* number of supplemental groups */
        gid_t   sc_groups[1];           /* variable length */
};
```

The LOCAL_PEEREID option may be used with getsockopt(2) to get the PID and effective user and group IDs of a SOCK_STREAM peer when it did connect(2) or bind(2). The returned structure is

```
struct unpcbid {
        pid_t unp_pid;                  /* process id */
        uid_t unp_euid;                         /* effective user id */
        gid_t unp_egid;                         /* effective group id */
};
```

as defined in ⟨sys/un.h⟩.

The **SOCKCREDSIZE**() macro computes the size of the sockcred structure for a specified number of groups. The cmsghdr fields have the following values:

```
cmsg_len = sizeof(struct cmsghdr) + SOCKCREDSIZE(ngroups)
cmsg_level = SOL_SOCKET
cmsg_type = SCM_CREDS
```

EXAMPLES

The following code fragment shows how to bind a socket to pathname:

```
const char *pathname = "/path/to/socket";
struct sockaddr_un addr;
int ret;

memset(&addr, 0, sizeof(addr));
addr.sun_family = AF_LOCAL;
if (strlen(pathname) ≥ sizeof(addr.sun_path))
        goto too_long;
strncpy(addr.sun_path, pathname, sizeof(addr.sun_path));
ret = bind(s, (const struct sockaddr *)&addr, SUN_LEN(&addr));
if (ret != 0)
        goto bind_failed;
...
```

COMPATIBILITY

The *sun_len* field exists only in system derived from 4.4BSD. On systems which don't have the **SUN_LEN**() macro, the following definition is recommended:

```
#ifndef SUN_LEN
#define SUN_LEN(su)    sizeof(struct(sockaddr_un))
```

```
#endif
```

SEE ALSO

`socket`(2), `intro`(4)

Stuart Sechrest, *An Introductory 4.4BSD Interprocess Communication Tutorial*. (see `/usr/share/doc/psd/20.ipctut`)

Samuel J. Leffler, Robert S. Fabry, William N. Joy, Phil Lapsley, Steve Miller, and Chris Torek, *Advanced 4.4BSD IPC Tutorial*. (see `/usr/share/doc/psd/21.ipc`)

NAME

unixbp — Unix Backplane driver

SYNOPSIS

```
unixbp0 at ioc0 bank 6
```

DESCRIPTION

The **unixbp** driver controls the interrupt-masking hardware on the expansion-card backplane of certain systems. This hardware gives NetBSD the ability to selectively block interrupts from particular expansion cards, and to determine which cards are currently generating interrupts by examining a single register.

SEE ALSO

podulebus(4)

NAME

up — UNIBUS storage module controller/disk drives

SYNOPSIS

```
sc0 at uba? csr 0176700 vector upintr
up0 at sc0 drive 0
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

This is a generic UNIBUS storage module disk driver. It is specifically designed to work with the Emulex SC-21 and SC-31 controllers. It can be easily adapted to other controllers (although bootstrapping will not necessarily be directly possible.)

The script MAKEDEV(8) should be used to create the **up** special files; consult mknod(8) if a special file needs to be made manually. It is recommended as a security precaution to not create special files for devices which may never be installed.

DISK SUPPORT

The driver interrogates the controller's holding register to determine the type of drive attached. The driver recognizes seven different drives: CDC 9762, CDC 9766, AMPEX DM980, AMPEX 9300, AMPEX Capricorn, FUJITSU 160, and FUJITSU Eagle (the Eagle is not supported by the SC-21).

Special file names begin with 'up' and 'rup' for the block and character files respectively. The second component of the name, a drive unit number in the range of zero to seven, is represented by a '?' in the disk layouts below. The last component of the name, the file system partition, is designated by a letter from 'a' to 'h' which also corresponds to a minor device number set: zero to seven, eight to 15, 16 to 23 and so forth for drive zero, drive two and drive three respectively (see physio(4)). The location and size (in 512 byte sectors) of the partitions for the above drives:

CDC 9762 partitions

disk	**start**	**length**	**cyls**
hp?a	0	15884	0-99
hp?b	16000	33440	100-309
hp?c	0	131680	0-822
hp?d	49600	15884	309-408
hp?e	65440	55936	409-758
hp?f	121440	10080	759-822
hp?g	49600	82080	309-822

CDC 9766 300M drive partitions

disk	**start**	**length**	**cyl**
up?a	0	15884	0-26
up?b	16416	33440	27-81
up?c	0	500384	0-822
up?d	341696	15884	562-588
up?e	358112	55936	589-680
up?f	414048	861760	681-822
up?g	341696	158528	562-822
up?h	49856	291346	82-561

AMPEX DM980 partitions

disk	**start**	**length**	**cyls**
hp?a	0	15884	0-99
hp?b	16000	33440	100-309

hp?c	0	131680	0-822
hp?d	49600	15884	309-408
hp?e	65440	55936	409-758
hp?f	121440	10080	759-822
hp?g	49600	82080	309-822

AMPEX 9300 300M drive partitions

disk	**start**	**length**	**cyl**
up?a	0	15884	0-26
up?b	16416	33440	27-81
up?c	0	495520	0-814
up?d	341696	15884	562-588
up?e	358112	55936	589-680
up?f	414048	81312	681-814
up?g	341696	153664	562-814
up?h	49856	291346	82-561

AMPEX Capricorn 330M drive partitions

disk	**start**	**length**	**cyl**
hp?a	0	15884	0-31
hp?b	16384	33440	32-97
hp?c	0	524288	0-1023
hp?d	342016	15884	668-699
hp?e	358400	55936	700-809
hp?f	414720	109408	810-1023
hp?g	342016	182112	668-1023
hp?h	50176	291346	98-667

FUJITSU 160M drive partitions

disk	**start**	**length**	**cyl**
up?a	0	15884	0-49
up?b	16000	33440	50-154
up?c	0	263360	0-822
up?d	49600	15884	155-204
up?e	65600	55936	205-379
up?f	121600	141600	380-822
up?g	49600	213600	155-822

FUJITSU Eagle partitions

disk	**start**	**length**	**cyls**
hp?a	0	15884	0-16
hp?b	16320	66880	17-86
hp?c	0	808320	0-841
hp?d	375360	15884	391-407
hp?e	391680	55936	408-727
hp?f	698880	109248	728-841
hp?g	375360	432768	391-841
hp?h	83520	291346	87-390

The up?a partition is normally used for the root file system, the up?b partition as a paging area, and the up?c partition for pack-pack copying (it maps the entire disk). On 160M drives the up?g partition maps the rest of the pack. On other drives both up?g and up?h are used to map the remaining cylinders.

FILES

`/dev/up[0-7][a-h]`	block files
`/dev/rup[0-7][a-h]`	raw files

DIAGNOSTICS

up%d%c: hard error %sing fsbn %d[-%d] cs2=%b er1=%b er2=%b. An unrecoverable error occurred during transfer of the specified filesystem block number(s), which are logical block numbers on the indicated partition. The contents of the cs2, er1 and er2 registers are printed in octal and symbolically with bits decoded. The error was either unrecoverable, or a large number of retry attempts (including offset positioning and drive recalibration) could not recover the error.

up%d: write locked. The write protect switch was set on the drive when a write was attempted. The write operation is not recoverable.

up%d: not ready. The drive was spun down or off line when it was accessed. The i/o operation is not recoverable.

up%d: not ready (flakey). The drive was not ready, but after printing the message about being not ready (which takes a fraction of a second) was ready. The operation is recovered if no further errors occur.

up%d%c: soft ecc reading fsbn %d[-%d]. A recoverable ECC error occurred on the specified sector of the specified disk partition. This happens normally a few times a week. If it happens more frequently than this the sectors where the errors are occurring should be checked to see if certain cylinders on the pack, spots on the carriage of the drive or heads are indicated.

sc%d: lost interrupt. A timer watching the controller detecting no interrupt for an extended period while an operation was outstanding. This indicates a hardware or software failure. There is currently a hardware/software problem with spinning down drives while they are being accessed which causes this error to occur. The error causes a UNIBUS reset, and retry of the pending operations. If the controller continues to lose interrupts, this error will recur a few seconds later.

SEE ALSO

hk(4), hp(4), uda(4)

HISTORY

The **up** driver appeared in 4.0BSD.

BUGS

A program to analyze the logged error information (even in its present reduced form) is needed.

The partition tables for the file systems should be read off of each pack, as they are never quite what any single installation would prefer, and this would make packs more portable.

NAME

upl — USB support for Prolific based host-to-host adapters

SYNOPSIS

```
upl* at uhub?
```

DESCRIPTION

The **upl** driver provides support for Prolific PL2301/PL2302 based host-to-host USB connectors.

The **upl** driver appears as a point-to-point network interface and should be configured with `ifconfig`(8) in the same way as other point-to-point interfaces. The **upl** device supports IPv4 only.

HARDWARE

The **upl** driver supports the following adapters from the following vendors:

- Aserton
- Bencent
- Butterfly
- Camtel
- Inland
- Longshine
- PC Linq
- Prolific
- StarMount
- Univex
- VVMer

Belkin, Entrega, and Xircom do *NOT* use this chip and are not supported by this driver.

SEE ALSO

`netintro`(4), `usb`(4), `ifconfig`(8)

HISTORY

The **upl** driver appeared in NetBSD 1.5.

NAME

uplcom — USB support for Prolific PL-2303 serial adapters driver

SYNOPSIS

```
uplcom* at uhub?
ucom*   at uplcom?
```

HARDWARE

The **uplcom** driver supports the following adapters:

BAFO BF-800
BAFO BF-810
ELECOM UC-SGT
HAL Corporation Crossam2
IOGEAR UC-232A
I/O DATA USB-RSAQ
I/O DATA USB-RSAQ2
I/O DATA USB-RSAQ3
I/O DATA USB-RSAQ5
PLANEX USB RS-232 URS-03
Sitecom CN-116 USB to serial
Sony Ericsson DCU-10
Sony Ericsson DCU-11
SOURCENEXT KeiKaiDenwa 8

DESCRIPTION

The **uplcom** driver provides support for the Prolific PL-2303 USB-to-RS-232 Bridge chip.

The device is accessed through the ucom(4) driver which makes it behave like a tty(4).

SEE ALSO

tty(4), ucom(4), usb(4)

HISTORY

The **uplcom** driver appeared in NetBSD 1.6.

NAME

urio — USB driver for the Diamond Multimedia Rio500 MP3 player

SYNOPSIS

urio* **at uhub?**

DESCRIPTION

The **urio** driver provides support for Rio500 MP3 player from Diamond MultiMedia.

The driver implements minimal support to access the device and requires a user application for up and downloading music.

FILES

`/dev/urio0` device node

EXAMPLES

Using the Rio500 package from sourceforge the command

```
rio_add_song 01_BlinkarBla.mp3
```

will download a song over the USB connection into your Rio500.

SEE ALSO

usb(4)
http://rio500.sourceforge.org

HISTORY

The **urio** driver appeared in NetBSD 1.5. This driver is based on a FreeBSD Rio driver written by Iwasa Kazmi.

NAME

url — Realtek RTL8150L USB Ethernet driver

SYNOPSIS

```
url*    at uhub?
urlphy* at mii?
```

DESCRIPTION

The **url** driver provides support for USB Ethernet adapters based on the Realtek RTL8150L USB-ether bridge chip.

The **url** driver supports the following adapters:

CompUSA USBKR100
GreenHouse GH-USB100B
Melco Inc. LUA-KTX
Sitecom LN013

For more information on configuring this device, see `ifconfig`(8).

DIAGNOSTICS

url%d: watchdog timeout A packet was queued for transmission and a transmit command was issued, however the device failed to acknowledge the transmission before a timeout expired.

url%d: no memory for rx list The driver failed to allocate an mbuf for the receiver ring.

SEE ALSO

`arp`(4), `mii`(4), `netintro`(4), `usb`(4), `ifconfig`(8)

HISTORY

The **url** device driver first appeared in NetBSD 1.6.

AUTHORS

The **url** driver was written by Shingo WATANABE ⟨nabe@nabechan.org⟩.

NAME

usb — Universal Serial Bus driver

SYNOPSIS

```
ehci*   at cardbus? function ?
ehci*   at pci? dev ? function ?
ohci*   at cardbus? function ?
ohci*   at pci? dev ? function ?
slhci*  at isa? port ? irq ?
slhci*  at pcmcia? function ?
uhci*   at cardbus? function ?
uhci*   at pci? dev ? function ?
usb*    at ehci? flags X
usb*    at ohci? flags X
usb*    at uhci? flags X
usb*    at slhci? flags X
uhub*   at usb?
uhub*   at uhub? port ? configuration ? interface ? vendor ? product ?
     release ?
XX*     at uhub? port ? configuration ? interface ? vendor ? product ?
     release ?

options USBVERBOSE

#include <dev/usb/usb.h>
#include <dev/usb/usbhid.h>
```

DESCRIPTION

NetBSD provides machine-independent bus support and drivers for USB devices.

The NetBSD **usb** driver has three layers (like scsi(4) and pcmcia(4)): the controller, the bus, and the device layer. The controller attaches to a physical bus (like pci(4)). The USB bus attaches to the controller and the root hub attaches to the bus. Further devices, which may include further hubs, attach to other hubs. The attachment forms the same tree structure as the physical USB device tree. For each USB device there may be additional drivers attached to it.

The **uhub** device controls USB hubs and must always be present since there is at least a root hub in any USB system.

The *flags* argument to the *usb* device affects the order in which the device detection happens during cold boot. Normally, only the USB host controller and the *usb* device are detected during the autoconfiguration when the machine is booted. The rest of the devices are detected once the system becomes functional and the kernel thread for the *usb* device is started. Sometimes it is desirable to have a device detected early in the boot process, e.g., the console keyboard. To achieve this use a *flags* value of 1.

NetBSD supports the following machine-independent USB drivers:

Storage devices

umass(4) USB Mass Storage Devices, e.g., external disk drives

Wired network interfaces

aue(4) ADMtek AN986/ADM8511 Pegasus family 10/100 USB Ethernet device

axe(4)	ASIX Electronics AX88172/AX88178/AX88772 10/100/Gigabit USB Ethernet device
cdce(4)	USB Communication Device Class Ethernet device
cue(4)	CATC USB-EL1201A USB Ethernet device
kue(4)	Kawasaki LSI KL5KUSB101B USB Ethernet device
udav(4)	Davicom DM9601 10/100 USB Ethernet device
url(4)	Realtek RTL8150L 10/100 USB Ethernet device

Wireless network interfaces

atu(4)	Atmel AT76C50x IEEE 802.11b wireless network device
ral(4)	Ralink Technology USB IEEE 802.11b/g wireless network device
rum(4)	Ralink Technology USB IEEE 802.11a/b/g wireless network device
ubt(4)	USB Bluetooth dongles
zyd(4)	ZyDAS ZD1211/ZD1211B USB IEEE 802.11b/g wireless network device

Serial and parallel interfaces

ubsa(4)	Belkin USB serial adapter
uchcom(4)	WinChipHead CH341/340 based USB serial adapter
ucom(4)	USB tty support
ucycom(4)	Cypress microcontroller based USB serial adapter
uftdi(4)	FT8U100AX USB serial adapter
ugensa(4)	USB generic serial adapter
uhmodem(4)	USB Huawei 3G wireless modem device
uipaq(4)	iPAQ USB units
ukyopon(4)	USB Kyocera AIR-EDGE PHONE device
ulpt(4)	USB printer support
umct(4)	MCT USB-RS232 USB serial adapter
umodem(4)	USB modem support
uplcom(4)	Prolific PL-2303 USB serial adapter
uslsa(4)	Silicon Laboratories CP2101/CP2102 based USB serial adapter
uvisor(4)	USB Handspring Visor
uvscom(4)	SUNTAC Slipper U VS-10U USB serial adapter

Audio devices

uaudio(4)	USB audio devices
umidi(4)	USB MIDI devices
urio(4)	Diamond Multimedia Rio MP3 players

Radio receiver devices

udsbr(4)	D-Link DSB-R100 USB radio device

Human Interface Devices

uhid(4)	Generic driver for Human Interface Devices
uhidev(4)	Base driver for all Human Interface Devices
ukbd(4)	USB keyboards that follow the boot protocol
ums(4)	USB mouse devices

Miscellaneous devices

stuirda(4)	Sigmaltel 4116/4220 USB-IrDA bridge
uep(4)	USB eGalax touch-panel

`ugen`(4)	USB generic devices
`uirda`(4)	USB IrDA bridges
`upl`(4)	Prolific based host-to-host adapters
`uscanner`(4)	USB scanner support
`usscanner`(4)	SCSI-over-USB scanners
`ustir`(4)	SigmaTel STIr4200 USB IrDA bridges
`utoppy`(4)	Topfield TF5000PVR range of digital video recorders
`uyap`(4)	USB YAP phone firmware loader

INTRODUCTION TO USB

The USB 1.x is a 12 Mb/s serial bus with 1.5 Mb/s for low speed devices. USB 2.x handles 480 Mb/s. Each USB has a host controller that is the master of the bus; all other devices on the bus only speak when spoken to.

There can be up to 127 devices (apart from the host controller) on a bus, each with its own address. The addresses are assigned dynamically by the host when each device is attached to the bus.

Within each device there can be up to 16 endpoints. Each endpoint is individually addressed and the addresses are static. Each of these endpoints will communicate in one of four different modes: control, isochronous, bulk, or interrupt. A device always has at least one endpoint. This endpoint has address 0 and is a control endpoint and is used to give commands to and extract basic data, such as descriptors, from the device. Each endpoint, except the control endpoint, is unidirectional.

The endpoints in a device are grouped into interfaces. An interface is a logical unit within a device; e.g., a compound device with both a keyboard and a trackball would present one interface for each. An interface can sometimes be set into different modes, called alternate settings, which affects how it operates. Different alternate settings can have different endpoints within it.

A device may operate in different configurations. Depending on the configuration the device may present different sets of endpoints and interfaces.

Each device located on a hub has several `config`(1) locators:

`port` this is the number of the port on closest upstream hub.

`configuration` this is the configuration the device must be in for this driver to attach. This locator does not set the configuration; it is iterated by the bus enumeration.

`interface` this is the interface number within a device that an interface driver attaches to.

`vendor` this is the 16 bit vendor id of the device.

`product` this is the 16 bit product id of the device.

`release` this is the 16 bit release (revision) number of the device.

The first locator can be used to pin down a particular device according to its physical position in the device tree. The last three locators can be used to pin down a particular device according to what device it actually is.

The bus enumeration of the USB bus proceeds in several steps:

1. Any device specific driver can to attach to the device.
2. If none is found, any device class specific driver can attach.
3. If none is found, all configurations are iterated over. For each configuration all the interface are iterated over and interface drivers can attach. If any interface driver attached in a certain configuration the iteration over configurations is stopped.

4. If still no drivers have been found, the generic USB driver can attach.

USB CONTROLLER INTERFACE

Use the following to get access to the USB specific structures and defines.

```
#include <dev/usb/usb.h>
```

The `/dev/usbN` can be opened and a few operations can be performed on it. The `poll`(2) system call will say that I/O is possible on the controller device when a USB device has been connected or disconnected to the bus.

