



ChorusOS man pages section 9DRV: Driver Implementations

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 806-3343
December 10, 1999

Copyright 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303-4900 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, docs.sun.com, AnswerBook, AnswerBook2, ChorusOS, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, Californie 94303-4900 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, AnswerBook, AnswerBook2, ChorusOS, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



Contents

PREFACE 5

amd29xxx(9DRV) 11

benchns16550(9DRV) 13

cheerio(9DRV) 15

dec2115x (9DRV) 18

dec21x4x(9DRV) 24

ebus(9DRV) 29

el3(9DRV) 32

epic100(9DRV) 36

fccEther(9DRV) 40

i8254(9DRV) 44

intro(9DRV) 47

isabios(9DRV) 48

isapci(9DRV) 50

m48txx(9DRV) 52

mc146818(9DRV) 54

ne2000(9DRV) 56

ns16550(9DRV) 59

pcibios(9DRV) 62

pciconf(9DRV)	64
pcienum(9DRV)	67
quicc8260(9DRV)	69
quicc8xx(9DRV)	74
raven(9DRV)	79
ric(9DRV)	83
sabre(9DRV)	85
sccEther(9DRV)	88
sccUart (9DRV)	92
simba(9DRV)	95
smc1660(9DRV)	98
smcuart (9DRV)	101
tbDec(9DRV)	104
universe(9DRV)	106
univRemCom(9DRV)	111
vt82c586(9DRV)	114
w83c553 (9DRV)	119
z85x30(9DRV)	122
Index	124

PREFACE

Overview

A man page is provided for both the naive user, and sophisticated user who is familiar with the ChorusOS™ operating system and is in need of on-line information. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

The following is a list of sections in the ChorusOS man pages and the information it references:

- *Section 1CC: User Utilities; Host and Target Utilities*
- *Section 1M: System Management Utilities*
- *Section 2DL: System Calls; Data Link Services*
- *Section 2K: System Calls; Kernel Services*
- *Section 2MON: System Calls; Monitoring Services*
- *Section 2POSIX: System Calls; POSIX System Calls*
- *Section 2RESTART: System Calls; Hot Restart and Persistent Memory*
- *Section 2SEG: System Calls; Virtual Memory Segment Services*
- *Section 3FTPD: Libraries; FTP Daemon*
- *Section 3M: Libraries; Mathematical Libraries*
- *Section 3POSIX: Libraries; POSIX Library Functions*
- *Section 3RPC: Libraries; RPC Services*
- *Section 3STDC: Libraries; Standard C Library Functions*
- *Section 3TELD: Libraries; Telnet Services*
- *Section 4CC: Files*

- *Section 5FEA: ChorusOS Features and APIs*
- *Section 7P: Protocols*
- *Section 7S: Services*
- *Section 9DDI: Device Driver Interfaces*
- *Section 9DKI: Driver to Kernel Interface*
- *Section 9DRV: Driver Implementations*

ChorusOS man pages are grouped in Reference Manuals, with one reference manual per section.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and `man(1)` for more information about man pages in general.

NAME	This section gives the names of the commands or functions documented, followed by a brief description of what they do.
SYNOPSIS	<p>This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.</p> <p>The following special characters are used in this section:</p> <p>[] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.</p> <p>. . . Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, 'filename . . . '.</p> <p> Separator. Only one of the arguments separated by this character can be specified at time.</p> <p>{ } Braces. The options and/or arguments enclosed within braces are</p>

interdependent, such that everything enclosed must be treated as a unit.

FEATURES

This section provides the list of features which offer an interface. An API may be associated with one or more system features. The interface will be available if one of the associated features has been configured.

DESCRIPTION

This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES.. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.

OPTIONS

This lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

OPERANDS

This section lists the command operands and describes how they affect the actions of the command.

OUTPUT

This section describes the output - standard output, standard error, or output files - generated by the command.

RETURN VALUES

If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or -1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.

ERRORS

On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

USAGE	<p>This section is provided as a guidance on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality:</p> <p>Commands Modifiers Variables Expressions Input Grammar</p>
EXAMPLES	<p>This section provides examples of usage or of how to use a command or function. Wherever possible, a complete example including command line entry and machine response is shown. Whenever an example is given, the prompt is shown as <code>example%</code> or if the user must be superuser, <code>example#</code>. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.</p>
ENVIRONMENT VARIABLES	<p>This section lists any environment variables that the command or function affects, followed by a brief description of the effect.</p>
EXIT STATUS	<p>This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion and values other than zero for various error conditions.</p>
FILES	<p>This section lists all filenames referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.</p>
SEE ALSO	<p>This section lists references to other man pages, in-house documentation and outside publications.</p>
DIAGNOSTICS	<p>This section lists diagnostic messages with a brief explanation of the condition causing the error.</p>
WARNINGS	<p>This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.</p>
NOTES	<p>This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.</p>

BUGS

This section describes known bugs and wherever possible, suggests workarounds.

Driver Implementations

NAME	amd29xxx – amd29xxx compatible flash driver								
SYNOPSIS	<p>drv/src/flash/amd29xxx/amd29xxx.h - driver structures and constants</p> <p>drv/src/flash/amd29xxx/amd29xxx.c - driver code</p> <p>drv/src/flash/amd29xxx/amd29xxxProp.h - driver specific properties</p>								
FEATURES	DRV								
DESCRIPTION	Implements flash driver interface for amd290 family chips.								
EXTENDED DESCRIPTION	<p>The amd29xxx flash driver implements the 'flash' device driver interface. The driver works on the amd29xxx family including amd29010, amd29040, amd29080, amd29016, amd29032 8-bit wide uniform sector and other compatible chips. Driver presumes 4x8-bit board mounting and uses 32 bit data access.</p> <p>The driver uses the common bus driver interface provided by a parent bus driver. Thus, the driver is generic and may be applied to any bus providing the same type of interface.</p> <p>The driver does not provide the <code>drv_probe()</code> entry. So, the amd29xxx driver does not enumerate the bus, detect an amd29xxx device or create an associated device node. When the amd29xxx driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The driver does not provide the <code>drv_bind()</code> routine. In other words, the driver does not support dynamic driver-to-device binding. When the amd29xxx driver is used, the driver should be explicitly bound to the device using the <code>PROP_DRIVER</code> property. It is assumed that the device-to-driver binding is done either by a device node creator or by a separate binder driver. Such a driver-to-device binder could be developed for this particular bus architecture.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p> <p>The driver does not support hot-plug bus events.</p> <p>The Table below summarizes characteristics of the amd29xxx flash driver.</p> <table> <tr> <td>driver name:</td><td>"sun:bus-amd29xxx-flash"</td></tr> <tr> <td>hardware:</td><td>amd29xxx compatible flash chip</td></tr> <tr> <td>exported interface:</td><td>"flash" (FLASH_CLASS)</td></tr> <tr> <td>exported interface version:</td><td>0 (FLASH_VERSION_INITIAL)</td></tr> </table>	driver name:	"sun:bus-amd29xxx-flash"	hardware:	amd29xxx compatible flash chip	exported interface:	"flash" (FLASH_CLASS)	exported interface version:	0 (FLASH_VERSION_INITIAL)
driver name:	"sun:bus-amd29xxx-flash"								
hardware:	amd29xxx compatible flash chip								
exported interface:	"flash" (FLASH_CLASS)								
exported interface version:	0 (FLASH_VERSION_INITIAL)								

imported parent interface:	"bus" (BUS_CLASS)
minimal parent interface version:	0 (BUS_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	supported
system (emergency) shut-down:	not supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	not supported

There is only one mandatory property used by the amd29xxx driver.

Name	Alias	Type
"mem-rgn"	BUS_PROP_MEM_RGN	<bus class specific>

The BUS_PROP_MEM_RGN property specifies the base address of the media and is used by the driver as a parameter for the `bus->mem_map()` call. The property value is bus class specific.

The upper layer should use optional properties specified in the Table below to map locked flash regions. The presence of each property gives information to the driver client about whether the locked region is writable, executable in place and cacheable.