The following `ioctl`(2) commands are supported on the controller device:

`USB_DEVICEINFO` *`struct usb_device_info`*

This command can be used to retrieve some information about a device on the bus. The *addr* field should be filled before the call and the other fields will be filled by information about the device on that address. Should no such device exist an error is reported.

```
struct usb_device_info {
        uint8_t udi_bus;
        uint8_t udi_addr;
        usb_event_cookie_t udi_cookie;
        char           udi_product[USB_MAX_ENCODED_STRING_LEN];
        char           udi_vendor[USB_MAX_ENCODED_STRING_LEN];
        char           udi_release[8];
        char           udi_serial[USB_MAX_ENCODED_STRING_LEN];
        uint16_t       udi_productNo;
        uint16_t       udi_vendorNo;
        uint16_t       udi_releaseNo;
        uint8_t udi_class;
        uint8_t udi_subclass;
        uint8_t udi_protocol;
        uint8_t udi_config;
        uint8_t udi_speed;
#define USB_SPEED_LOW  1
#define USB_SPEED_FULL 2
#define USB_SPEED_HIGH 3
        int            udi_power;
        int            udi_nports;
        char           udi_devnames[USB_MAX_DEVNAMES][USB_MAX_DEVNAMELEN];
        uint8_t udi_ports[16];
#define USB_PORT_ENABLED 0xff
#define USB_PORT_SUSPENDED 0xfe
#define USB_PORT_POWERED 0xfd
#define USB_PORT_DISABLED 0xfc
};
```

The *product*, *vendor*, *release*, and *serial* fields contain self-explanatory descriptions of the device.

The *class* field contains the device class.

The *config* field shows the current configuration of the device.

The *lowspeed* field is set if the device is a USB low speed device.

The *power* field shows the power consumption in milli-amps drawn at 5 volts, or zero if the device is self powered.

If the device is a hub the *nports* field is non-zero and the *ports* field contains the addresses of the connected devices. If no device is connected to a port one of the *USB_PORT_*∗ values indicates its status.

`USB_DEVICESTATS` *`struct usb_device_stats`*
This command retrieves statistics about the controller.

```
struct usb_device_stats {
        u_long  uds_requests[4];
};
```

The *requests* field is indexed by the transfer kind, i.e. *UE_*∗, and indicates how many transfers of each kind that has been completed by the controller.

`USB_REQUEST` *`struct usb_ctl_request`*
This command can be used to execute arbitrary requests on the control pipe. This is *DANGEROUS* and should be used with great care since it can destroy the bus integrity.

The include file ⟨`dev/usb/usb.h`⟩ contains definitions for the types used by the various `ioctl`(2) calls. The naming convention of the fields for the various USB descriptors exactly follows the naming in the USB specification. Byte sized fields can be accessed directly, but word (16 bit) sized fields must be access by the **`UGETW`**(*`field`*) and **`USETW`**(*`field`*, *`value`*) macros to handle byte order and alignment properly.

The include file ⟨`dev/usb/usbhid.h`⟩ similarly contains the definitions for Human Interface Devices (HID).

USB EVENT INTERFACE

All USB events are reported via the `/dev/usb` device. This devices can be opened for reading and each `read`(2) will yield an event record (if something has happened). The `poll`(2) system call can be used to determine if an event record is available for reading.

The event record has the following definition:

```
struct usb_event {
        int                                 ue_type;
#define USB_EVENT_CTRLR_ATTACH 1
#define USB_EVENT_CTRLR_DETACH 2
#define USB_EVENT_DEVICE_ATTACH 3
#define USB_EVENT_DEVICE_DETACH 4
#define USB_EVENT_DRIVER_ATTACH 5
#define USB_EVENT_DRIVER_DETACH 6
        struct timespec                     ue_time;
        union {
                struct {
                        int                 ue_bus;
                } ue_ctrlr;
                struct usb_device_info      ue_device;
                struct {
                        usb_event_cookie_t  ue_cookie;
                        char                ue_devname[16];
                } ue_driver;
        } u;
};
```

The *ue_type* field identifies the type of event that is described. The possible events are attach/detach of a host controller, a device, or a device driver. The union contains information pertinent to the different types of events.
The *ue_bus* contains the number of the USB bus for host controller events.
The *ue_device* record contains information about the device in a device event event.
The *ue_cookie* is an opaque value that uniquely determines which device a device driver has been attached to (i.e., it equals the cookie value in the device that the driver attached to). The *ue_devname* contains the name of the device (driver) as seen in, e.g., kernel messages.

Note that there is a separation between device and device driver events. A device event is generated when a physical USB device is attached or detached. A single USB device may have zero, one, or many device drivers associated with it.

KERNEL THREADS

For each USB bus, i.e., for each host controller, there is a kernel thread that handles attach and detach of devices on that bus. The thread is named *usbN* where *N* is the bus number.

In addition there is a kernel thread, *usbtask*, which handles various minor tasks that are initiated from an interrupt context, but need to sleep, e.g., time-out abort of transfers.

SEE ALSO

The USB specifications can be found at:
http://www.usb.org/developers/docs/

`usbhidaction`(1), `usbhidctl`(1), `cardbus`(4), `ehci`(4), `isa`(4), `ohci`(4), `pci`(4), `pcmcia`(4), `slhci`(4), `uhci`(4), `usbdevs`(8)

HISTORY

The **`usb`** driver appeared in NetBSD 1.4.

BUGS

There should be a serial number locator, but NetBSD does not have string valued locators.

NAME

uscanner — minimal USB support for scanners

SYNOPSIS

uscanner* at uhub?

DESCRIPTION

The **uscanner** driver provides minimal support for USB scanners. The driver recognizes a number of USB scanners, but to actually scan anything there needs to be software that knows about the particular scanner. The SANE package provides support for some scanners.

HARDWARE

The **uscanner** driver works with the following scanners:

Epson

GT-7000
Perfection 610U
Perfection 636U
Perfection 1200U
Perfection 1200U Photo
Perfection 1260
Perfection 1660
Perfection 2400

HP

ScanJet 4100C
ScanJet 5200C
ScanJet 6300C

Many other scanners are recognized, but there is no software support for them yet.

SEE ALSO

scanimage(1), usb(4), usscanner(4)
http://www.buzzard.org.uk/jonathan/scanners-usb.html

HISTORY

The **uscanner** driver appeared in NetBSD 1.6.

BUGS

This driver should not really exist. The scanners it recognizes can be accessed through the ugen(4) driver instead. A proper scanner driver would provide a uniform interface to scanners instead of exposing the innards of the scanner. The reason the driver exists is to have something that is compatible with the Linux driver.

NAME

userconf — in-kernel device configuration manager

SYNOPSIS

options USERCONF

DESCRIPTION

userconf is the in-kernel device configuration manager. It is used to alter the kernel autoconfiguration framework at runtime. **userconf** is activated from the boot loader by passing the **-c** option to the kernel.

COMMAND SYNTAX

The general command syntax is:

command [*option*]

userconf has a more(1)-like functionality; if a number of lines in a command's output exceeds the number defined in the lines variable, then **userconf** displays "-- more --" and waits for a response, which may be one of:

<return> one more line.

<space> one more page.

q abort the current command, and return to the command input mode.

COMMANDS

userconf supports the following commands:

lines *count*
Specify the number of lines before more.

base *8* | *10* | *16*
Base for displaying large numbers.

change *devno* | *dev*
Change devices.

disable *devno* | *dev*
Disable devices.

enable *devno* | *dev*
Enable devices.

exit A synonym for **quit**.

find *devno* | *dev*
Find devices.

help Display online help.

list List current configuration.

quit Leave userconf.

? A synonym for **help**.

HISTORY

The **userconf** framework first appeared in OpenBSD 2.0, and was then integrated into NetBSD 1.6.

AUTHORS

The **userconf** framework was written by Mats O Jansson ⟨moj@stacken.kth.se⟩.

NAME

uslsa — USB support for Silicon Labs CP210x series serial adapters

SYNOPSIS

```
uslsa* at uhub?
ucom*  at uslsa?
```

HARDWARE

The **uslsa** driver is known to work with the following adapters:

Siemens MC60 Data Cable
Suunto USB Serial Adaptor
Helicomm IP-Link 1220-DVM
Nokia CA-42 USB

DESCRIPTION

The **uslsa** driver provides support for the Silicon Labs USB-to-RS-232 Bridge chip.

The device is accessed through the ucom(4) driver which makes it behave like a tty(4).

SEE ALSO

tty(4), ucom(4), usb(4)

http://www.silabs.com.

HISTORY

The **uslsa** driver appeared in NetBSD 4.0.

AUTHORS

The **uslsa** driver was written by Jonathan A. Kollasch. Code and style was borrowed from existing NetBSD USB-serial drivers. Due to the unavailability of documentation, Craig Shelley's reverse-engineered Linux driver was used as documentation.

CAVEATS

Hardware flow control may not work correctly.

Settings other than 8 data bits, no parity, and 1 stop bit seem to be refused by the chip.

NAME

usscanner — driver for some SCSI-over-USB scanners

SYNOPSIS

```
usscanner* at uhub?
scsibus*   at usscanner?
ss*        at scsibus?
```

DESCRIPTION

The **usscanner** provides support for some scanners based on non-standard SCSI-over-USB, e.g., HP5300C.

HARDWARE

The **usscanner** driver works with the following scanners:

HP

HP ScanJet 5300C

SEE ALSO

scsi(4), usb(4), uscanner(4), pkgsrc/graphics/sane-backends

HISTORY

The **usscanner** driver appeared in NetBSD 1.6.

NAME

ustir — SigmaTel STIr4200-based USB-IrDA bridge support

SYNOPSIS

```
ustir*   at uhub?
irframe* at ustir?
```

DESCRIPTION

The **ustir** driver provides support for USB-IrDA bridges based on the SigmaTel STIr4200. The driver is known to work with the following adapters:

Mars II 740 USB IrDA Adapter

Access to the IrDA device is through the irframe(4) driver.

SEE ALSO

irframe(4), uirda(4), usb(4)

HISTORY

The **ustir** driver appeared in NetBSD 1.6.

AUTHORS

The **ustir** driver was written by David Sainty ⟨David.Sainty@dtsp.co.nz⟩.

BUGS

The STIr4200 cannot notify the driver when it has received data, instead it has to be continuously polled. This driver polls (when idle) at a fairly low rate of 10 times per second, which means that system performance is not overly affected by rapid polling, but latency is fairly high.

NAME

ut — UNIBUS TU45 tri-density tape drive interface

SYNOPSIS

```
ut0 at uba0 csr 0172440 vector utintr
tj0 at ut0 drive 0
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **ut** interface provides access to a standard tape drive interface as describe in mtio(4). Hardware implementing this on the VAX is typified by the System Industries SI 9700 tape subsystem. Tapes may be read or written at 800, 1600, and 6250 BPI.

DIAGNOSTICS

tj%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

tj%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

tj%d: can't change density in mid-tape. An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.

ut%d: soft error bn%d cs1=%b er=%b cs2=%b ds=%b. The formatter indicated a corrected error at a density other than 800bpi. The data transferred is assumed to be correct.

ut%d: hard error bn%d cs1=%b er=%b cs2=%b ds=%b. A tape error occurred at block

bn. Any error is fatal on non-raw tape; when possible the driver will have retried the operation which failed several times before reporting the error.

tj%d: lost interrupt. A tape operation did not complete within a reasonable time, most likely because the tape was taken off-line during rewind or lost vacuum. The controller should, but does not, give an interrupt in these cases. The device will be made available again after this message, but any current open reference to the device will return an error as the operation in progress aborts.

SEE ALSO

mt(1), mtio(4)

HISTORY

The **ut** driver appeared in 4.2BSD.

BUGS

May hang if a physical error (non-data) occurs.

NAME

utoppy — USB driver for the Topfield TF5000PVR range of digital video recorders

SYNOPSIS

```
utoppy* at uhub? port ?

#include <dev/usb/utoppy.h>
```

DESCRIPTION

The **utoppy** driver provides support for the Topfield TF5000PVR range of DVB recorders (nicknamed Toppy) which are popular in Europe and Australia. These recorders have a USB device interface which can be used to transfer recordings to and from the unit's hard disk. The USB interface can also be used to upload binary images for execution on the Toppy's MIPS cpu.

The Toppy's USB protocol has not been officially documented by Topfield, but the basic features have been reverse engineered by others in order to write replacements for the official Altair download/upload program from Topfield.

Existing replacements for Altair suffer from the fact that they are ultimately built on top of ugen(4). This has a number of detrimental side-effects:

1. Performance suffers since all Toppy command packets have to cross the user-kernel interface.
2. The userland programs are full of clutter to deal with interpreting the command/status packets, not to mention byte-swapping and host endian issues.
3. Signals can interrupt a data transfer at a critical point, leaving the Toppy in an undefined state. For example, interrupting a download with Turbo mode enabled will leave the Toppy completely unresponsive to the remote control, and prevent timer-based recordings from starting.

The **utoppy** driver provides a clean and stable interface to the Toppy protocol, and ensures that an interrupt caused by a signal does not leave the Toppy in an undefined state.

UTOPPY INTERFACE

Use the following header file to get access to the utoppy specific structures and defines.

```
#include <dev/usb/utoppy.h>
```

The **utoppy** driver can be accessed through the /dev/utoppyN character device. The primary means of controlling the driver is by issuing a series of ioctl(2) system calls followed by read(2) or write(2) system calls as appropriate.

The following ioctl(2) commands are supported by the **utoppy** driver:

UTOPPYIOTURBO *int *mode*

> This command can be used to enable or disable Turbo mode for subsequent UTOPPYIOREADFILE or UTOPPYIOWRITEFILE commands (see below). If *num* is non-zero, Turbo mode will be enabled. Otherwise Turbo mode will be disabled. In non-Turbo mode, the Toppy's USB interface is capable of sustaining around 5.6 Mbit/s during a file transfer. With Turbo mode enabled, it can sustain just over 16 Mbit/s. Of course, these figures are valid only if the Toppy is connected via a USB 2.0 interface. Performance using an older USB 1 interface will be significantly lower.

UTOPPYIOCANCEL *void*

> This command can be used to abort an in-progress UTOPPYIOREADDIR, UTOPPYIOREADFILE, or UTOPPYIOWRITEFILE command.

UTOPPYIOREBOOT *void*
This command will cause the Toppy to reboot cleanly.

UTOPPYIOSTATS *struct utoppy_stats *stats*
This command retrieves statistics for the Toppy's hard disk.

```
struct utoppy_stats {
        uint64_t us_hdd_size;  /* Size of the disk, in bytes */
        uint64_t us_hdd_free;  /* Free space, in bytes */
};
```

UTOPPYIORENAME *struct utoppy_rename *rename*
This command is used to rename a file or directory on the Toppy's hard disk. The full pathname to each file must be provided.

```
struct utoppy_rename {
        char *ur_old_path;     /* Path to existing file */
        char *ur_new_path;     /* Path to new file */
};
```

UTOPPYIOMKDIR *char *path*
This command creates the directory specified by *path*.

UTOPPYIODELETE *char *path*
This command deletes the file or directory specified by *path*.

UTOPPYIOREADDIR *char *path*
This command initiates a read of the contents of the directory specified by *path*. After issuing this command, the directory contents must be read using consecutive read(2) system calls. Each read(2) will transfer one or more directory entries into the user-supplied buffer. The buffer must be large enough to receive at least one directory entry. When read(2) returns zero, all directory entries have been read.

A directory entry is described using the following data structure:

```
struct utoppy_dirent {
        char ud_path[UTOPPY_MAX_FILENAME_LEN + 1];
        enum {
                UTOPPY_DIRENT_UNKNOWN,
                UTOPPY_DIRENT_DIRECTORY,
                UTOPPY_DIRENT_FILE
        } ud_type;
        off_t ud_size;
        time_t ud_mtime;
        uint32_t ud_attributes;
};
```

The *ud_path* field contains the name of the directory entry.

The *ud_type* field specifies whether the entry corresponds to a file or a sub-directory.

The *ud_size* field is valid for files only, and specifies the file's size in bytes.

The *ud_mtime* field describes the file or directory's modification time, specified as seconds from the Unix epoch. The timestamp is relative to the current timezone, so localtime(3) can be used to convert it into human readable form. Note that the Toppy sets directory timestamps to a predefined value so they are not particularly useful.

The *ud_attributes* field is not used at this time.

UTOPPYIOREADFILE *struct utoppy_readfile ∗*
This command is used to initiate reading a file from the Toppy's hard disk. The full pathname, together with the file offset at which to start reading, is specified using the following data structure:

```
struct utoppy_readfile {
        char ∗ur_path;
        off_t ur_offset;
};
```

After issuing this command, the file must be read using consecutive `read`(2) system calls. When `read`(2) returns zero, the entire file has been read.

UTOPPYIOWRITEFILE *struct utoppy_writefile ∗*
This command is used to initiate writing to a file on the Toppy's hard disk. The file to be written is described using the following data structure:

```
struct utoppy_writefile {
        char ∗uw_path;
        off_t uw_offset;
        off_t uw_size;
        time_t uw_mtime;
};
```

The *uw_path* field specifies the full pathname of the file to be written.

The *uw_offset* field specifies the file offset at which to start writing, assuming the file already exists. Otherwise, *uw_offset* must be zero.

The protocol requires that the Toppy must be informed of a file's size in advance of the file being written. This is accomplished using the *uw_size* field. It may be possible to work around this limitation in a future version of the driver.

The *uw_mtime* field specifies the file's timestamp expressed as seconds from the Unix epoch in the local timezone.

Due to limitations with the protocol, a **utoppy** device can be opened by only one application at a time. Also, only a single `UTOPPYIOREADDIR`, `UTOPPYIOREADFILE`, or `UTOPPYIOWRITEFILE` command can be in progress at any given time.

FILES

`/dev/utoppy0` device node

SEE ALSO

`utoppya`(1), `usb`(4)

HISTORY

The **utoppy** driver appeared in NetBSD 4.0.

AUTHORS

Steve C. Woodford ⟨scw@netbsd.org⟩

NAME

uu — TU-58/DECtape II UNIBUS cassette interface

SYNOPSIS

```
options UUDMA
uu0 at uba0 csr 0176500 vector uurintr uuxintr
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **uu** device provides access to dual DEC TU-58 tape cartridge drives connected to the UNIBUS via a DL-11W interface module.

The interface supports only block I/O to the TU-58 cassettes (see `physio`(4)). The drives are normally manipulated with the `arff`(8) program using the ‘‘m’’ and ‘‘f’’ options.

The driver provides for an optional write and verify (read after write) mode that is activated by specifying the ‘‘a’’ device.

The TU-58 is treated as a single device by the system even though it has two separate drives, ‘`uu0`’ and ‘`uu1`’. If there is more than one TU-58 unit on a system, the extra drives are named ‘`uu2`’, ‘`uu3`’ etc.

NOTES

Assembly language code to assist the driver in handling the receipt of data (using a pseudo-DMA approach) should be included when using this driver; specify ‘`options UUDMA`’ in the configuration file.

FILES

`/dev/uu?`
`/dev/uu?a`

DIAGNOSTICS

uu%d: no bp, active %d. A transmission complete interrupt was received with no outstanding I/O request. This indicates a hardware problem.

uu%d protocol error, state=%s, op=%x, cnt=%d, block=%d. The driver entered an illegal state. The information printed indicates the illegal state, the operation currently being executed, the I/O count, and the block number on the cassette.

uu%d: break received, transfer restarted. The TU-58 was sending a continuous break signal and had to be reset. This may indicate a hardware problem, but the driver will attempt to recover from the error.

uu%d receive state error, state=%s, byte=%x. The driver entered an illegal state in the receiver finite state machine. The state is shown along with the control byte of the received packet.

uu%d: read stalled. A timer watching the controller detected no interrupt for an extended period while an operation was outstanding. This usually indicates that one or more receiver interrupts were lost and the transfer is restarted.

uu%d: hard error bn%d, pk_mod %o. The device returned a status code indicating a hard error. The actual error code is shown in octal. No retries are attempted by the driver.

ERRORS

The following errors may be returned:

`[ENXIO]` Nonexistent drive (on open); offset is too large or bad (undefined) `ioctl`(2) code.

[EIO] Open failed, the device could not be reset.

[EBUSY] Drive in use.

SEE ALSO

tu(4), arff(8)

HISTORY

The **uu** driver appeared in 4.2BSD.

NAME

uvisor — USB support for the Handspring Visor, a Palmpilot compatible PDA

SYNOPSIS

```
uvisor* at uhub?
ucom*   at uvisor? portno ?
```

DESCRIPTION

The **uvisor** driver provides support for the Handspring Visor, a Palmpilot compatible PDA.

The device is accessed through the ucom(4) driver which makes it behave like a tty(4). The device has several ports for different purposes, each of them gets its own ucom(4) device. The attach message describes the purpose of each port.

The usual Pilot tools can be used to access the Visor on the HotSync port.

SEE ALSO

tty(4), ucom(4), usb(4)

HISTORY

The **uvisor** driver appeared in NetBSD 1.5.

NAME

uvscom — USB support for SUNTAC Slipper U VS-10U serial adapters driver

SYNOPSIS

```
uvscom* at uhub?
ucom*   at uvscom?
```

DESCRIPTION

The **uvscom** driver provides support for the SUNTAC Slipper U VS-10U chip. Slipper U is a PC Card to USB converter for data communication card adapters. It supports DDI Pocket's Air H" C@rd, C@rd H" 64, NTT's P-in, P-in m@ster, and various other data communication card adapters.

The device is accessed through the ucom(4) driver which makes it behave like a tty(4).

SEE ALSO

tty(4), ucom(4), usb(4)

HISTORY

The **uvscom** driver first appeared in FreeBSD and later in NetBSD 1.6.

NAME

uyap — USB YAP phone firmware loader

SYNOPSIS

uyap* **at uhub?**

DESCRIPTION

The **uyap** driver downloads firmware into a YAP phone. The device will then disconnect and reappear as a composite device with two audio pipes (handled by uaudio(4)) and a HID device (handled by uhid(4)).

SEE ALSO

uaudio(4), uhid(4), usb(4)

HISTORY

The **uyap** driver appeared in NetBSD 1.6.

NAME

va — Benson-Varian printer/plotter interface

SYNOPSIS

```
va0 at uba0 csr 0164000 vector vaintr
vz0 at va0 drive 0
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

(NOTE: the configuration description, while counter-intuitive, is actually as shown above.)

The Benson-Varian printer/plotter in normally used with the line printer system. This description is designed for those who wish to drive the Benson-Varian directly.

In print mode, the Benson-Varian uses a modified ASCII character set. Most control characters print various non-ASCII graphics such as daggers, sigmas, copyright symbols, etc. Only LF and FF are used as format effectors. LF acts as a newline, advancing to the beginning of the next line, and FF advances to the top of the next page.

In plot mode, the Benson-Varian prints one raster line at a time. An entire raster line of bits (2112 bits = 264 bytes) is sent, and then the Benson-Varian advances to the next raster line.

Note: The Benson-Varian must be sent an even number of bytes. If an odd number is sent, the last byte will be lost. Nulls can be used in print mode to pad to an even number of bytes.

To use the Benson-Varian yourself, you must realize that you cannot open the device, `/dev/va0` if there is an daemon active. You can see if there is an active daemon by doing a `lpq`(1) and seeing if there are any files being printed. Printing should be turned off using `lpc`(8).

To set the Benson-Varian into plot mode include the file ⟨`sys/vcmd.h`⟩ and use the following `ioctl`(2) call

```
ioctl(fileno(va), VSETSTATE, plotmd);
```

where *`plotmd`* is defined to be

```
int plotmd[] = { VPLOT, 0, 0 };
```

and *`va`* is the result of a call to `fopen`(3) on stdio. When you finish using the Benson-Varian in plot mode you should advance to a new page by sending it a FF after putting it back into print mode, i.e. by

```
int prtmd[] = { VPRINT, 0, 0 };
...
fflush(va);
ioctl(fileno(va), VSETSTATE, prtmd);
write(fileno(va), "\f\0", 2);
```

FILES

`/dev/va0`

DIAGNOSTICS

The following error numbers are significant at the time the device is opened.

`[ENXIO]` The device is already in use.

`[EIO]` The device is offline.

The following message may be printed on the console.

va%d: npr timeout. The device was not able to get data from the UNIBUS within the timeout period, most likely because some other device was hogging the bus. (But see **BUGS** below).

SEE ALSO

`lpr`(1), `vp`(4), `lpd`(8)

HISTORY

The **va** driver appeared in 4.0BSD.

BUGS

The 1's (one's) and l's (lower-case el's) in the Benson-Varian's standard character set look very similar; caution is advised.

The interface hardware is rumored to have problems which can play havoc with the UNIBUS. We have intermittent minor problems on the UNIBUS where our `va`(4) lives, but haven't ever been able to pin them down completely.

NAME

vald — Toshiba Programmable I/O controller

SYNOPSIS

```
vald* at acpi?
```

DESCRIPTION

Some Toshiba computers have a special I/O controller that handles various interface devices. This special I/O controller is accessed by the "GHCI" ACPI method. The **vald** driver provides some support for it.

The **vald** driver handles the following hot-keys:

Fn+F5	Switch between LCD and External Video output.
Fn+F6	Decrease LCD brightness.
Fn+F7	Increase LCD brightness.
Fn+F8	Switch fan (ON/OFF).

The **vald** driver has only been tested on the Libretto L3 and on the Portege 3440.

SEE ALSO

acpi(4)

HISTORY

The **vald** driver appeared in NetBSD 1.6.

BUGS

vald may have problems with X11 when switching between LCD and External Video output.

NAME

veriexec — Veriexec pseudo-device

SYNOPSIS

```
pseudo-device veriexec
```

DESCRIPTION

Veriexec verifies the integrity of specified executables and files before they are run or read. This makes it much more difficult to insert a trojan horse into the system and also makes it more difficult to run binaries that are not supposed to be running, for example, packet sniffers, DDoS clients and so on.

The **veriexec** pseudo-device is used to load and delete entries to and from the in-kernel *Veriexec* databases, as well as query information about them. It can also be used to dump the entire database.

Kernel-userland interaction

Veriexec uses `proplib`(3) for communication between the kernel and userland.

`VERIEXEC_LOAD`

Load an entry for a file to be monitored by *Veriexec*.

The dictionary passed contains the following elements:

Name	Type	Purpose
file	string	filename for this entry
entry-type	uint8	entry type (see below)
fp-type	string	fingerprint hashing algorithm
fp	data	the fingerprint

"entry-type" can be one or more (binary-OR'd) of the following:

Type	Effect
`VERIEXEC_DIRECT`	can execute directly
`VERIEXEC_INDIRECT`	can execute indirectly (interpreter, `mmap`(2))
`VERIEXEC_FILE`	can be opened
`VERIEXEC_UNTRUSTED`	located on untrusted storage

`VERIEXEC_DELETE`

Removes either an entry for a single file or entries for an entire mount from *Veriexec*.

The dictionary passed contains the following elements:

Name	Type	Purpose
file	string	filename or mount-point

`VERIEXEC_DUMP`

Dump the *Veriexec* monitored files database from the kernel.

Only files that the filename is kept for them will be dumped. The returned array contains dictionaries with the following elements:

Name	Type	Purpose
file	string	filename
fp-type	string	fingerprint hashing algorithm
fp	data	the fingerprint
entry-type	uint8	entry type (see above)

`VERIEXEC_FLUSH`
Flush the *Veriexec* database, removing all entries.

This command has no parameters.

`VERIEXEC_QUERY`
Queries *Veriexec* about a file, returning information that may be useful about it.

The dictionary passed contains the following elements:

Name	Type	Purpose
file	string	filename

The dictionary returned contains the following elements:

Name	Type	Purpose
entry-type	uint8	entry type (see above)
status	uint8	entry status
fp-type	string	fingerprint hashing algorithm
fp	data	the fingerprint

“status” can be one of the following:

Status	Meaning
`FINGERPRINT_NOTEVAL`	`not evaluated`
`FINGERPRINT_VALID`	`fingerprint match`
`FINGERPRINT_MISMATCH`	`fingerprint mismatch`

Note that the requests `VERIEXEC_LOAD`, `VERIEXEC_DELETE`, and `VERIEXEC_FLUSH` are not permitted once the strict level has been raised past 0.

SEE ALSO

`proplib`(3), `sysctl`(3), `security`(8), `sysctl`(8), `veriexecctl`(8), `veriexecgen`(8), `veriexec`(9)

NOTES

`veriexec` is part of the default configuration on the following architectures: amd64, i386, prep, sparc64.

AUTHORS

Brett Lymn ⟨blymn@NetBSD.org⟩
Elad Efrat ⟨elad@NetBSD.org⟩

NAME

vesafb — VESA framebuffer device driver for wscons

SYNOPSIS

```
vesabios* at vesabiosbus?
vesafb* at vesabios?
wsdisplay* at vesafb? console ?

options KVM86
options VESAFB_WIDTH=640
options VESAFB_HEIGHT=480
options VESAFB_DEPTH=8
options VESAFB_PM
```

DESCRIPTION

vesafb provides support for framebuffer consoles on i386 display adapters which implement VESA BIOS extensions (VBE) 2.0 or above. It is used within the wscons(4) console framework. And its functions are available via the internal wsdisplay(4) interface.

Since this interface is accessed via x86 BIOS calls, this driver only works on the i386 port.

Supported kernel options:

options KVM86
Required for vesabios. This enables the framework for running BIOS code in a virtual 8086 machine.

options VESAFB_WIDTH=XXX
Sets the width, such as 640 or 1024. Default is 640.

options VESAFB_HEIGHT=XXX
Sets the height, such as 480 or 768. Default is 480.

options VESAFB_DEPTH=XX
Sets the bits per pixel, such as 8 or 16. Default is 8.

options VESAFB_PM
Enables the power management support by enabling the WSDISPLAYIO_SVIDEO and WSDISPLAYIO_GVIDEO ioctl implementations. This is not enabled by default, as in some cases it might cause the system to hang. Tested on: ATI Radeon M9 (failed), NeoMagic 256AV (passed), VIA VT8623 (failed).

SEE ALSO

wscons(4), wsdisplay(4)

HISTORY

The **vesafb** driver appeared in NetBSD 4.0.

BUGS

The **vesafb** driver attaches to the console very early in the i386 boot process; since **consinit**() is called before KVM86 is initialized, the **vesafb_cnattach**() function assumes that hardware is present. This causes the console to be blank until the **vesafb** driver attaches.

vesafb does not support VBE 1.2 and below, as they do not provide support for linear framebuffers.

options VESAFB_PM may cause some systems to hang.

NAME

vga — VGA graphics driver for wscons

SYNOPSIS

```
options VGA_CONSOLE_SCREENTYPE="??x??"
options VGA_CONSOLE_ATI_BROKEN_FONTSEL

vga0 at isa?
vga* at pci?
wsdisplay* at vga? console ?
```

DESCRIPTION

This driver handles VGA graphics hardware within the `wscons`(4) console framework. It doesn't provide direct device driver entry points but makes its functions available via the internal `wsdisplay`(4) interface.

The **vga** driver supports text-mode hardware acceleration on the VGA hardware. Currently, the driver runs the display with a 720×400 pixel resolution. The VGA text-mode accelerator divides the display into fixed-size character cells. The size of the character cells specifies the number of characters available on the screen and the resolution of the font. The wsdisplay screen "types" supported by the **vga** driver are described by the number of character cells available on the screen. See below for a complete list of supported screen modes in the **vga** driver.

Each screen mode requires a suitable font to be loaded into the kernel by the `wsfontload`(8) utility, before the screen can be used. The size of the font and the screen mode must match for use on the 720×400 display. For example, a screen mode with 80 columns and 40 rows requires a font where each character is 8 pixels wide and 10 pixels high. The **vga** driver can display fonts of the original IBM type and ISO-8859-1 encoded fonts. A builtin font of 256 characters and 8×16 pixels is always present on the VGA hardware.

The colour VGA hardware supports the display of 16 different colours at the same time. It is possible with VGA colour systems to use fonts with 512 characters at any one time. This is due to the fact that with VGA adapters one can specify an alternate font to be used instead of bright letters (used for highlighting on the screen). As an experimental feature, the "higher half" fonts of the former NetBSD/i386 **pcvt** driver distribution can be used too if the kernel option "WSCONS_SUPPORT_PCVTFONTS" was set at compile time. This is only useful with the "*bf" screen types; a font containing the ASCII range of characters must be available too on this screen.

Currently, the following screen types are supported:

80x25 This is the standard VGA text mode with 80 columns and 25 rows. Sixteen different colors can be displayed at the same time. Characters are 8×16 pixels, and a font consists of 256 characters.

80x25bf is a modified version of the previous. It only allows 8 colors to be displayed. In exchange, it can access two fonts at the same time, so that 512 different characters can be displayed.

80x40 A text mode with 80 columns and 40 rows. Similar to the standard mode, 16 colors and 256 characters are available. Characters are 8×10 pixels. For this mode to be useful, a font of that character size must be downloaded.

80x40bf is analogously to "80x25bf" a version with 512 displayable characters but 8 colors only.

80x50 A text mode with 80 columns and 50 rows. Similar to the standard mode, 16 colors and 256 characters are available. Characters are 8×8 pixels. For this mode to be useful, a font of that character size must be downloaded.

80x50bf is analogously to "80x25bf" a version with 512 displayable characters but 8 colors only.

80x24 is a variant of the “80x25” screen type which displays 24 lines only. It uses the standard 8x16 VGA font. This mode might be useful for applications which depend on closer DEC VT100 compatibility.

80x24bf Analogously, like “80x24” but with 512 character slots and 8 colors.

If you have an Ati videocard and you are experiencing problems with fonts other than 80x25, you can try to set **options VGA_CONSOLE_ATI_BROKEN_FONTSEL** in you kernel configuration and see if it helps.

The **vga** driver supports multiple virtual screens on one physical display. The screens allocated on one display can be of different “types”. The type is determined at the time the virtual screen is created and can’t be changed later. Screens are either created at kernel startup (then the default type is used) or later with help of the wsconscfg(8) utility.

SEE ALSO

isa(4), pcdisplay(4), pci(4), wscons(4), wsconscfg(8), wsfontload(8)

BUGS

Only a subset of the possible text modes is supported.

VGA cards are supposed to emulate an MDA if a monochrome display is connected. In this case, the device will naturally not support colors at all, but offer the capability to display underlined characters instead. The “80x25bf”, “80x40bf”, “80x50bf” and “80x24bf” screen types will not be available. This mode of operation has not been tested.

NAME

vge — VIA Networking Technologies VT6122 PCI Gigabit Ethernet adapter driver

SYNOPSIS

vge* **at pci? dev ? function ?**

Configuration of PHYs is also necessary. See mii(4).

DESCRIPTION

The **vge** driver provides support for various NICs and embedded Ethernet interfaces based on the VIA Networking Technologies VT6122 Gigabit Ethernet controller chips.

The VT6122 is a 33/66Mhz 64-bit PCI device which combines a tri-speed MAC with an integrated 10/100/1000 copper PHY. (Some older cards use an external PHY.) The MAC supports TCP/IP hardware checksums (IPv4 only), TCP large send, VLAN tag insertion and stripping, as well as VLAN filtering, a 64-entry CAM filter and a 64-entry VLAN filter, 64-bit multicast hash filter, 4 separate transmit DMA queues, flow control and jumbo frames up to 16K in size. The VT6122 has a 16K receive FIFO and 48K transmit FIFO.

The **vge** driver takes advantage of the VT6122's checksum offload and VLAN tagging features, as well as the jumbo frame and CAM filter support. The CAM filter is used for multicast address filtering to provide 64 perfect multicast address filter support. If it is necessary for the interface to join more than 64 multicast groups, the driver will switch over to using the hash filter.

The jumbo frame support can be enabled by setting the interface MTU to any value larger than the default of 1500 bytes, up to a maximum of 9000 bytes. The receive and transmit checksum offload support can be toggled on and off using the ifconfig(8) utility.

The **vge** driver supports the following media types:

autoselect Enable autoselection of the media type and options. The user can manually override the autoselected mode by adding media options to rc.conf(5).

10baseT/UTP Set 10Mbps operation. The ifconfig(8) **mediaopt** option can also be used to select either **full-duplex** or **half-duplex** modes.

100baseTX Set 100Mbps (Fast Ethernet) operation. The ifconfig(8) **mediaopt** option can also be used to select either **full-duplex** or **half-duplex** modes.

1000baseTX Set 1000baseTX operation over twisted pair. The ifconfig(8) **mediaopt** option can also be used to select either **full-duplex** or **half-duplex** modes.

The **vge** driver supports the following media options:

full-duplex Force full duplex operation.

half-duplex Force half duplex operation.

The **vge** driver also supports one special link option for 1000baseTX cards:

link0 With 1000baseTX cards, establishing a link between two ports requires that one port be configured as a master and the other a slave. With autonegotiation, the master/slave settings will be chosen automatically. However when manually selecting the link state, it is necessary to force one side of the link to be a master and the other a slave. The **vge** driver configures the ports as slaves by default. Setting the **link0** flag with ifconfig(8) will set a port as a master instead.

For more information on configuring this device, see ifconfig(8).

HARDWARE

The **vge** driver supports VIA Networking VT3119 and VT6122 based Gigabit Ethernet adapters including:

- VIA Networking LAN-on-motherboard Gigabit Ethernet
- ZyXEL GN650-T 64-bit PCI Gigabit Ethernet NIC (ZX1701)
- ZyXEL GN670-T 32-bit PCI Gigabit Ethernet NIC (ZX1702)

DIAGNOSTICS

vge%d: couldn't map memory The driver failed to initialize PCI shared memory mapping. This might happen if the card is not in a bus-master slot.

vge%d: unable to map interrupt A fatal initialization error has occurred.

vge%d: watchdog timeout The device has stopped responding to the network, or there is a problem with the network connection (cable). Driver resets the device.

SEE ALSO

arp(4), ciphy(4), mii(4), netintro(4), ukphy(4), vlan(4), ifconfig(8)

HISTORY

The **vge** device driver first appeared in FreeBSD 5.3 and then in NetBSD 3.0.

AUTHORS

The **vge** driver was written by Bill Paul ⟨wpaul@windriver.com⟩. The NetBSD port was done by Jaromir Dolecek ⟨jdolecek@NetBSD.org⟩.

BUGS

VLAN packet filtering is done in software at the moment, though using hardware VLAN tagging.

NAME

viaenv — VIA VT82C686A/VT8231 Hardware Monitor and Power Management Timer

SYNOPSIS

viaenv* **at pci? dev ? function ?**

DESCRIPTION

The **viaenv** is an envsys(4) compatible driver for the Hardware Monitor and the Power Management timer in the VIA VT82C686A and VT8231 South Bridges.

The device has 10 sensors:

Sensor	Units	Typical Use
TSENS1	uK	CPU temperature
TSENS2	uK	System temperature
TSENS3	uK	?
FAN1	RPM	CPU fan
FAN2	RPM	System fan
VSENS1	uV DC	CPU core voltage (2.0V)
VSENS2	uV DC	North Bridge core voltage (2.5V)
Vcore	uV DC	Internal core voltage (3.3V)
VSENS3	uV DC	+5V
VSENS4	uV DC	+12V

Sensor data is updated every 1.5 seconds.

SEE ALSO

envsys(4), envstat(8)

HISTORY

The **viaenv** device appeared in NetBSD 1.5.

BUGS

Interrupt support is unimplemented, as is support for setting values.

NAME

viaide — AMD, NVIDIA and VIA IDE disk controllers driver

SYNOPSIS