Name	Alias	Type
"region-write"	FLASH_PROP_RGN_WRITE	<flash class specific>
"region-exec"	FLASH_PROP_RGN_EXEC	<flash class specific>
"region-cache"	FLASH_PROP_RGN_CACHE	<flash class specific>

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`bus(9DDI)`, `flash(9DDI)`

NAME	benchns16550 – ns16x50 device driver															
SYNOPSIS	<div>ddi/bench/bench.h</div> <div>drv/src/bench/ns16550/bench_ns16550.c</div> <div>drv/src/bench/ns16550/benchns16550Prop.h</div>															
FEATURES	DRV															
DESCRIPTION	<p>The ns16550 device exports the bench device driver interface. This driver uses the bus driver interface provided by its parent bus driver.</p> <p>The benchns16550 driver does not provide the <code>drv_probe()</code> routine. In other words, the benchns16550 driver does not enumerate the bus, nor does it detect a benchns16550 device or create an associated device node. When the benchns16550 driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The driver does not provide the <code>drv_bind()</code> routine. In other words, the driver does not support dynamic driver-to-device binding. When the benchns16550 driver is used, the driver should be explicitly bound to the device using the <code>PROP_DRIVER</code> property. It is assumed that the device-to-driver binding is done either by a device node creator or by a separate binder driver. Such a driver-to-device binder could be developed for this particular bus architecture.</p> <p>The driver does not implement the <code>drv_unload()</code>. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>This driver can be used to measure the interrupt latency corresponding to its level in the device hierarchy on a given target board.</p> <table><tr><td>driver name:</td><td colspan="2">“sun:bus-ns16550-bench”</td></tr><tr><td>hardware:</td><td colspan="2">“NS16x50 compatible UART chip”</td></tr><tr><td>exported interface:</td><td>“bench”</td><td>(BENCH_CLASS)</td></tr><tr><td>exported interface version:</td><td>0</td><td>(BENCH_VERSION_INITIAL)</td></tr><tr><td>imported parent interface:</td><td>“bus”</td><td>(BUS_CLASS)</td></tr></table>	driver name:	“sun:bus-ns16550-bench”		hardware:	“NS16x50 compatible UART chip”		exported interface:	“bench”	(BENCH_CLASS)	exported interface version:	0	(BENCH_VERSION_INITIAL)	imported parent interface:	“bus”	(BUS_CLASS)
driver name:	“sun:bus-ns16550-bench”															
hardware:	“NS16x50 compatible UART chip”															
exported interface:	“bench”	(BENCH_CLASS)														
exported interface version:	0	(BENCH_VERSION_INITIAL)														
imported parent interface:	“bus”	(BUS_CLASS)														

The Table below lists device node properties used by the ns16550 bench driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

name	alias	type	m/o	default value
"io-regs"	BUS_PROP_IO_REGS	<bus class specific>	m	
"intr"	BUS_PROP_INTR	<bus class specific>	m	
"clock-freq"	PROP_CLOCK_FREQ	PropClockFreq	o	1843200

The BUS_PROP_IO_REGS property specifies the ns16x50 I/O registers range. The property value is bus class specific.

The BUS_PROP_INTR property specifies the ns16x50 interrupt source. The property value is bus class specific.

The PROP_CLOCK_FREQ property specifies the input clock frequency in Hz.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

bus(9DDI), uart(9DDI)