```
viaide* at pci? dev ? function ? flags 0x0000
options PCIIDE_AMD756_ENABLEDMA
```

DESCRIPTION

The **viaide** driver supports the following IDE controllers and provides the interface with the hardware for the **ata** driver:

- Advanced Micro Devices AMD-756, 766, 768 and CS5536 IDE Controllers
- NVIDIA nForce, nForce2, nForce2 400, nForce3, nForce3 250, nForce4, MCP04, MCP55, MCP61, MCP65, MCP67 IDE and SATA Controllers.
- VIA Technologies VT82C586, VT82C586A, VT82C596A, VT82C686A, VT8233A, VT8235, VT8237/VT8237R IDE Controllers, VT6421 Serial RAID Controller and CX700 IDE Controller.

The 0x0002 flag forces the **viaide** driver to disable DMA on chipsets for which DMA would normally be enabled. This can be used as a debugging aid, or to work around problems where the IDE controller is wired up to the system incorrectly.

SEE ALSO

ata(4), atapi(4), intro(4), pci(4), pciide(4), wd(4), wdc(4)

NOTES

Drives on the VT6421 Serial RAID Controller can only be accessed after they have been configured into RAID or JBOD sets via its BIOS. It is also inaptly named as it has both SATA and PATA interfaces.

BUGS

The AMD756 chip revision D2 has a bug affecting DMA (but not Ultra-DMA) modes. The workaround documented by AMD is to not use DMA on any drive which does not support Ultra-DMA modes. This does not appear to be necessary on all drives, the PCIIDE_AMD756_ENABLEDMA option can be used to force multiword DMA on the buggy revisions. Multiword DMA can eventually be disabled on a per-drive basis with config flags, see wd(4). The bug, if triggered, will cause a total system hang.

The timings used for the PIO and DMA modes for controllers listed above are for a PCI bus running at 30 or 33 MHz. This driver may not work properly on overclocked systems.

NAME

vidcaudio — VIDC audio device driver

SYNOPSIS

```
vidcaudio0 at vidc0
audio*     at vidcaudio0
```

DESCRIPTION

The **vidcaudio** driver provides support for the audio system of the ARM VIDC20 and of the video and sound macrocell of the ARM7500 and ARM7500FE.

The driver can work in two modes, either using the internal 8-bit DACs of the VIDC20 and its internal reference clock, or using the serial sound output of any supported chip feeding external 16-bit DACs and an external sound clock. On systems with a VIDC20, the choice of mode is based on the `*Configure SoundSystem` setting under RISC OS.

SEE ALSO

`audio`(4)

HISTORY

The **vidcaudio** driver was introduced in NetBSD 1.2. The driver was largely rewritten for NetBSD 2.0 because it had bit-rotted so far as to be useless.

AUTHORS

The first version was written by Melvin Tang-Richardson. The rewrite was by Ben Harris.

NAME

vidcvideo — VIDC video device driver

SYNOPSIS

```
vidcvideo0 at vidc0
wsdisplay* at vidcvideo0 console ?

makeoptions MONITOR="AKF85"
makeoptions MODES="1024,768,60 800,600,60"
```

DESCRIPTION

The **vidcvideo** driver provides support for the video system of the ARM VIDC20 and of the video and sound macrocell of the ARM7500 and ARM7500FE within the wscons(4) framework. It provides support for both a text-based console display and a memory-mapped frame buffer usable by the X Window System.

The **vidcvideo** driver chooses a screen mode to use when NetBSD starts up, and has no means of changing it thereafter. The driver attempts to choose a screen mode from its built-in list that most closely matches the mode being used under RISC OS. If the built-in list of modes is inadequate, a different one can be substituted by compiling a kernel using the **MONITOR** option to refer to a RISC OS ModeInfo file, specifying the list of modes to extract from it in **MODES**.

FILES

/usr/src/sys/arch/acorn32/conf/monitors Directory for ModeInfo files.

SEE ALSO

wsdisplay(4)

HISTORY

Before NetBSD 4.0, use of **MONITOR** and **MODES** was compulsory, as there was no built-in list of screen modes.

NAME

vlan — IEEE 802.1Q Virtual LAN network device

SYNOPSIS

```
pseudo-device vlan
```

DESCRIPTION

The **vlan** interface provides support for IEEE 802.1Q Virtual Local Area Networks (VLAN). This supports the trunking of more than one network on a single network interface. This is particularly useful on routers or on hosts which must be connected to many different networks through a single physical interface.

To use a **vlan** interface, the administrator must first create the interface and then specify the VID (VLAN identifier, the first 12 bits from a 16-bit integer which distinguishes each VLAN from any others) and physical interface associated with the VLAN. This can be done by using the ifconfig(8) **create**, **vlan**, and **vlanif** subcommands from a shell command line or script. From within a C program, use the ioctl(2) system call with the SIOCSIFCREATE and SIOCSIFVLAN arguments.

To be compatible with other IEEE 802.1Q devices, the **vlan** interface supports a 1500 byte MTU, which means that the parent interface will have to handle packets that are 4 bytes larger than the original Ethernet standard. Drivers supporting this increased MTU are:

- drivers using the DP8390 core (such as ec(4), ne(4), we(4), and possibly others)
- bge(4)
- bnx(4)
- ea(4)
- eb(4)
- epic(4)
- ex(4)
- fxp(4)
- gem(4)
- hme(4)
- le(4)
- sip(4)
- ste(4)
- stge(4)
- ti(4)
- tl(4)
- tlp(4)
- vge(4)
- wm(4)
- xi(4)

vlan can be used with devices not supporting the IEEE 802.1Q MTU, but then the MTU of the **vlan** interface will be 4 bytes too small and will not interoperate properly with other IEEE 802.1Q devices, unless the MTU of the other hosts on the VLAN are also lowered to match.

EXAMPLES

The following will create interface **vlan0** with VID six, on the Ethernet interface **tlp0**:

```
ifconfig vlan0 create
ifconfig vlan0 vlan 6 vlanif tlp0
```

After this set up, IP addresses (and/or other protocols) can be assigned to the **vlan0** interface. All other hosts on the Ethernet connected to **tlp0** which configure a VLAN and use VID six will see all traffic transmitted through **vlan0**.

The same VLAN can be created at system startup time by placing the following in `/etc/ifconfig.vlan0`:

```
create
vlan 6 vlanif tlp0
```

SEE ALSO

`ifconfig`(8)

HISTORY

The **`vlan`** device first appeared in NetBSD 1.5.1, and was derived from a VLAN implementation that appeared in FreeBSD and OpenBSD.

BUGS

The **`vlan`** interfaces do not currently inherit changes made to the physical interfaces' MTU.

NAME

vmmon — unknown

SYNOPSIS

unknown

DESCRIPTION

No description.

SEE ALSO

Nothing

BUGS

jdolecek@NetBSD.org has not yet written this man page.

NAME

vmnet — unknown

SYNOPSIS

unknown

DESCRIPTION

No description.

SEE ALSO

Nothing

BUGS

jdolecek@NetBSD.org has not yet written this man page.

NAME

vnd — vnode disk driver

SYNOPSIS

```
pseudo-device vnd
options VND_COMPRESSION
```

DESCRIPTION

The **vnd** driver provides a disk-like interface to a file. This is useful for a variety of applications, including swap files and building miniroot or floppy disk images.

This document assumes that you're familiar with how to generate kernels, how to properly configure disks and pseudo-devices in a kernel configuration file.

In order to compile in support for the vnd, you must add a line similar to the following to your kernel configuration file:

```
pseudo-device     vnd             # vnode disk driver
```

To also compile in support for reading compressed disk images, add the following option to your kernel config file:

```
options     VND_COMPRESSION    # compressed vnd(4)
```

Compressed disk images are expected in the cloop2 format, they can be created from "normal" disk images by the `vndcompress`(1) program.

There is a run-time utility that is used for configuring both compressed and uncompressed **vnd**s. See `vnconfig`(8) for more information.

FILES

`/dev/{,r}vnd*` vnd device special files.

SEE ALSO

`config`(1), `vndcompress`(1), `MAKEDEV`(8), `fsck`(8), `mount`(8), `newfs`(8), `vnconfig`(8)

HISTORY

The vnode disk driver was originally written at the University of Utah. The compression handling is based on code by Cliff Wright ⟨cliff@snipe444.org⟩.

BUGS

The **vnd** driver does not work if the file does not reside in a local filesystem.

NAME

vp — Versatec printer/plotter interface

SYNOPSIS

```
vp0 at uba0 csr 0177510 vector vpintr vpintr
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The Versatec printer/plotter is normally used with the line printer system. This description is designed for those who wish to drive the Versatec directly.

To use the Versatec yourself, you must realize that you cannot open the device, `/dev/vp0` if there is a daemon active. You can see if there is a daemon active by doing a `lpq`(1), and seeing if there are any files being sent. Printing should be turned off using `lpc`(8).

To set the Versatec into plot mode you should include ⟨`sys/vcmd.h`⟩ and use the `ioctl`(2) call

```
ioctl(fileno(vp), VSETSTATE, plotmd);
```

where *plotmd* is defined to be

```
int plotmd[] = { VPLOT, 0, 0 };
```

and *vp* is the result of a call to `fopen`(3) on stdio. When you finish using the Versatec in plot mode you should eject paper by sending it a EOT after putting it back into print mode, i.e. by

```
int prtmd[] = { VPRINT, 0, 0 };
...
fflush(vp);
ioctl(fileno(vp), VSETSTATE, prtmd);
write(fileno(vp), "\04", 1);
```

FILES

`/dev/vp0`

DIAGNOSTICS

The following error numbers are significant at the time the device is opened.

`[ENXIO]` The device is already in use.

`[EIO]` The device is offline.

SEE ALSO

`lpr`(1), `va`(4), `lpd`(8)

HISTORY

A **vp** driver appeared in Version 7 AT&T UNIX.

BUGS

The configuration part of the driver assumes that the device is set up to vector print mode through 0174 and plot mode through 0200. As the configuration program can't be sure which vector interrupted at boot time, we specify that it has two interrupt vectors, and if an interrupt comes through 0200 it is reset to 0174. This is safe for devices with one or two vectors at these two addresses. Other configurations with 2 vectors may require changes in the driver.

NAME

vr — Ethernet driver for VIA Rhine Ethernet boards

SYNOPSIS

```
vr* at pci? dev ? function ?
```

Configuration of PHYs are necessary. See mii(4).

DESCRIPTION

The **vr** device driver supports network adapters based on the VIA VT3043(Rhine), VIA VT86C100A(Rhine-II), and VIA VT6105(Rhine-III) chips.

HARDWARE

Supported cards include:

D-Link DFE530TX

MEDIA SELECTION

Media selection is done using ifconfig(8) using the standard ifmedia(4) mechanism. Refer to those manual pages for more information.

SEE ALSO

ifmedia(4), mii(4), netintro(4), pci(4), ifconfig(8)

NAME

vs — X68K built-in voice synthesizer driver

SYNOPSIS

```
vs0 at intio0 addr 0xe92000 dma 3 dmaintr 106
audio* at vs?
```

DESCRIPTION

vs drives x68k's built-in voice synthesizer. It is implemented as an audio(4) device.

SEE ALSO

audio(4), intio(4)

NAME

vv — Proteon proNET 10 Megabit ring network

SYNOPSIS

```
vv0 at uba0 csr 0161000 vector vvrint vvxint
```

DESCRIPTION

NOTE: This driver has not been ported from 4.4BSD yet.

The **vv** interface provides access to a 10 Mb/s Proteon proNET ring network.

The network address of the interface must be specified with an `SIOCSIFADDR` `ioctl`(2) before data can be transmitted or received. It is only permissible to change the network address while the interface is marked "down".

The host's hardware address is discovered by putting the interface in digital loopback mode (not joining the ring) and sending a broadcast packet from which the hardware address is extracted.

Transmit timeouts are detected through use of a watchdog routine. Lost input interrupts are checked for when packets are sent out.

If the installation is running CTL boards which use the old broadcast address of '`0`' instead of the new address of `0xff`, the define `OLD_BROADCAST` should be specified in the driver.

The driver can use "trailer" encapsulation to minimize copying data on input and output. This may be disabled, on a per-interface basis, by setting the `IFF_NOTRAILERS` flag with an `SIOCSIFFLAGS` `ioctl`(2).

DIAGNOSTICS

vv%d: host %d. The software announces the host address discovered during autoconfiguration.

vv%d: can't initialize. The software was unable to discover the address of this interface, so it deemed "dead" will not be enabled.

vv%d: error vvocsr=%b. The hardware indicated an error on the previous transmission.

vv%d: output timeout. The token timer has fired and the token will be recreated.

vv%d: error vvicsr=%b. The hardware indicated an error in reading a packet off the ring.

vv%d: can't handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

vv%d: vs_olen=%d. The ring output routine has been handed a message with a preposterous length. This results in an immediate *panic: vs_olen*.

SEE ALSO

`inet`(4), `netintro`(4)

HISTORY

The **vv** driver appeared in 4.2BSD.

BUGS

The encapsulation of trailer packets in the 4.2BSD version of this driver was incorrect (the packet type was in VAX byte order). As a result, the trailer encapsulation in this version is not compatible with the 4.2BSD VAX version.

NAME

wax — GSC bus controller and I/O subsystem

SYNOPSIS

```
wax* at mainbus0
wax* at phantomas?
gsc* at wax?
```

DESCRIPTION

The **wax** GSC bus controller is a version of lasi(4) with limited functionality and no additional devices on the crystal. It is mainly used as a GSC bus controller for additional onboard devices (e.g. 2nd RS232 or HIL) and sometimes add-on cards (such as a TR-9000 card for the HP 9000/712 models).

MACHINES

An incomplete list of machines that use the **wax** bus controller:

- 712/* add-on cards
- 715/{64/80/100}[XC]
- 725/100
- 743/*
- 744/*
- 748[i]
- B132L[+], B160L, B180L+
- C100, C110, C132L, C160L, C160, C180, C200, C240, C360
- J200, J210[XC], J280, J282, J2240
- all D-Class machines

SEE ALSO

asp(4), gsc(4), intro(4), io(4), lasi(4), phantomas(4)

Precision I/O Architecture Reference Specification, Hewlett-Packard.

712 I/O Subsystem ERS, Revision 1.1, Dwg No. A-A2263-66510-31, Hewlett-Packard, 12 February 1993.

Volume 46 Number 2, Hewlett-Packard Journal, April 1995.

HISTORY

The **wax** driver appeared in OpenBSD 3.4. It was ported to NetBSD 1.6 by Matthew Fredette.

NAME

wd — WD100x compatible hard disk driver

SYNOPSIS

```
wd* at atabus? drive ? flags 0x0000
wd* at umass?
options WD_SOFTBADSECT
```

DESCRIPTION

The **wd** driver supports hard disks that emulate the Western Digital WD100x. This includes standard MFM, RLL, ESDI, IDE, and EIDE drives.

The flags are used only with controllers that support DMA operations and mode settings (like some pciide controllers). The lowest order nibble (rightmost digit) of the flags defines the PIO mode, the next four bits define the DMA mode and the third nibble defines the UltraDMA mode. For each set of four bits, the 3 lower bits define the mode to use and the last bit must be set to 1 for this setting to be used. For DMA and UDMA, 0xf (1111) means 'disable'. For example, a flags value of 0x0fac (1111 1010 1100) means 'use PIO mode 4, DMA mode 2, disable UltraDMA'. 0x0000 means "use whatever the drive claims to support."

The kernel configuration option "**options WD_SOFTBADSECT**" enables a software managed bad-sector list which will prevent further accesses to sectors where an unrecoverable read error occured. A user interface is provided by `dkctl`(8). Unlike the (historical) mechanisms provided by `bad144`(8) and `badsect`(8) the software list does neither support sector replacement nor is it saved across reboots.

SEE ALSO

`ata`(4), `intro`(4), `pciide`(4), `scsi`(4), `umass`(4), `wdc`(4), `atactl`(8), `dkctl`(8)

BUGS

The optional software bad sector list does not interoperate well with sector remapping features of modern disks. To let the disk remap a sector internally, the software bad sector list must be flushed or disabled before.

NAME

wdc — WD100x compatible hard disk controllers driver

SYNOPSIS

ISA and ISA plug'n'play controllers

```
wdc0 at isa? port 0x1f0 irq 14 flags 0x00
wdc1 at isa? port 0x170 irq 15 flags 0x00
wdc* at isapnp?
```

Open Firmware ISA

```
wdc* at ofisa?
```

PCMCIA controllers

```
wdc* at pcmcia? function ?
```

arm32

```
wdc0 at pioc? offset 0x01f0 irq 9
```

atari

```
wdc0 at mainbus0
```

amiga

```
wdc0 at mainbus0
```

DESCRIPTION

The **wdc** driver provides the interface with the hardware for the ata(4) driver. This driver supports IDE and EIDE controllers, as well as MFM, RLL and ESDI on the ISA bus. PCI IDE controllers in legacy mode are also supported, but the pciide(4) driver may provide more functionality.

On the ISA front-end, the following flags are supported:

- 0x0001 Enables 32-bit I/O negotiation in the driver. This is known to cause problems with some motherboards.
- 0x0002 Don't probe for the second drive.
- 0x0004 Set WDC_CAPABILITY_ATA_NOSTREAM flag.
- 0x0008 Set WDC_CAPABILITY_ATAPI_NOSTREAM flag.

SEE ALSO

ata(4), atapi(4), intro(4), isa(4), isapnp(4), mainbus(4), ofisa(4), pciide(4), pcmcia(4), scsi(4), wd(4)

NAME

wds — WD7000 compatible ISA SCSI driver

SYNOPSIS

```
wds0    at isa? port 0x350 irq 15 drq 6         # WD7000, TMC-7000
wds1    at isa? port 0x358 irq 11 drq 5
scsibus* at wds?
```

DESCRIPTION

The **wds** driver supports the WD7000 family of SCSI adaptors and compatibles, including:

WD7000-ASC	busmastering DMA controller
WD7000-ASE	an ASC with floppy controller and ESDI, manufactured for Apollo workstations.
WD7000-FASST2	an ASC with new firmware and scatter-gather hardware.

SEE ALSO

cd(4), ch(4), intro(4), isa(4), scsi(4), sd(4), st(4)

NAME

wdsc — Western Digital WD33c93 SCSI Bus Interface Controller

SYNOPSIS

```
wdsc0 at pcc? ipl 2
```

DESCRIPTION

The **wdsc** driver provides an abstraction layer between the SCSI hardware fitted to the Motorola MVME147 single board computer (WD33C93), and the machine independent SCSI drivers described in scsibus(4).

In addition to sending the required SCSI commands to target devices on the SCSI bus, the **wdsc** driver deals with DMA, device interrupts, sync/async negotiation and target disconnects/reconnects.

DIAGNOSTICS

Too many to mention.

SEE ALSO

pcc(4), scsibus(4)

BUGS

Negotiation of Synchronous SCSI data transfers has been deliberately disabled in the MVME147's **wdsc** driver as it has proved extremely difficult to make it work reliably in this mode.

The **wdsc** driver does not respond well to exceptional conditions caused by poor SCSI cabling and/or termination. In those instances the machine is likely to require a hard reset to recover.

The current **wdsc** driver should be blown away and replaced with a machine independent version.

NAME

wdsc — Western Digital WD33c93 SCSI Bus Interface Controller

SYNOPSIS

```
wdsc* at hpc? offset ?
```

DESCRIPTION

The **wdsc** driver provides an abstraction layer between the SCSI hardware found in multitudinous SGI machines (including Personal Iris series, Indigo, Indy, Challenge S, Indigo2, and Challenge M) and the machine independent SCSI drivers described in scsibus(4).

In addition to sending the required SCSI commands to target devices on the SCSI bus, the **wdsc** driver deals with DMA, device interrupts, sync/async negotiation, and target disconnects/reconnects.

SEE ALSO

hpc(4), scsibus(4)

HISTORY

Wayne Knowles ported the sgimips incarnation of the **wdsc** driver from the amiga and atari ports. It first appeared in NetBSD 1.6.

NAME

we — Western Digital/SMC WD80x3, SMC Elite Ultra, and SMC EtherEZ Ethernet cards device driver

SYNOPSIS

```
we0 at isa? port 0x280 iomem 0xd0000 irq 9
we1 at isa? port 0x300 iomem 0xcc000 irq 10
we* at mca? slot ?
```

DESCRIPTION

The **we** device driver supports Western Digital/SMC WD80x3, SMC Elite Ultra, and SMC EtherEZ Ethernet cards.

FLAG VALUES

For some clone boards the driver is not able to recognize 16bit or 8bit interfaces correctly. Since this makes a huge difference (see diagnostic section below) you can override this by specifying flags value in the config file:

```
we2 at isa? port 0x300 iomem 0xe0000 irq 15 flags 4
```

The values to add together for flags are:

2 force adapter to be treated as 8bit, even if it probes as a 16bit interface. Improper use of this flag will make the driver fail or send invalid Ethernet packets.

4 force adapter to be treated as 16bit, even if it probes as a 8bit interface. For example the COMPEX ENT/U boards identify as WD8003 compatibles, but are in fact 16bit cards. Using this flag on a board that really is a 8bit board will result in bogus packets being sent.

8 disable the use of double transmit buffers to save space in the on-board RAM for more receive buffers.

Note that all supported MCA cards are 16bit.

MEDIA SELECTION

The ability to select media from software is dependent on the particular model of WD/SMC card. The following models support only manual configuration: WD8003S, WD8003E, and WD8013EBT.

Other WD/SMC 80x3 interfaces support two types of media on a single card. All support the AUI media type. The other media is either BNC or UTP behind a transceiver. Software cannot differentiate between BNC and UTP cards. On some models, the AUI port is always active.

The SMC Elite Ultra and SMC EtherEZ interfaces support three media a single card: AUI, BNC, and UTP. If the transceiver is active, the BNC media is selected. Otherwise, the AUI and UTP ports are both active.

To enable the AUI media, select the *10base5* or *aui* media type with `ifconfig`(8)'s **media** directive. To select the other media (transceiver), select the *10base2* or *bnc* media type.

DIAGNOSTICS

we0: overriding IRQ <n> to <m> The IRQ specified in the kernel configuration file is different from that found in the card's configuration registers. The value in the kernel configuration file is being overridden by the one configured into the card.

we0: can't wildcard IRQ on a <model> The IRQ was wildcarded in the kernel configuration file, and the card is a WD8003S, WD8003E, or WD8013EBT, which do not support software IRQ configuration.

we0: failed to clear shared memory at offset <off> The memory test was unable to clear the interface's shared memory region. This often indicates that the card is configured at a conflicting *iomem* address.

we0: warning - receiver ring buffer overrun The DP8390 Ethernet chip used by this board implements a shared-memory ring-buffer to store incoming packets.

The 16bit boards (8013 series) have 16k of memory as well as fast memory access speed. Typical memory access speed on these boards is about 4MB/second. These boards generally have no problems keeping up with full Ethernet speed and the ring-buffer seldom overfills.

However, the 8bit boards (8003) usually have only 8k bytes of shared memory. This is only enough room for about 4 full-size (1500 byte) packets. This can sometimes be a problem, especially on the original WD8003E, because these boards' shared-memory access speed is quite slow; typically only about 1MB/second. The overhead of this slow memory access, and the fact that there is only room for 4 full-sized packets means that the ring-buffer will occasionally overrun. When this happens, the board must be reset to avoid a lockup problem in early revision 8390's. Resetting the board causes all of the data in the ring-buffer to be lost, requiring it to be retransmitted/received, congesting the board further. Because of this, maximum throughput on these boards is only about 400-600k per second.

This problem is exasperated by NFS because the 8bit boards lack sufficient memory to support the default 8k byte packets that NFS and other protocols use as their default. If these cards must be used with NFS, use the NFS **-r** and **-w** options in `/etc/fstab` to limit NFS's packet size. 4096 byte packets generally work.

SEE ALSO

`ifmedia`(4), `intro`(4), `isa`(4), `mca`(4), `ifconfig`(8)

NAME

wesc — Warp Engine low level SCSI interface

SYNOPSIS

```
wesc0 at zbus0
```

DESCRIPTION

The Amiga architecture uses a common machine independent scsi sub-system provided in the kernel source. The machine independent drivers that use this code access the hardware through a common interface. (see scsibus(4)) This common interface interacts with a machine dependent interface, such as **wesc**, which then handles the hardware specific issues.

The **wesc** interface handles things such as DMA and interrupts as well as actually sending commands, negotiating synchronous or asynchronous transfers and handling disconnect/reconnect of SCSI targets. The hardware that **wesc** uses is based on the NCR53c710 SCSI chip.

HARDWARE

The **wesc** interface supports the following Zorro III expansion cards:

WARP ENGINE MacroSystem Development Warp Engine internal SCSI, manufacturer 2203, product 19

DIAGNOSTICS

wesc%s: abort %s: dstat %02x, sstat0 %02x sbcl %02x The scsi operation %s was aborted due to error. Dstat, sstat and sbcl are registers within the NCR53c710 SCSI chip.

siop id %d reset0 The NCR53c710 SCSI chip has been reset and configure at id %d.

SIOP interrupt: %x sts %x msg %x sbcl %x The NCR53c710 SCSI chip has interrupted unexpectedly.

SIOP: SCSI Gross Error The NCR53c710 SCSI chip has indicated that it is confused.

SIOP: Parity Error The NCR53c710 SCSI chip has indicated that it has detected a parity error on the SCSI bus.

SEE ALSO

scsibus(4)

HISTORY

The **wesc** interface first appeared in NetBSD 1.0

NAME

wi — WaveLAN/IEEE and PRISM-II 802.11 wireless network driver

SYNOPSIS

```
wi* at pcmcia? function ?
wi* at pci? dev ? function ?
```

DESCRIPTION

The **wi** driver provides support for Lucent Technologies WaveLAN/IEEE PCCARD adapters (also known as WaveLAN II cards) and various PCI/MiniPCI/PCCARD adapters which use Intersil PRISM-II and PRISM-2.5 chipsets. Note that while Lucent sells both ISA and PCMCIA WaveLAN/IEEE devices, the ISA product is actually a PCMCIA card in an ISA to PCMCIA bridge adapter. Consequently, the **wi** driver is required for both the ISA and PCMCIA NICs. Also note that some of the PRISM-II adapters works only at 3.3V, hence `cardbus`(4) support is required for those cards to set VCC correctly, even though they are 16-bit cards.

The core of the WaveLAN/IEEE is the Lucent Hermes controller. All host/device interaction is via programmed I/O with the Hermes. The Hermes supports 802.11 and 802.3 frames, power management, BSS, WDS and ad-hoc operation modes. The Silver and the Gold cards of the WaveLAN/IEEE also support WEP. Unlike the other IEEE 802.11 network cards, the WaveLAN Gold cards accept 104 bits key (13 characters) for WEP encryption. The Intersil PRISM-II controller supports WEP as well.

The **wi** driver encapsulates all traffic as 802.11 frames, however it can receive either 802.11 or 802.3 frames. Transmit speed is selectable between 1Mbps fixed, 2Mbps fixed or 2Mbps with auto fallback. For WaveLAN/IEEE Turbo adapters, speeds up to 6Mbps are available. For WaveLAN/IEEE Turbo 11Mbps adapters and PRISM-II adapters, speeds up to 11Mbps are available.

The **wi** driver supports configuration of Lucent cards for special ad-hoc operation. In this mode, the nwid is ignored and stations can communicate among each other without the aid of an access point. Note that this mode is specific to Lucent chips, and not in the IEEE 802.11 specification. Due to changes in the implementation of this special ad-hoc mode, Lucent-based cards with different firmware revisions may not interoperate in this mode. This mode is no longer the default and must be selected using the `ifconfig`(8) (media option “adhoc,flag0”) utility.

Recent versions of Lucent and PRISM-II firmware support IBSS creation. IBSS is the standard IEEE 802.11 ad-hoc mode. In this mode, the nwid should be specified. At least one node must be able to create IBSS. The IBSS mode is enabled by “adhoc” or “ibss” media option. IBSS creation is automatically enabled if supported.

The **wi** driver defaults to infrastructure mode (i.e., using an access point).

Recent versions of PRISM-II firmware support operating as an 802.11 Access Point. In this mode, the Access Point station should set the nwid. This will create a standard 802.11 network, and the Access Point station will show up in an Access Point scan. This mode is enabled using the “hostap” media option.

For more information on configuring this device, see `ifconfig`(8) and `ifmedia`(4).

HARDWARE

Cards supported by the **wi** driver include:

Alvarion BreezeNET
Lucent WaveLAN/IEEE 2.0Mb Bronze
Lucent WaveLAN/IEEE 2.0Mb Silver

Lucent WaveLAN/IEEE Turbo
Lucent WaveLAN/IEEE Turbo 11Mbps
Melco AIR CONNECT WLI-PCM-L11, WLI-PCM-L11G
Melco AIR CONNECT WLI-CF-S11G
Compaq WL100 11Mbps Wireless
Corega Wireless LAN PCC-11, PCCA_11, PCCB_11
DEC/Cabletron RoamAbout 802.11 DS High Rate
D-Link DWL-520 11Mbps PCI Card, Revs. A1,A2,B1,B2
D-Link DWL-520 11Mbps PCI Card, Rev. C1 *not supported*, see `atw`(4)
D-Link DWL-520 11Mbps PCI Card, Rev. D *not supported*, see `rtw`(4)
D-Link DWL-650 11Mbps WLAN Card
D-Link DWL-650 11Mbps WLAN Card, Rev. P *not supported* (Prism 3 chipset)
D-Link DCF-650W CF Card
ELECOM Air@Hawk LD-WL11
ELSA AirLancer MC-11
Ericsson Wireless LAN
Farallon Skyline 11Mbps Wireless
Intel PRO/Wireless 2011 LAN PC Card
ICOM SL-1100
IO-DATA WN-B11/PCM
Intersil PRISM Mini-PCI
Linksys Group, Inc. Instant Wireless Network PC Card, CF Card
Linksys Group, Inc. Instant Wireless Network WMP11 PCI Card
Linksys Group, Inc. Instant Wireless Network WMP11v4 PCI Card *not supported*
NCR WaveLAN/IEEE
NEC Wireless Card CMZ-RT-WP, PK-WL001, PC-WL/11C
PLANEX GeoWave/GW-NS110
Symbol Spectrum24 Wireless Networker PC Card, CF Card
TDK LAK-CD011WL
SMC EZ Connect 11M Wireless CF Card SMC2642W
SMC EliteConnect Wireless Adapter PC Card SMC2531W-B
Z-Com XI-626 PCI Card

The original PRISM-I chipset is supported by the `awi`(4) driver.

DIAGNOSTICS

wi%d: init failed The WaveLAN failed to come ready after an initialization command was issued.

wi%d: failed to allocate %d bytes on NIC The driver was unable to allocate memory for transmit frames in the NIC's on-board RAM.

wi%d: device timeout The WaveLAN failed to generate an interrupt to acknowledge a transmit command.

SEE ALSO

`arp`(4), `ifmedia`(4), `netintro`(4), `pci`(4), `pcmcia`(4), `ifconfig`(8), `wiconfig`(8)

HCF Light programming specification, http://www.wavelan.com.

HISTORY

The **`wi`** device driver first appeared in NetBSD 1.5.

AUTHORS

The **wi** driver was written by Bill Paul ⟨wpaul@ctr.columbia.edu⟩.

BUGS

The execution of `wiconfig`(8) while the interface is down can produce some error messages.

NAME

wm — Intel i8254x Gigabit Ethernet driver

SYNOPSIS

```
wm* at pci? dev ? function ?
```

Configuration of PHYs may also be necessary. See mii(4).

DESCRIPTION

The **wm** device driver supports Gigabit Ethernet interfaces based on the Intel i8254x family of Gigabit Ethernet chips. The interfaces supported by the **wm** driver include:

- Intel i82542 1000BASE-X Ethernet
- Intel i82543GC 1000BASE-X Ethernet
- Intel i82543GC 1000BASE-T Ethernet
- Intel i82544EI 1000BASE-T Ethernet
- Intel i82544EI 1000BASE-X Ethernet
- Intel i82544GC 1000BASE-T Ethernet
- Intel i82544GC (LOM) 1000BASE-T Ethernet
- Intel i82540EM 1000BASE-T Ethernet
- Intel i82540EM (LOM) 1000BASE-T Ethernet
- Intel i82540EP 1000BASE-T Ethernet
- Intel i82541EI 1000BASE-T Ethernet
- Intel i82541EI (Mobile) 1000BASE-T Ethernet
- Intel i82541ER 1000BASE-T Ethernet
- Intel i82541GI 1000BASE-T Ethernet
- Intel i82541PI 1000BASE-T Ethernet
- Intel i82545EM 1000BASE-T Ethernet
- Intel i82545EM 1000BASE-X Ethernet
- Intel i82545GB 1000BASE-T Ethernet
- Intel i82545GB 1000BASE-X Ethernet
- Intel i82546EB 1000BASE-T Ethernet (dual-port)
- Intel i82546EB 1000BASE-X Ethernet (dual-port)
- Intel i82546GB 1000BASE-T Ethernet (dual-port)
- Intel i82546GB 1000BASE-X Ethernet (dual-port)
- Intel i82547EI 1000BASE-T Ethernet (CSA)
- Intel i82547GI 1000BASE-T Ethernet (CSA)
- Intel i82571 1000BASE-T Ethernet (dual-port)

- Intel i82572 1000BASE-T Ethernet
- Intel i82573 1000BASE-T Ethernet
- Intel i82801H (ICH8 LAN) 1000BASE-T Ethernet
- Intel i82801I (ICH9 LAN) 1000BASE-T Ethernet
- Intel i80003 1000BASE-T Ethernet

In addition to Intel's own "PRO/1000" line of Gigabit Ethernet interfaces, these chips also appear on some server systems, processor evaluation boards, and in embedded systems.

The i8254x supports IPv4/TCP/UDP checksumming and TCP segmentation in hardware. The **wm** driver supports these features of the chip. See `ifconfig`(8) for information on how to enable this feature.

SEE ALSO

`arp`(4), `ifmedia`(4), `mii`(4), `netintro`(4), `pci`(4), `ifconfig`(8)

HISTORY

The **wm** driver first appeared in NetBSD 1.6.

AUTHORS

The **wm** driver was written by Jason R. Thorpe ⟨thorpej@wasabisystems.com⟩.

BUGS

The Intel i82545GM and i82546GB controllers with internal SERDES are not currently supported.

NAME

wpi — Intel PRO/Wireless 3945ABG IEEE 802.11a/b/g wireless network driver

SYNOPSIS

wpi* **at pci? dev ? function ?**

DESCRIPTION

The **wpi** driver provides support for Intel PRO/Wireless 3945ABG Mini PCI Express network adapters.

These are the modes the **wpi** driver can operate in:

BSS mode
: Also known as *infrastructure* mode, this is used when associating with an access point, through which all traffic passes. This mode is the default.

monitor mode
: In this mode the driver is able to receive packets without associating with an access point. This disables the internal receive filter and enables the card to capture packets from networks to which it wouldn't normally have access, or to scan for access points.

wpi supports software WEP. Wired Equivalent Privacy (WEP) is the de facto encryption standard for wireless networks. It can be typically configured in one of three modes: no encryption; 40-bit encryption; or 104-bit encryption. Unfortunately, due to serious weaknesses in the WEP protocol it is strongly recommended that it not be used as the sole mechanism to secure wireless communication. WEP is not enabled by default.

CONFIGURATION

The **wpi** driver can be configured at runtime with `ifconfig`(8) using the following parameters:

bssid *bssid*
: Set the desired BSSID.

-bssid
: Unset the desired BSSID. The interface will automatically select a BSSID in this mode, which is the default.

chan *n*
: Set the channel (radio frequency) to be used by the driver based on the given channel ID *n*.

-chan
: Unset the desired channel to be used by the driver. The driver will automatically select a channel in this mode, which is the default.

media *media*
: The **wpi** driver supports the following *media* types:

  autoselect Enable autoselection of the media type and options.

mediaopt *opts*
: The **wpi** driver supports the following media options:

  monitor Select monitor mode.

-mediaopt *opts*
: Disable the specified media options on the driver and return it to the default mode of operation (BSS).

mode *mode*
The **wpi** driver supports the following modes:

11a Force 802.11a operation.
11b Force 802.11b operation.
11g Force 802.11g operation.

nwid *id*
Set the network ID. The *id* can either be any text string up to 32 characters in length, or a series of hexadecimal digits up to 64 digits. An empty *id* string allows the interface to connect to any available access points. By default the **wpi** driver uses an empty string. Note that network ID is synonymous with Extended Service Set ID (ESSID).

nwkey *key*
Enable WEP encryption using the specified *key*. The *key* can either be a string, a series of hexadecimal digits (preceded by '0x'), or a set of keys of the form "n:k1,k2,k3,k4", where 'n' specifies which of the keys will be used for transmitted packets, and the four keys, "k1" through "k4", are configured as WEP keys. If a set of keys is specified, a comma (',') within the key must be escaped with a backslash. Note that if multiple keys are used, their order must be the same within the network. **wpi** is capable of using both 40-bit (5 characters or 10 hexadecimal digits) or 104-bit (13 characters or 26 hexadecimal digits) keys.

-nwkey
Disable WEP encryption. This is the default mode of operation.

FILES

The driver needs at least version 2.14.4 of the following firmware file, which is loaded when an interface is brought up:

```
/usr/pkg/libdata/if_wpi/iwlwifi-3945.ucode
```

This firmware file is not free because Intel refuses to grant distribution rights without contractual obligations. As a result, even though NetBSD includes the driver, the firmware file cannot be included and users have to download this file on their own. The official person to state your views to about this issue is peter.engelbrecht@intel.com at +1 (858) 391 1857.

The firmware can be found at `http://intellinuxwireless.org/` (Microcode), or can be installed using the `pkgsrc/sysutils/wpi-firmware2` package.

EXAMPLES

```
# ifconfig wpi0 nwkey 0x1deadbeef1
```

Return wpi0 to its default settings:

```
# ifconfig wpi0 -bssid -chan media autoselect \
        nwid "" -nwkey
```

Join an existing BSS network, "my_net":

```
# ifconfig wpi0 192.168.1.1 netmask 0xffffff00 nwid my_net
```

DIAGNOSTICS

wpi%d: device timeout A frame dispatched to the hardware for transmission did not complete in time. The driver will reset the hardware. This should not happen.

wpi%d: fatal firmware error For some reason, the firmware crashed. The driver will reset the hardware. This should not happen.

wpi%d: Radio transmitter is off The radio transmitter is off and thus no packet can go out. The driver will reset the hardware. Make sure the laptop radio switch is on.

wpi%d: could not read firmware file For some reason, the driver was unable to read the firmware image from the filesystem. The file might be missing or corrupted.

wpi%d: firmware file too short: %d bytes The firmware image is corrupted and can't be loaded into the adapter.

wpi%d: could not load firmware An attempt to load the firmware into the adapter failed. The driver will reset the hardware.

NOTES

802.11a is not working yet.

On some laptops the radio transmitter button must be pushed twice to get the driver working, or you will get a `wpi%d: fatal firmware error` when the interface will be set to up

SEE ALSO

`arp`(4), `ifmedia`(4), `intro`(4), `netintro`(4), `pci`(4), `ifconfig`(8), `firmload`(9), `pkgsrc/sysutils/wpi-firmware2`

The IPW Web Page, http://damien.bergamini.free.fr/ipw/.

AUTHORS

The **`wpi`** driver was originally written by Damien Bergamini ⟨damien@openbsd.org⟩. NetBSD porting was done by
Jean-Baptiste Campesato ⟨camjelemon@gmail.com⟩.

NAME

wscons — workstation console access

SYNOPSIS

```
options WSEMUL_SUN
options WSEMUL_VT100
options WSEMUL_NO_DUMB
options WSEMUL_DEFAULT="xxx"
options WS_DEFAULT_FG=WSCOL_XXX
options WS_DEFAULT_BG=WSCOL_XXX
options WS_DEFAULT_COLATTR="(WSATTR_XXX | WSATTR_YYY)"
options WS_DEFAULT_MONOATTR="(WSATTR_XXX | WSATTR_YYY)"
options WS_KERNEL_FG=WSCOL_XXX
options WS_KERNEL_BG=WSCOL_XXX
options WS_KERNEL_COLATTR="(WSATTR_XXX | WSATTR_YYY)"
options WS_KERNEL_MONOATTR="(WSATTR_XXX | WSATTR_YYY)"
options WSCOMPAT_USL_SYNCTIMEOUT=nnn
options WSDISPLAY_COMPAT_PCVT
options WSDISPLAY_COMPAT_SYSCONS
options WSDISPLAY_COMPAT_USL
options WSDISPLAY_COMPAT_RAWKBD
options WSKBD_EVENT_AUTOREPEAT
options WSKBD_USONLY

wsdisplay* at ...
wskbd*     at ... mux N
wsmouse*   at ... mux N

pseudo-device wsmux N
```

DESCRIPTION

The **wscons** driver provides support for machine independent access to the console.

wscons is made of a number of cooperating modules, in particular

- hardware support for display adapters, keyboards and mice, see wsdisplay(4), wskbd(4), and wsmouse(4)
- input event multiplexor, see wsmux(4)
- terminal emulation modules (see below), and
- compatibility options to support control operations and other low-level behaviour of existing terminal drivers (see below)

Terminal emulations

wscons does not define its own set of terminal control sequences and special keyboard codes in terms of termcap(5). Instead a “terminal emulation” is assigned to each virtual screen when the screen is created. (See wsconscfg(8).) Different terminal emulations can be active at the same time on one display. The following choices are available:

dumb This minimal terminal support is available unless the kernel option **options WSEMUL_NO_DUMB** was specified at build time. No control sequences are supported besides the ASCII control characters. The cursor is not addressable. Only ASCII keyboard codes will be delivered, cursor and functions keys do not work.

sun The "sun" console emulation is available if **options WSEMUL_SUN** was specified at kernel build time. It supports the control sequences of SUN machine consoles and delivers its keyboard codes for function and keypad keys in use. This emulation is sufficient for full-screen applications.

vt100 is available with the kernel compile option **options WSEMUL_VT100**. It provides the most commonly used functions of DEC VT100 terminals with some extensions introduced by the DEC VT220 and DEC VT320 models. The features of the original VT100 which are not or not completely implemented are:

- VT52 support, 132-column-mode, smooth scroll, light background, keyboard autorepeat control, external printer support, keyboard locking, newline/linefeed switching: Escape sequences related to these features are ignored or answered with standard replies. (DECANM, DECCOLM, DECSCLM, DECSCNM, DECARM, DECPFF, DECPEX, KAM, LNM)
- Function keys are not reprogrammable and fonts can not be downloaded. DECUDK and DECDLD sequences will be ignored.
- Neither C1 control set characters will be recognized nor will 8-bit keyboard codes be delivered.
- The "DEC supplemental graphic" font is approximated by the ISO-latin-1 font, though there are subtle differences.
- The actual rendering quality depends on the underlying graphics hardware driver. Characters might be missing in the available fonts and be substituted by more or less fitting replacements.

  Depending on the keyboard used, not all function keys might be available.

In addition to the plain VT100 functions are supported:

- ANSI colors.
- Some VT220 -like presentation state settings and -reports (DECRSPS), especially tabulator settings.

In most applications, **wscons** will work sufficiently as a VT220 emulator.

The *WSEMUL_DEFAULT* kernel option is used to select one of the described terminal options as the default choice. The default takes effect at kernel startup, i.e. for the operating system console or additional screens allocated through the *WSDISPLAY_DEFAULTSCREENS* option (see wsdisplay(4)), or if no emulation type was passed to the wsconscfg(8) utility.

Compatibility options

these options allow X servers and other programs using low-level console driver functions which were written specifically for other console drivers to run on NetBSD systems. The options are in particular:

WSDISPLAY_COMPAT_USL
Support the protocol for switches between multiple virtual screens on one display as used by most PC-UNIX variants.

WSDISPLAY_COMPAT_RAWKBD
Allows to get raw XT keyboard scancodes from PC keyboards as needed by i386 X servers.

WSDISPLAY_COMPAT_PCVT
Emulates enough of the NetBSD/i386 "pcvt" driver to make X servers work.

WSDISPLAY_COMPAT_SYSCONS
Emulates enough of the FreeBSD "syscons" driver to make X servers work. Useful with FreeBSD binary emulation.

Linux/i386 X servers usually run successfully if the first two options are enabled together with the NetBSD Linux binary emulation.

(To have programs looking for device special files of other console drivers find the **wscons** driver entry points, symlinks are a helpful measure.)

Other options

options WS_DEFAULT_FG=WSCOL_XXX,

options WS_DEFAULT_BG=WSCOL_XXX,

options WS_DEFAULT_COLATTR="(WSATTR_XXX | **WSATTR_YYY)"**
and

options WS_DEFAULT_MONOATTR="(WSATTR_XXX | **WSATTR_YYY)"**
allow to make default console output appear in specific colors and attributes. "WS_DEFAULT_FG" and "WS_DEFAULT_BG" set the foreground / background used on color displays. The "WSCOL_XXX" arguments are colors as defined in `src/sys/dev/wscons/wsdisplayvar.h`. "WS_DEFAULT_COLATTR" and "WS_DEFAULT_MONOATTR" are additional attribute flags used on color or monochrome displays, respectively. The arguments are defined in the same header file. Whether the attributes are supported or not depends on the actually used graphics adapter. These options are ignored by the "dumb" terminal emulation.

options WS_KERNEL_FG=WSCOL_XXX,

options WS_KERNEL_BG=WSCOL_XXX,

options WS_KERNEL_COLATTR="(WSATTR_XXX | **WSATTR_YYY)"**
and

options WS_KERNEL_MONOATTR="(WSATTR_XXX | **WSATTR_YYY)"**
allow to make console output originating from the kernel appear differently than output from user level programs (via `/dev/console` or the specific tty device like `/dev/ttyE0`). Their meaning is the same as their 'WS_DEFAULT_∗' counterparts.

options WSCOMPAT_USL_SYNCTIMEOUT=nnn
The virtual screen switching protocol enabled by "WSDISPLAY_COMPAT_USL" uses a somewhat complex handshake protocol to pass control to user programs such as X servers controlling a virtual screen. In order to prevent a non-responsive application from locking the whole console system, a screen switch will be rolled back after a 5 second timeout if the application does not respond. This option can be used to specify in seconds a different timeout value.

options WSKBD_EVENT_AUTOREPEAT
If set, this option enables auto repeat even in event mode. The auto repeat will generate key down events while the key is pressed.

options WSKBD_USONLY
In order to strip down the space usage of wscons, all keymaps except the US english one can be removed from the kernel with this option, which results in a space gain of about 10kB.

SEE ALSO

`wsdisplay`(4), `wskbd`(4), `wsmouse`(4), `wsmux`(4), `wsconscfg`(8), `wsconsctl`(8), `wsfontload`(8), `wscons`(9)

NAME

wsdisplay — generic display device support in wscons

SYNOPSIS

wsdisplay* **at ega? console ?**
(EGA display on ISA)
wsdisplay* **at vga? console ?**
(VGA display on ISA or PCI)
wsdisplay* **at pcdisplay? console ?**
(generic PC (ISA) display)
wsdisplay* **at vesafb? console ?**
(VESA frame buffer display, i386 only)
wsdisplay* **at tga? console ?**
(DEC TGA display, alpha only)
wsdisplay* **at pfb? console ?**
(PCI framebuffer, bebox only)
wsdisplay0 at ofb? console ?
(Open Firmware framebuffer, macppc only)
wsdisplay* **at nextdisplay? console ?**
(NeXT display)
wsdisplay0 at smg0
(VAXstation small monochrome display)
wsdisplay* **at ... kbdmux N**

options WSDISPLAY_BORDER_COLOR=WSCOL_XXX
options WSDISPLAY_CUSTOM_BORDER
options WSDISPLAY_CUSTOM_OUTPUT
options WSDISPLAY_DEFAULTSCREENS=N
options WSDISPLAY_SCROLLSUPPORT

DESCRIPTION

The **wsdisplay** driver is an abstraction layer for display devices within the wscons(4) framework. It attaches to the hardware specific display device driver and makes it available as a text terminal or graphics interface.

A display device can have the ability to display characters on it (without the help of an X server), either directly by hardware or through software putting pixel data into the display memory. Such displays are called “emulating”, the **wsdisplay** driver will connect a terminal emulation module and provide a tty-like software interface. In contrary, non-emulating displays can only be used by special programs like X servers.

The *console* locator in the configuration line refers to the device’s use as the output part of the operating system console. A device specification containing a positive value here will only match if the device is in use as the system console. (The console device selection in early system startup is not influenced.) This way, the console device can be connected to a known wsdisplay device instance. (Naturally, only “emulating” display devices are usable as console.)

The *kbdmux* locator in the configuration line refers to the wsmux(4) that will be used to get keyboard events. If this locator is -1 no mux will be used.

The logical unit of an independent contents displayed on a display (sometimes referred to as “virtual terminal”) is called a “screen” here. If the underlying device driver supports it, multiple screens can be used on one display. (As of this writing, only the vga(4) and the VAX “smg” display drivers provide this ability.) Screens have different minor device numbers and separate tty instances. One screen possesses the “focus”, this means it is visible and its tty device will get the keyboard input. (In some cases − if no screen is set up or

if a screen was just deleted − it is possible that no focus is present at all.) The focus can be switched by either special keyboard input (typically CTRL-ALT-F*n*) or an ioctl command issued by a user program. Screens are created and deleted through the `/dev/ttyEcfg` control device (preferably using the `wsconscfg`(8) utility). Alternatively, the compile-time option `WSDISPLAY_DEFAULTSCREENS`=*n* will also create (at autoconfiguration time) *n* initial screens of the display driver's default type with the system's default terminal emulator.

Kernel options

The following kernel options are available to configure the behavior of the **`wsdisplay`** driver:

`options WSDISPLAY_BORDER_COLOR=WSCOL_XXX`
Sets the border color at boot time. Possible values are defined in `src/sys/dev/wscons/wsdisplayvar.h`. Defaults to 'WSCOL_BLACK'.

`options WSDISPLAY_CUSTOM_BORDER`
Enables the `WSDISPLAYIO_GBORDER` and `WSDISPLAYIO_SBORDER` ioctls, which allow the customization of the border color from userland (after boot). See `wsconsctl`(8).

`options WSDISPLAY_CUSTOM_OUTPUT`
Enables the `WSDISPLAYIO_GMSGATTRS` and `WSDISPLAYIO_SMSGATTRS` ioctls, which allow the customization of the console output and kernel messages from userland (after boot). See `wsconsctl`(8).

`options WSDISPLAY_DEFAULTSCREENS=N`
Sets the number of virtual screens to allocate at boot time. Useful for small root filesystems where the `wsconscfg`(8) utility is not wanted.

`options WSDISPLAY_SCROLLSUPPORT`
Enables scrolling support. The key combinations are **`LEFT SHIFT + PAGE UP`** and **`LEFT SHIFT + PAGE DOWN`** by default. Please note that this function may not work under the system console and is available depending on the framebuffer you are using.

Ioctls

The following `ioctl`(2) calls are provided by the **`wsdisplay`** driver or by devices which use it. Their definitions are found in ⟨`dev/wscons/wsconsio.h`⟩.

`WSDISPLAYIO_GTYPE (int)`
Retrieve the type of the display. The list of types is in ⟨`dev/wscons/wsconsio.h`⟩.

`WSDISPLAYIO_GINFO (struct wsdisplay_fbinfo)`
Retrieve basic information about a framebuffer display. The returned structure is as follows:

```
struct wsdisplay_fbinfo {
        u_int   height;
        u_int   width;
        u_int   depth;
        u_int   cmsize;
};
```

The *height* and *width* members are counted in pixels. The *depth* member indicates the number of bits per pixel, and *cmsize* indicates the number of color map entries accessible through `WSDISPLAYIO_GETCMAP` and `WSDISPLAYIO_PUTCMAP`. This call is likely to be unavailable on text-only displays.

`WSDISPLAYIO_GETCMAP (struct wsdisplay_cmap)`
Retrieve the current color map from the display. This call needs the following structure set up beforehand:

```
struct wsdisplay_cmap {
        u_int   index;
        u_int   count;
        u_char  *red;
        u_char  *green;
        u_char  *blue;
};
```

The *index* and *count* members specify the range of color map entries to retrieve. The *red*, *green*, and *blue* members should each point to an array of *count* `u_char`s. On return, these will be filled in with the appropriate entries from the color map. On all displays that support this call, values range from 0 for minimum intensity to 255 for maximum intensity, even if the display does not use eight bits internally to represent intensity.

`WSDISPLAYIO_PUTCMAP (struct wsdisplay_cmap)`
Change the display's color map. The argument structure is the same as for `WSDISPLAYIO_GETCMAP`, but *red*, *green*, and *blue* are taken as pointers to the values to use to set the color map. This call is not available on displays with fixed color maps.

`WSDISPLAYIO_GVIDEO (int)`
Get the current state of the display's video output. Possible values are:

`WSDISPLAYIO_VIDEO_OFF`
The display is blanked.

`WSDISPLAYIO_VIDEO_ON`
The display is enabled.

`WSDISPLAYIO_SVIDEO (int)`
Set the state of the display's video output. See `WSDISPLAYIO_GVIDEO` above for possible values.

`WSDISPLAYIO_GCURPOS (struct wsdisplay_curpos)`
Retrieve the current position of the hardware cursor. The returned structure is as follows:

```
struct wsdisplay_curpos {
        u_int x, y;
};
```

The *x* and *y* members count the number of pixels right and down, respectively, from the top-left corner of the display to the hot spot of the cursor. This call is not available on displays without a hardware cursor.

`WSDISPLAYOP_SCURPOS (struct wsdisplay_curpos)`
Set the current cursor position. The argument structure, and its semantics, are the same as for `WSDISPLAYIO_GCURPOS`. This call is not available on displays without a hardware cursor.

`WSDISPLAYIO_GCURMAX (struct wsdisplay_curpos)`
Retrieve the maximum size of cursor supported by the display. The *x* and *y* members of the returned structure indicate the maximum number of pixel rows and columns, respectively, in a hardware cursor on this display. This call is not available on displays without a hardware cursor.

`WSDISPLAYIO_GCURSOR (struct wsdisplay_cursor)`
Retrieve some or all of the hardware cursor's attributes. The argument structure is as follows:

```
struct wsdisplay_cursor {
        u_int   which;
        u_int   enable;
        struct wsdisplay_curpos pos;
        struct wsdisplay_curpos hot;
        struct wsdisplay_cmap cmap;
        struct wsdisplay_curpos size;
        u_char *image;
        u_char *mask;
};
```

The *which* member indicates which of the values the application requires to be returned. It should contain the logical OR of the following flags:

`WSDISPLAYIO_CURSOR_DOCUR`
: Get *enable*, which indicates whether the cursor is currently displayed (non-zero) or not (zero).

`WSDISPLAYIO_CURSOR_DOPOS`
: Get *pos*, which indicates the current position of the cursor on the display, as would be returned by `WSDISPLAYIO_GCURPOS`.

`WSDISPLAYIO_CURSOR_DOHOT`
: Get *hot*, which indicates the location of the "hot spot" within the cursor. This is the point on the cursor whose position on the display is treated as being the position of the cursor by other calls. Its location is counted in pixels from the top-right corner of the cursor.

`WSDISPLAYIO_CURSOR_DOCMAP`
: Get *cmap*, which indicates the current cursor color map. Unlike in a call to `WSDISPLAYIO_GETCMAP`, *cmap* here need not have its *index* and *count* members initialized. They will be set to 0 and 2 respectively by the call. This means that *cmap.red*, *cmap.green*, and *cmap.blue* must each point to at least enough space to hold two `u_char`s.

`WSDISPLAYIO_CURSOR_DOSHAPE`
: Get *size*, *image*, and *mask*. These are, respectively, the dimensions of the cursor in pixels, the bitmap of set pixels in the cursor and the bitmap of opaque pixels in the cursor. The format in which these bitmaps are returned, and hence the amount of space that must be provided by the application, are device-dependent.

`WSDISPLAYIO_CURSOR_DOALL`
: Get all of the above.

The device may elect to return information that was not requested by the user, so those elements of `struct wsdisplay_cursor` which are pointers should be initialized to `NULL` if not otherwise used. This call is not available on displays without a hardware cursor.

`WSDISPLAYIO_SCURSOR (struct wsdisplay_cursor)`
: Set some or all of the hardware cursor's attributes. The argument structure is the same as for `WSDISPLAYIO_GCURSOR`. The *which* member specifies which attributes of the cursor are to be changed. It should contain the logical OR of the following flags:

`WSDISPLAYIO_CURSOR_DOCUR`
: If *enable* is zero, hide the cursor. Otherwise, display it.

`WSDISPLAYIO_CURSOR_DOPOS`
Set the cursor's position on the display to *pos*, the same as `WSDISPLAYIO_SCURPOS`.

`WSDISPLAYIO_CURSOR_DOHOT`
Set the "hot spot" of the cursor, as defined above, to *hot*.

`WSDISPLAYIO_CURSOR_DOCMAP`
Set some or all of the cursor color map based on *cmap*. The *index* and *count* elements of *cmap* indicate which color map entries to set, and the entries themselves come from *cmap.red*, *cmap.green*, and *cmap.blue*.

`WSDISPLAYIO_CURSOR_DOSHAPE`
Set the cursor shape from *size*, *image*, and *mask*. See above for their meanings.

`WSDISPLAYIO_CURSOR_DOALL`
Do all of the above.

This call is not available on displays without a hardware cursor.

`WSDISPLAYIO_GMODE (u_int)`
Get the current mode of the display. Possible results include:

`WSDISPLAYIO_MODE_EMUL`
The display is in emulating (text) mode.

`WSDISPLAYIO_MODE_MAPPED`
The display is in mapped (graphics) mode.

`WSDISPLAYIO_MODE_DUMBFB`
The display is in mapped (frame buffer) mode.

`WSDISPLAYIO_SMODE (u_int)`
Set the current mode of the display. For possible arguments, see `WSDISPLAYIO_GMODE`.

`WSDISPLAYIO_LINEBYTES (u_int)`
Get the number of bytes per row, which may be the same as the number of pixels.

`WSDISPLAYIO_GMSGATTRS (struct wsdisplay_msgattrs)`
Get the attributes (colors and flags) used to print console messages, including separate fields for default output and kernel output. The returned structure is as follows:

```
struct wsdisplay_msgattrs {
        int default_attrs, default_bg, default_fg;
        int kernel_attrs, kernel_bg, kernel_fg;
};
```

The *default_attrs* and *kernel_attrs* variables are a combination of *WSATTR_∗* bits, and specify the attributes used to draw messages. The *default_bg*, *default_fg*, *kernel_bg* and *kernel_fg* variables specify the colors used to print messages, being '_bg' for the background and '_fg' for the foreground; their values are one of all the *WSCOL_∗* macros available.

`WSDISPLAYIO_SMSGATTRS (struct wsdisplay_msgattrs)`
Set the attributes (colors and flags) used to print console messages, including separate fields for default output and kernel output. The argument structure is the same as for `WSDISPLAYIO_GMSGATTRS`.

`WSDISPLAYIO_GBORDER (u_int)`
Retrieve the color of the screen border. This number corresponds to an ANSI standard color.

`WSDISPLAYIO_SBORDER (u_int)`
Set the color of the screen border, if applicable. This number corresponds to an ANSI standard color. Not all drivers support this feature.

`WSDISPLAYIO_GETWSCHAR (struct wsdisplay_char)`
Gets a single character from the screen, specified by its position. The structure used is as follows:

```
struct wsdisplay_char {
        int row, col;
        uint16_t letter;
        uint8_t background, foreground;
        char flags;
};
```

The *row* and *col* parameters are used as input; the rest of the structure is filled by the ioctl and is returned to you. *letter* is the ASCII code of the letter found at the specified position, *background* and *foreground* are its colors and *flags* is a combination of 'WSDISPLAY_CHAR_BRIGHT' and/or 'WSDISPLAY_CHAR_BLINK'.

`WSDISPLAYIO_PUTWSCHAR (struct wsdisplay_char)`
Puts a character on the screen. The structure has the same meaning as described in `WSDISPLAY_GETWSCHAR`, although all of its fields are treated as input.

`WSDISPLAYIO_SSPLASH (u_int)`
Toggle the splash screen. This call is only available with the `SPLASHSCREEN` kernel option.

`WSDISPLAYIO_SPROGRESS (u_int)`
Update the splash animation. This call is only available with the `SPLASHSCREEN` and `SPLASHSCREEN_PROGRESS` kernel options.

FILES

`/dev/ttyE*` Terminal devices (per screen).

`/dev/ttyEcfg` Control device.

`/dev/ttyEstat` Status device.

`/usr/include/dev/wscons/wsconsio.h`

SEE ALSO

`ioctl`(2), `pcdisplay`(4), `tty`(4), `i386/vesafb`(4), `vga`(4), `wscons`(4), `wsconscfg`(8), `wsconsctl`(8), `wsfontload`(8), `wsdisplay`(9)

BUGS

The **`wsdisplay`** code currently limits the number of screens on one display to 8.

The terms "wscons" and "wsdisplay" are not cleanly distinguished in the code and in manual pages.

"non-emulating" display devices are not tested.

NAME

wsfont — dynamic font loading support

SYNOPSIS

```
pseudo-device wsfont
```

DESCRIPTION

The **wsfont** pseudo device allows display fonts for the wscons subsystem to be loaded dynamically into the kernel. The fonts are loaded dynamically using the `wsfontload`(8) utility.

SEE ALSO

`wscons`(4), `wsdisplay`(4), `wsconsctl`(8), `wsfontload`(8), `wsfont`(9)

BUGS

Fonts cannot be unloaded from the kernel.

NAME

wskbd — generic keyboard support in wscons

SYNOPSIS

wskbd* at pckbd? console ? mux 1
(standard PC keyboard)
wskbd* at ukbd? console ? mux 1
(USB keyboard)
wskbd* at lkkbd? console ? mux 1
(DEC LK200/400 serial keyboard)
wskbd0 at akbd? console ? mux 1
(Apple ADB keyboard)
wskbd0 at nextkbd? console ? mux 1
(NeXT keyboard)
wskbd* at vrkiu? console ? mux 1
(NEC VR4000 series HPC keyboard)
wskbd* at skbd? console ? mux 1
(keyboard of misc hpcmips handheld devices)
wskbd* at btkbd? console ? mux 1
(Bluetooth keyboard)

DESCRIPTION

The **wskbd** driver handles common tasks for keyboards within the wscons(4) framework. It is attached to the hardware specific keyboard drivers and provides their connection to "wsdisplay" devices and a character device interface.

The common keyboard support consists of:

- Mapping from keycodes (defined by the specific keyboard driver) to keysyms (hardware independent, defined in /usr/include/dev/wscons/wsksymdef.h).
- Handling of "compose" sequences. Characters commonly not present as separate key on keyboards can be generated after either a special "compose" key is pressed or a "dead accent" character is used.
- Certain translations, like turning an "ALT" modifier into an "ESC" prefix.
- Automatic key repetition ("typematic").
- Parameter handling for "keyboard bells".
- Generation of "keyboard events" for use by X servers.

The **wskbd** driver provides a number of ioctl functions to control key maps and other parameters. These functions are accessible though the associated "wsdisplay" device as well. A complete list is in /usr/include/dev/wscons/wsconsio.h. The wsconsctl(8) utility allows to access key maps and other variables.

The *console* locator in the configuration line refers to the device's use as input part of the operating system console. A device specification containing a positive value here will only match if the device is in use as system console. (The console device selection in early system startup is not influenced.) This way, the console device can be connected to a known wskbd device instance.

FILES

/dev/wskbd*

`/usr/include/dev/wscons/wsksymdef.h`

`/usr/include/dev/wscons/wsconsio.h`.

SEE ALSO

`btkbd`(4), `pckbd`(4), `ukbd`(4), `wscons`(4), `wsmux`(4), `wsconsctl`(8), `wskbd`(9)

NAME

wsmouse — generic mouse support in wscons

SYNOPSIS

wsmouse* **at pms? mux 0**
(PS/2 mouse, including ‘‘IntelliMouse’’-compatible wheel mice)
wsmouse* **at ums? mux 0**
(USB mouse)
wsmouse* **at lms? mux 0**
(Logitech bus mouse, i386 only)
wsmouse* **at mms? mux 0**
(Microsoft InPort mouse, i386 only)
wsmouse0 **at ams? mux 0**
(Apple ADB mouse)
wsmouse* **at btms? mux 0**
(Bluetooth mouse)

DESCRIPTION

The **wsmouse** driver is an abstraction layer for mice within the wscons(4) framework. It is attached to the hardware specific mouse drivers and provides a character device interface which returns *struct wscons_event* via read(2). For use with X servers, “mouse events” can be generated.

The wsconsctl(8) utility gives access to several configurable details that affect this driver.

Ioctls

The following ioctl(2) calls are provided by the **wsmouse** driver or by devices which use it. Their definitions are found in dev/wscons/wsconsio.h.

WSMOUSEIO_GETREPEAT (struct wsmouse_repeat)
Retrieve the current automatic button repeating configuration. The structure returned is as follows:

```
struct wsmouse_repeat {
        unsigned long   wr_buttons;
        unsigned int    wr_delay_first;
        unsigned int    wr_delay_decrement;
        unsigned int    wr_delay_minimum;
};
```

The *wr_buttons* field is a bit mask that specifies which buttons send press and release events periodically while they are physically held down. The least significant bit corresponds to button 0.

The other three fields describe the frequency upon which these automatic events are sent. *wr_delay_first* specifies the milliseconds before the first repeated event is sent. *wr_delay_decrement* is used to calculate the delay between the most recently generated event and the forthcoming one: the previous delay is taken and it is decreased by the value given in this variable. *wr_delay_minimum* specifies the minimum delay, in milliseconds, between two consecutive events.

WSMOUSEIO_SETREPEAT (struct wsmouse_repeat)
Set the automatic button repeating configuration. See WSMOUSEIO_GETREPEAT above for more details.

FILES

`/dev/wsmouse*`

`/usr/include/dev/wscons/wsconsio.h`.

SEE ALSO

`btms`(4), `lms`(4), `mms`(4), `pms`(4), `uep`(4), `ums`(4), `wscons`(4), `wsmux`(4), `moused`(8), `wsconsctl`(8), `wsmoused`(8), `wsmouse`(9)

NAME

wsmux — console keyboard/mouse multiplexor for wscons

SYNOPSIS

```
wskbd*   at ... mux 1
wsmouse* at ... mux 0

pseudo-device wsmux
```

DESCRIPTION

The **wsmux** is a pseudo-device driver that allows several wscons(4) input devices to have their events multiplexed into one stream.

The typical usage for this device is to have two multiplexors, one for mouse events and one for keyboard events. All wsmouse(4) devices should direct their events to the mouse mux (normally 0) and all keyboard devices, except the console, should direct their events to the keyboard mux (normally 1). A device will send its events to the mux indicated by the *mux* locator. If none is given the device will not use a multiplexor. The keyboard multiplexor should be connected to the display, using the wsconscfg(8) command. It will then receive all keystrokes from all keyboards and, furthermore, keyboards can be dynamically attached and detached without further user interaction. In a similar way, the window system will open the mouse multiplexor and receive all mouse events; mice can also be dynamically attached and detached.

If a wskbd(4) or wsmouse(4) device is opened despite having a mux it will be detached from the mux.

It is also possible to inject events into a multiplexor from a user program.

FILES

For each mux device, /dev/wsmuxN there is a control device /dev/wsmuxctlN. The control device has a minor number 128 greater than the regular mux device. It can be used to control the mux even when it is open, e.g., by wsmuxctl(8).

/dev/wsmouse a.k.a. /dev/wsmux0

/dev/wskbd a.k.a. /dev/wsmux1

/usr/include/dev/wscons/wsconsio.h

SEE ALSO

wscons(4), wsdisplay(4), wskbd(4), wsmouse(4), moused(8), wsconscfg(8), wsconsctl(8), wsfontload(8), wsmoused(8), wsmuxctl(8)

NAME

wss — Windows Sound System hardware audio driver

SYNOPSIS

```
wss0   at isa? port 0x530 irq 10 drq 0 drq2 1
wss0   at isa? port 0x530 irq 10 drq 0 flags 1
wss*   at acpi?
wss*   at isapnp?
wss*   at pnpbios? index ?
audio* at audiobus?
opl*   at wss?
```

DESCRIPTION

The **wss** driver supports Microsoft's Windows Sound System, the MAD16 chip based hardware, and their clones.

The base I/O port is set by a jumper on the board; valid choices are 0x530, 0x604, 0xE80, or 0xF40. Both IRQ and DMA channels are software programmable. The IRQ may be set to 7, 9, 10, or 11, and the DMA channel may be set to 0, 1, or 3.

The device **flags** must have bit 0 (value 1) set to enable the MAD16 support. This is to avoid potential conflicts with other devices when probing the MAD16 because it requires use of extra I/O ports not in the base port range. Chips needing this flag include the OPTi 82C928, 82C929, 82C831, and the OAK OTI-601D. Setting bit 1 (value 2) in **flags** disables the joystick port on MAD16 hardware.

SEE ALSO

acpi(4), audio(4), isa(4), isapnp(4), joy(4), opl(4), pnpbios(4)

NAME

wt — Archive/Wangtek cartridge tape driver

SYNOPSIS

```
wt0 at isa? port 0x300 irq 5 drq 1
```

DESCRIPTION

The **wt** driver provides support for the following Archive and Wangtek boards:

QIC-02
QIC-36

Raw devices are indicated by an “r” preceding the device name. The driver can be opened with either rewind on close (`/dev/rst*`) or no rewind on close (`/dev/nrst*`) options.

The tape format is specified by adding 8 or 16 to the device number. For example, for the wt0 device’s raw rewind on close device nodes, `/dev/rwt0` is the 150 MByte density device, `/dev/rwt8` is the 120 MByte density device, and `/dev/rwt16` is the 60 MByte device.

SEE ALSO

`intro`(4)

BUGS

The **wt** driver conflicts unpleasantly with the **we** driver (SMC Ethernet adapters) at the same I/O address. The probe reprograms their EEPROMs.

NAME

xbox — SPARC SBus Expansion Subsystem

SYNOPSIS

```
xbox*   at sbus? slot ? offset ?
sbus*   at xbox?
```

DESCRIPTION

The **xbox** driver provides support for the Sun SBus Expansion Subsystem. This device consists of an SBus card and a chassis which has three additional SBus slots.

SEE ALSO

intro(4), sbus(4)

HISTORY

Support for the **xbox** first appeared in NetBSD 1.4.

NAME

xcfb — DECstation 5000/xx PMAG-DV colour framebuffer

SYNOPSIS

```
xcfb* at tc? slot ? offset ?
wsdisplay* at xcfb?
```

DESCRIPTION

The **xcfb** driver provides support for the PMAG-DV framebuffer found on the DECstation 5000/xx baseboard. It is an 8 bpp colour framebuffer capable of running at a resolution of 1024-by-768 at 66 Hz.

SEE ALSO

cfb(4), mfb(4), pm(4), px(4), pxg(4), sfb(4), tc(4), tfb(4), wscons(4)

NAME

xd — Xylogics 753 or 7053 VME SMD disk controller driver

SYNOPSIS

```
xdc0    at vmel0 addr 0xffffee80 level 3 vect 0x44 (sun4)
xdc1    at vmel0 addr 0xffffee90 level 3 vect 0x45 (sun4)
xdc2    at vmel0 addr 0xffffeea0 level 3 vect 0x46 (sun4)
xdc3    at vmel0 addr 0xffffeeb0 level 3 vect 0x47 (sun4)
xd*     at xdc? drive ?                            (sun4)
```

DESCRIPTION

The **xd** is a Xylogics 753 or 7053 SMD disk controller found on sun4 systems with the VME bus. The Xylogics 753 and 7053 are programmed the same way, but are different sizes. The 753 is a 6U VME card, while the 7053 is a 9U VME card.

SEE ALSO

intro(4), xy(4)

Xylogics manuals (scanned):
http://www.bitsavers.org/pdf/xylogics/

HISTORY

The **xd** driver first appeared in NetBSD 1.1 and was written by Charles D. Cranor.

NAME

xge — Neterion Xframe-I Ten Gigabit Ethernet driver

SYNOPSIS

xge* **at pci? dev ? function ?**

DESCRIPTION

The **xge** device driver supports the Neterion Xframe-I LR Ethernet adapter, which uses a single mode fiber (1310nm) interface.

The Xframe supports IPv4/TCP/UDP checksumming in hardware, as well as TCP Segmentation Offloading (TSO) and hardvare VLAN handling. The driver currently does not support the hardware VLAN feature. See `ifconfig`(8) for information on how to enable TSO and hardware checksum calculation.

DIAGNOSTICS

xge%s: failed configuring endian, %llx != %llx! The Xframe could not be turned into the correct endian operation. This is most likely a hardware error.

xge%d: failed allocating txmem.
xge%d: failed allocating rxmem. The computer has run out of kernel memory.

xge%d: adapter not quiescent, aborting
xge%d: ADAPTER_STATUS missing bits %s The Xframe could not be turned into a usable state. Most likely an Xframe hardware error.

xge%d: cannot create TX DMA maps
xge%d: cannot create RX DMA maps This error is either a kernel error or that the kernel has run out of available memory.

xge%d: bad compiler struct alignment, %d != %d The compiler did not align the structure correctly. This is a compiler problem.

SEE ALSO

`arp`(4), `ifmedia`(4), `netintro`(4), `pci`(4), `ifconfig`(8)

HISTORY

The **xge** driver first appeared in NetBSD 3.0.

AUTHORS

The **xge** driver was written by Anders Magnusson ⟨ragge@ludd.luth.se⟩.

BUGS

There should be an XGMII framework for the driver to use.

NAME

xi — Xircom CreditCard Ethernet device driver

SYNOPSIS

```
xi* at xirc?
```

Configuration of PHYs may also be necessary. See mii(4).

DESCRIPTION

The **xi** driver provides support for the Xircom CreditCard family of PCMCIA Ethernet adapters. Supported cards include:

Xircom CreditCard Ethernet II
Xircom CreditCard 10/100 Ethernet
Xircom RealPort 10/100 Ethernet + Modem (Ethernet function only)
Intel EtherExpress Pro/100
Compaq Netelligent 10/100

Cards which should work, but have not been confirmed include:

Xircom RealPort Ethernet
Xircom RealPort 10/100 Ethernet

Some Xircom Ethernet products are supported by the tlp(4) driver.

MEDIA SELECTION

Media selection is done using ifconfig(8) using the standard ifmedia(4) mechanism. Refer to those manual pages for more information.

SEE ALSO

ifmedia(4), mii(4), netintro(4), pcmcia(4), tlp(4), xirc(4), ifconfig(8)

HISTORY

The **xi** device driver appeared in NetBSD 1.5.

BUGS

The driver suffers from poor performance. Even with the 10/100 cards, do not expect more than ~450KB/s. Some 10/100 cards may not autonegotiate reliably and require manual media selection.

The Xircom multifunction cards which contain both Ethernet and modem interfaces are known to have problems. This is due to the card not reporting itself correctly as a multifunction card.

NAME

xirc — Xircom Ethernet/modem card driver

SYNOPSIS

```
xirc* at pcmcia? function ?
com*  at xirc?
xi*   at xirc?
```

DESCRIPTION

The **xirc** device driver provides support for the Xircom combined Ethernet and modem cards.

SEE ALSO

com(4), pcmcia(4), xi(4)

HISTORY

The **xirc** driver appeared in NetBSD 3.0.

NAME

xy — Xylogics 450 or 451 VME SMD disk controller driver

SYNOPSIS

```
xyc0    at vmes0 addr 0xffffee40 level 3 vect 0x48 (sun4)
xyc1    at vmes0 addr 0xffffee48 level 3 vect 0x49 (sun4)
xy*     at xyc? drive ?                            (sun4)
```

DESCRIPTION

The **xy** is a Xylogics 450/451 SMD disk controller found on sun4 systems with the VME bus.

SEE ALSO

intro(4), xd(4)

Xylogics manuals (scanned):
http://www.bitsavers.org/pdf/xylogics/

HISTORY

The **xy** driver first appeared in NetBSD 1.1 and was written by Charles D. Cranor.

NAME

yds — Yamaha DS-1 series PCI audio controller device driver

SYNOPSIS

```
yds*   at pci? dev ? function ?
audio* at audiobus?
mpu*   at yds?
opl*   at yds?
```

DESCRIPTION

The **yds** device driver supports built-in sound devices and sound cards based on Yamaha YMF724/740/744/754 PCI audio controller chip, also known as DS-1 series.

SEE ALSO

ac97(4), audio(4), mpu(4), opl(4), pci(4)

HISTORY

The **yds** device driver appeared in NetBSD 1.5.1.

BUGS

The game port is not supported.

The volume of the OPL part is fixed.

NAME

ym — Yamaha OPL3-SA2 and OPL3-SA3 audio device driver

SYNOPSIS

```
ym*    at acpi?
ym*    at isapnp?
ym*    at pnpbios? index ?
audio* at audiobus?
mpu*   at ym?
opl*   at ym?
```

DESCRIPTION

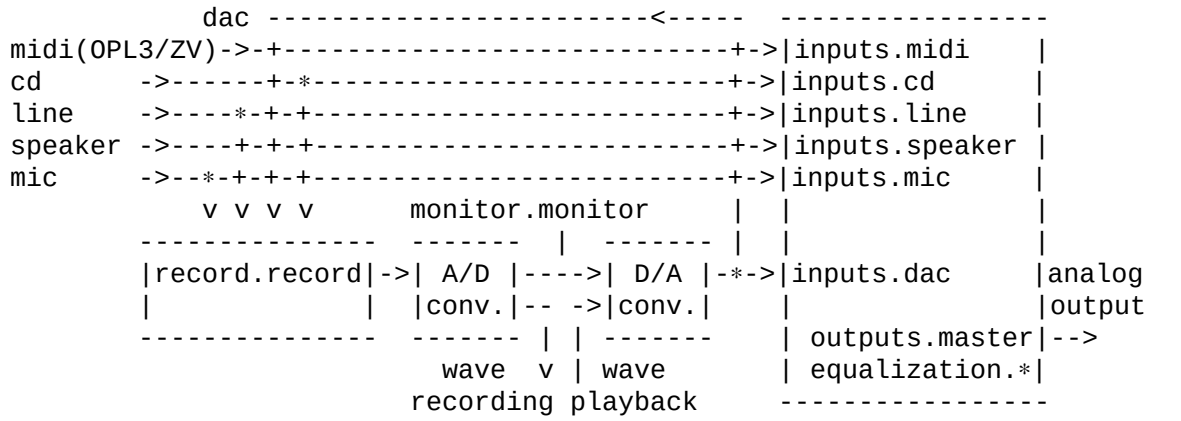
The **ym** driver provides support for Yamaha YMF711 (OPL3-SA2) and YMF715x (OPL3-SA3) sound devices.

The OPL3-SAx device has WSS compatible full-duplex 16bit CODEC, OPL3 FM synthesizer, and MPU401 compatible MIDI I/O port interface. Additionally, OPL3-SA3 has built-in '3D Enhanced' equalizer.

The joystick interface is handled by the `joy`(4) driver.

MIXER DEVICE

The mixer device of **ym** driver can be accessed by `mixerctl`(1) command. The layout is shown below.



Note that the '`inputs.dac`' is twice as sensitive as other '`inputs`' volume variables.

The hardware volume changes the '`outputs.master`' value.

If an external input source is unmuted by setting corresponding '`inputs.*.mute`' variable to '`off`', the device is never put in global power down or power save mode. This is because if the device is in global power down or power save mode, the output is automatically muted.

All the external input sources (CD playback, line input, speaker, and MIC) are muted by default.

The '`equalization.*`' variables does not exists on OPL3-SA2. The 'equalization.treble' and 'equalization.bass' are enhancement only, and any values below the center position (128) don't take any effect.

POWER MANAGEMENT

The **ym** driver is capable of power management on the OPL3-SAx devices. The following modes can be selected by setting '`power.save`' variable of `mixerctl`(1) to '`powerdown`', '`powersave`', and '`nosave`' respectively.

Global power-down mode

When a subpart of the device is unused, the part is power-down after a timeout period (specified by '`power.save.timeout`' variable of `mixerctl`(1) in seconds). When all the subparts of the device are unused, and all the external input sources are muted, the driver puts the device in 'Global Power Down' mode.

On the global power down mode, the power consumption is minimized (10μA (SA3) or 200μA (SA2) typ.), but the click noise on power up/down the device is rather loud. *This mode should not be used with headphones or hi−fi audio systems, or your ears or the systems may be damaged.*

Power save mode

When a subpart of the device is unused, the part is powered-down after a timeout period (specified by '`power.save.timeout`' variable of `mixerctl`(1) in seconds). When all the subparts of the device are unused, and all the external input sources are muted, the driver put the device in 'Power Save' mode.

In power save mode, the power consumption is reduced (5mA typ.). The click noise on power up/down of the device is very small, but this operation requires muting/unmuting the device, which make some noise. In order to reduce the noise, setting the master volume at the small value is effective.

No power-save mode

Once the device is powered-up, it remains on after the use of the device. Once a subpart of the device is powered-up, it shall not be power-down. This mode minimizes click noises on power switching, but maximizes power consumption (30-100mA).

On suspending, the device is put into power-save state.

SEE ALSO

`mixerctl`(1), `apm`(4), `audio`(4), `isapnp`(4), `joy`(4), `midi`(4), `mpu`(4), `opl`(4), `pnpbios`(4)

HISTORY

The **`ym`** device driver appeared in NetBSD 1.4.

BUGS

Although the parameters of the device are saved and restored on `apm`(4) suspend/resume, the DMA state is not restored. That is, if the system suspends during playback, this is not continued after suspend/resume cycle.

The joystick port is not under power management. If a `joy`(4) device is configured, the device will never be put in global power down or power save mode.

The external devices, such as Zoomed Video port, OPL4-ML/2, modem, and CD-ROM are not supported.

NAME

zero — the zero device

DESCRIPTION

The **zero** special device provides an infinite stream of zeros.

FILES

`/dev/zero`

NAME

zsc — serial communications interface

SYNOPSIS

```
zsc0 at obio?
```

DESCRIPTION

The **zsc** driver provides support for Z8530- and Z85230-compatible EIA RS-232C (CCITT V.28) communications interfaces. The Z8530 has a 3 character buffer, and the Z85230 has a 8 character buffer. However, due to limitations in the design of the Z85230, the driver is unable to distinguish it from the Z8530 and will not enable any enhanced features of the controller.

Input and output for each line may set to one of following baud rates; 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, or any other baud rate which is a factor of 115200.

FILES

`/dev/tty00`

`/dev/tty01`

SEE ALSO

`tty`(4), `zstty`(4)

HISTORY

The **zsc** driver was originally derived from the sun3 **zs** driver and is currently under development.

BUGS

Data loss is possible when using high speeds, particularly on older (68020 or 68030) Macintosh models and on busy systems.

NAME

zsc — serial communications interface

SYNOPSIS

```
zsc0 at pcc? ipl 4
```

DESCRIPTION

The **zsc** driver provides support for the two Z8530- EIA RS-232C (CCITT V.28) dual channel communications interfaces present on Motorola MVME147 single board computers. The **zsc** driver implements a convenient abstraction layer, which provides two channels per chip for the zstty(4) driver to attach to.

FILES

/dev/console

/dev/ttyZ1

/dev/ttyZ2

/dev/ttyZ3

SEE ALSO

pcc(4), zstty(4)

BUGS

Data loss is possible when using the higher bitrates on busy systems.

NAME

zssc — Zeus low level SCSI interface

SYNOPSIS

```
zssc0 at zbus0
```

DESCRIPTION

The Amiga architecture uses a common machine independent scsi sub-system provided in the kernel source. The machine independent drivers that use this code access the hardware through a common interface. (see scsibus(4)) This common interface interacts with a machine dependent interface, such as **zssc**, which then handles the hardware specific issues.

The **zssc** interface handles things such as DMA and interrupts as well as actually sending commands, negotiating synchronous or asynchronous transfers and handling disconnect/reconnect of SCSI targets. The hardware that **zssc** uses is based on the NCR53c710 SCSI chip.

DIAGNOSTICS

zssc%s: abort %s: dstat %02x, sstat0 %02x sbcl %02x The scsi operation %s was aborted due to error. Dstat, sstat and sbcl are registers within the NCR53c710 SCSI chip.

siop id %d reset0 The NCR53c710 SCSI chip has been reset and configure at id %d.

SIOP interrupt: %x sts %x msg %x sbcl %x The NCR53c710 SCSI chip has interrupted unexpectedly.

SIOP: SCSI Gross Error The NCR53c710 SCSI chip has indicated that it is confused.

SIOP: Parity Error The NCR53c710 SCSI chip has indicated that it has detected a parity error on the SCSI bus.

SEE ALSO

scsibus(4)

HISTORY

The **zssc** interface first appeared in NetBSD 1.0

NAME

zstty, **zsc**, **zs** — Zilog 8530 serial communications interface

SYNOPSIS

alpha (DEC 3000)

```
zsc0  at ioasic? offset 0x100000
zsc1  at ioasic? offset 0x180000
zstty0      at zsc0 channel ? # serial ports on B channels
zstty2      at zsc1 channel ? # serial ports on B channels
lkkbd0      at zsc1 channel ? # keyboard port on A channels
vsms0 at zsc0 channel ? # mouse port on A channels
```

cesfic

```
zsc*  at mainbus0
zstty*at zsc? channel ?
```

mac68k and macppc

```
zsc0  at obio?
zstty*at zsc? channel ?
```

mipsco

```
zsc0  at obio0 addr 0xbb000000
zstty0      at zsc0 channel 0
zstty1      at zsc0 channel 1
```

mvme68k

```
zsc*  at pcc? ipl 4
zsc*  at pcctwo? ipl 4
zstty*at zsc? channel ?
```

news68k

```
zsc0  at hb0 addr 0xe0d40000 ipl 5 vect 64 flags 0x0  # news1700
zsc0  at hb1 addr 0xe1780000 ipl 5 vect 64 flags 0x1  # news1200
zstty0      at zsc0 channel 0
zstty1      at zsc0 channel 1
```

newsmips

```
zsc0  at hb0 addr 0xbfec0000 level 1 flags 0x0  # on-board
zsc1  at hb0 addr 0xb8c40100 level 1 flags 0x1  # expansion board
zsc2  at hb0 addr 0xb8c40104 level 1 flags 0x1
zsc0  at ap?
zstty0      at zsc0 channel 0
zstty1      at zsc0 channel 1
zstty2      at zsc1 channel 0
zstty3      at zsc1 channel 1
zstty4      at zsc2 channel 0
zstty5      at zsc2 channel 1
```

next68k

```
zsc0  at intio? ipl 5
#zsc1 at intio? ipl 5
zstty0      at zsc0 channel 0 # ttya
```

```
zstty1      at zsc0 channel 1 # ttyb
```

sgimips

```
zsc*  at hpc0 offset ?
zstty*at zsc? channel ?
```

sparc

```
zs0   at mainbus0                  # sun4c
zs0   at obio0                     # sun4m
zs0   at obio0 addr 0xf1000000 level 12   # sun4/200 and sun4/300
zs0   at obio0 addr 0x01000000 level 12   # sun4/100
zstty0      at zs0 channel 0              # ttya
zstty1      at zs0 channel 1              # ttyb
zs1   at mainbus0                  # sun4c
zs1   at obio0                     # sun4m
zs1   at obio0 addr 0xf0000000 level 12   # sun4/200 and sun4/300
zs1   at obio0 addr 0x00000000 level 12   # sun4/100
kbd0  at zs1 channel 0             # keyboard
ms0   at zs1 channel 1             # mouse
zs2   at obio0 addr 0xe0000000 level 12   # sun4/300
zstty2      at zs2 channel 0              # ttyc
zstty3      at zs2 channel 1              # ttyd
```

sun2

```
zs0   at obio0 addr 0x002000  # 2/120, 2/170
zs1   at obmem0 addr 0x780000 # 2/120, 2/170
zs0   at obio0 addr 0x7f2000  # 2/50
zs1   at obio0 addr 0x7f1800  # 2/50
zs2   at mbmem0 addr 0x080800 # 2/120, 2/170 (first sc SCSI)
zs3   at mbmem0 addr 0x081000 # 2/120, 2/170 (first sc SCSI)
zs4   at mbmem0 addr 0x084800 # 2/120, 2/170 (second sc SCSI)
zs5   at mbmem0 addr 0x085000 # 2/120, 2/170 (second sc SCSI)
zstty*at zs? channel ?  # ttya
kbd0  at zstty?         # keyboard
ms0   at zstty?         # mouse
```

sun3

```
zstty0      at zsc1 channel 0 # ttya
zstty1      at zsc1 channel 1 # ttyb
kbd0  at zsc0 channel 0 # keyboard
ms0   at zsc0 channel 1 # mouse
```

x68k

```
zsc0  at intio0 addr 0xe98000 intr 112
zstty0      at zsc0 channel 0       # built-in RS-232C
ms0   at zsc0 channel 1       # standard mouse
#zsc1 at intio0 addr 0xeafc00 intr 113
#zstty2     at zsc1 channel 0
#zstty3     at zsc1 channel 1
#zsc2 at intio0 addr 0xeafc10 intr 114
#zstty4     at zsc2 channel 0
#zstty5     at zsc2 channel 1
```

DESCRIPTION

The **zstty** driver provides TTY support for Zilog 8530 Dual UART chips.

Input and output for each line may set to any baud rate in the range 50 to 38400 (and higher on some machines).

FILES

alpha

`/dev/ttyB0`

`/dev/ttyB1`

others

`/dev/ttya`

`/dev/ttyb`

DIAGNOSTICS

zs0∗: fifo overflow

The on-chip “FIFO” has overflowed and incoming data has been lost. This generally means the machine is not responding to interrupts from the ZS chip fast enough, which can be remedied only by using a lower baud rate.

zs0∗: ring overflow

The software input "ring" has overflowed. This usually means input flow-control is not configured correctly (i.e. incorrect cable wiring).

SEE ALSO

`kbd`(4), `ms`(4), `scc`(4), `tty`(4)

HISTORY

The **zstty** driver was derived from the **sparc zs** driver supplied with 4.4BSD UNIX.

CAVEATS

`/dev/ttyB1` on alpha is created by `MAKEDEV`(8) with minor number 2, so the corresponding device should be zstty2, not zstty1.

BUGS

The old Zilog 8530 chip has a very small FIFO (3 bytes?) and therefore has very strict latency requirements for the interrupt service routine. This limits the usable baud rates on many machines.

NAME

zx — Sun ZX / Leo frame buffer driver

SYNOPSIS

zx* **at sbus? slot ? offset ?**

DESCRIPTION

The **zx** is a memory based color frame buffer, with graphics acceleration and overlay capabilities. Its control registers, colour lookup table and pixel memory can be mapped into a user process address space by using the mmap(2) system call. The **zx** driver supports the minimal ioctl's needed to run X(7).

SEE ALSO

bwtwo(4), cgtwo(4), cgthree(4), cgfour(4), cgsix(4), cgeight(4)

NAME

zyd — ZyDAS ZD1211/ZD1211B USB IEEE 802.11b/g wireless network device

SYNOPSIS

zyd* **at uhub? port ?**

DESCRIPTION

The **zyd** driver provides support for wireless network adapters based around the ZyDAS ZD1211 and ZD1211B USB chips.

These are the modes the **zyd** driver can operate in:

BSS mode — Also known as *infrastructure* mode, this is used when associating with an access point, through which all traffic passes. This mode is the default.

monitor mode — In this mode the driver is able to receive packets without associating with an access point. This disables the internal receive filter and enables the card to capture packets from networks which it wouldn't normally have access to, or to scan for access points.

zyd supports software WEP. Wired Equivalent Privacy (WEP) is the de facto encryption standard for wireless networks. It can be typically configured in one of three modes: no encryption; 40-bit encryption; or 104-bit encryption. Unfortunately, due to serious weaknesses in WEP protocol it is strongly recommended that it not be used as the sole mechanism to secure wireless communication. WEP is not enabled by default.

CONFIGURATION

The **zyd** driver can be configured at runtime with `ifconfig`(8) or on boot with `ifconfig.if`(5) using the following parameters:

bssid *bssid*
Set the desired BSSID.

-bssid
Unset the desired BSSID. The interface will automatically select a BSSID in this mode, which is the default.

chan *n*
Set the channel (radio frequency) to be used by the driver based on the given channel ID *n*.

-chan
Unset the desired channel to be used by the driver. The driver will automatically select a channel in this mode, which is the default.

media *media*
The **zyd** driver supports the following *media* types:

autoselect Enable autoselection of the media type and options.
DS1 Set 802.11b DS 1Mbps operation.
DS2 Set 802.11b DS 2Mbps operation.
DS5 Set 802.11b DS 5.5Mbps operation.
DS11 Set 802.11b DS 11Mbps operation.

mediaopt *opts*
The **zyd** driver supports the following media options:

ibss Select Independent Basic Service Set (IBSS) operation.

-mediaopt *opts*
Disable the specified media options on the driver and return it to the default mode of operation (BSS).

nwid *id*
Set the network ID. The *id* can either be any text string up to 32 characters in length, or a series of hexadecimal digits up to 64 digits. An empty *id* string allows the interface to connect to any available access points. By default the **zyd** driver uses an empty string. Note that network ID is synonymous with Extended Service Set ID (ESSID).

-nwid
Set the network ID to the empty string to allow the interface to connect to any available access point.

nwkey *key*
Enable WEP encryption using the specified *key*. The *key* can either be a string, a series of hexadecimal digits (preceded by '0x'), or a set of keys of the form "n:k1,k2,k3,k4", where 'n' specifies which of the keys will be used for transmitted packets, and the four keys, "k1" through "k4", are configured as WEP keys. If a set of keys is specified, a comma (',') within the key must be escaped with a backslash. Note that if multiple keys are used, their order must be the same within the network. **zyd** is capable of using both 40-bit (5 characters or 10 hexadecimal digits) or 104-bit (13 characters or 26 hexadecimal digits) keys.

-nwkey
Disable WEP encryption. This is the default mode of operation.

HARDWARE

The following devices are known to be supported by the **zyd** driver:

3COM 3CRUSB10075
Acer WLAN-G-US1
Airlink+ AWLL3025
Airlink 101 AWLL3026
AOpen 802.11g WL54
Asus A9T integrated wirless
Asus WL-159g
Belkin F5D7050 v.4000
Billion BiPAC 3011G
Buffalo WLI-U2-KG54L
CC&C WL-2203B
DrayTek Vigor 550
Edimax EW-7317UG
Edimax EW-7317LDG
Fiberline Networks WL-43OU
iNexQ UR055g
Linksys WUSBF54G
Longshine LCS-8131G3
MSI US54SE
Philips SNU5600
Planet WL-U356
Planex GW-US54GZ
Planex GW-US54GZL

Planex GW-US54Mini
Safecom SWMULZ-5400
Sagem XG 760A
Sagem XG 76NA
Sandberg Wireless G54 USB
Sitecom WL-113
SMC SMCWUSB-G
Sweex wireless USB 54 Mbps
Tekram/Siemens USB adapter
Telegent TG54USB
Trendnet TEW-424UB
Trendnet TEW-429UB
TwinMOS G240
US Robotics 5423
X-Micro XWL-11GUZX
Yakumo QuickWLAN USB
Zonet ZEW2501
ZyXEL ZyAIR G-220

FILES

The adapter needs some firmware files, which are loaded on demand by the driver when a device is attached:

```
/libdata/firmware/zyd/zyd-zd1211
/libdata/firmware/zyd/zyd-zd1211b
```

See `firmload`(9) for how to change this.

EXAMPLES

The following `ifconfig.if`(5) example configures zyd0 to join whatever network is available on boot, using WEP key "0x1deadbeef1", channel 11:

```
inet 192.168.1.1 netmask 255.255.255.0 nwkey 0x1deadbeef1 chan 11
```

Configure zyd0 for WEP, using hex key "0x1deadbeef1":

```
# ifconfig zyd0 nwkey 0x1deadbeef1
```

Return zyd0 to its default settings:

```
# ifconfig zyd0 -bssid -chan media autoselect \
        nwid "" -nwkey
```

Join an existing BSS network, "my_net":

```
# ifconfig zyd0 192.168.0.2 netmask 0xffffff00 nwid my_net
```

DIAGNOSTICS

zyd%d: could not read firmware file %s (error=%d) For some reason, the driver was unable to read the firmware file from the filesystem. The file might be missing or corrupted.

zyd%d: could not load firmware (error=%d) An error occurred while attempting to upload the firmware to the onboard microcontroller unit.

zyd%d: could not send command (error=%s) An attempt to send a command to the firmware failed.

zyd%d: sorry, radio %s is not supported yet Support for the specified radio chip is not yet implemented in the driver. The device will not attach.

zyd%d: device version mismatch: 0x%x (only >= 43.30 supported) Early revisions of the ZD1211 chipset are not supported by this driver. The device will not attach.

zyd%d: device timeout A frame dispatched to the hardware for transmission did not complete in time. The driver will reset the hardware. This should not happen.

SEE ALSO

`arp`(4), `ifmedia`(4), `intro`(4), `netintro`(4), `usb`(4), `ifconfig.if`(5), `ifconfig`(8,) `firmload`(9)

AUTHORS

The **zyd** driver was written by Florian Stoehr ⟨ich@florian-stoehr.de⟩, Damien Bergamini ⟨damien@openbsd.org⟩, and Jonathan Gray ⟨jsg@openbsd.org⟩.

CAVEATS

The **zyd** driver does not support a lot of the functionality available in the hardware. More work is required to properly support the IBSS and power management features.