NAME	cheerio – Sun cheerio 10/100Mbps Ethernet device driver						
SYNOPSIS	<p>drv/src/net/ether/cheerio.h - cheerio hardware related constants</p> <p>drv/src/net/ether/cheerio.c - driver code</p> <p>drv/src/net/ether/cheeriprop.h - driver specific properties</p>						
FEATURES	DRV						
DESCRIPTION	<p>The cheerio Ethernet driver implements the ether device driver interface.</p> <p>The driver uses the pci bus driver interface provided by a pci parent bus driver. Thus, the driver may be applied to any pci bus implementing this interface.</p> <p>The cheerio driver does not provide the <code>drv_probe()</code> routine. In other words, the cheerio driver does not enumerate the pci bus, nor does it detect a cheerio device or create an associated device node. When the cheerio driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate pci bus enumerator driver.</p> <p>The cheerio driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties and thus, identifies a cheerio compatible device. The routine checks for the pci vendor and device identifiers matching a cheerio compatible device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the cheerio driver name. The parent bus driver uses such a property to determine the name of drivers servicing the device. Via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver, asking it to invoke the <code>drv_int()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if the <code>PROP_DRIVER</code> property is already present in the device node, as the <code>drv_bind()</code> routine will not override existing driver-to-driver binding.</p> <p>The driver does not implement the <code>drv_unload()</code> entry. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>The driver supports the <code>DKI_SYS_SHUTDOWN</code> event specified by the common DKI interface.</p> <p>The table below summarizes characteristics of the cheerio driver.</p> <table> <tr> <td>driver name:</td><td>"sun:pci-cheerio-ether"</td></tr> <tr> <td>hardware:</td><td>Sun cheerio 10/100Mbps Ethernet</td></tr> <tr> <td>exported interface:</td><td>"ether" (ETHER_CLASS)</td></tr> </table>	driver name:	"sun:pci-cheerio-ether"	hardware:	Sun cheerio 10/100Mbps Ethernet	exported interface:	"ether" (ETHER_CLASS)
driver name:	"sun:pci-cheerio-ether"						
hardware:	Sun cheerio 10/100Mbps Ethernet						
exported interface:	"ether" (ETHER_CLASS)						

driver name:	"sun:pci-cheerio-ether"
exported interface version:	0 (ETHER_VERSION_INITIAL)
imported parent interface:	"pci" (PCI_CLASS)
minimal parent interface version:	0 (PCI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

The table below lists device node properties used by the `cheerio` driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"dev-num"	PCI_PROP_DEV_NUM	PciPropDevNum	m	
"func-num"	PCI_PROP_FUNC_NUM	PciPropFuncNum	m	
"io-regs"	PCI_PROP_IO_REGS	PciPropIoRegs	m	
"intr"	PCI_PROP_INTR	PciPropIntr	m	
"dma-burst"	PCI_PROP_DMA_BURST	PciPropDmaBurst	o	32
"ether-addr"	ETHER_PROP_ADDR	EtherPropAddr	m	
"tx-frames"	CHEERIO_PROP_ADDR	CheerioPropTxFrames	o	32
"rx-frames"	CHEERIO_PROP_RX_FRAMES	CheerioPropRxFrames	o	64

The `PCI_PROP_VEND_ID` property specifies the Sun pci vendor identifier (must be 0x108e).

The `PCI_PROP_DEV_ID` property specifies the cheerio pci device identifier (must be 0x1001).

The `PCI_PROP_DEV_NUM` property specifies the pci device number of the device.

The `PCI_PROP_FUNC_NUM` property specifies the pci function number of the device.

The `PCI_PROP_IO_REGS` property specifies the cheerio address space within pci memory space.

The `PCI_PROP_INTR` property specifies the interrupt value assigned to the cheerio device node.

The `PCI_PROP_DMA_BURST` property specifies the dma burst sizes.

The `ETHER_PROP_ADDR` property specifies the cheerio Ethernet address.

The `CHEERIO_PROP_TX_FRAMES` specifies the maximum number of transmit frames.

The `CHEERIO_PROP_RX_FRAMES` specifies the maximum number of receive frames.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`ether(9DDI)`, `pci(9DDI)`

NAME	dec2115x – dec2115x pci-to-pci bridges family, pci bus driver
SYNOPSIS	<div>drv/src/pci/dec2115c/dec2115x.h - dec2115c hardware related constants</div> <div>drv/src/pci/dec2115x/dec2115x.c - driver code</div> <div>drv/src/pci/dec2115x/dec2115xprop.h - driver specific properties</div>
FEATURES	DRV
DESCRIPTION	<p>The dec2115x pci-to-pci bridge driver implements:</p> <ul style="list-style-type: none"> ■ The common bus driver interface. ■ The pci bus driver interface <p>The driver works on an Intel 2115x (21150, 21152, 21153 or 21154) pci-to-pci bus bridge.</p> <p>The driver uses the pci bus driver interface provided by a parent pci bus driver. Thus, the driver may be applied to any bus driver implementing the pci bus interface.</p> <p>The dec2115x driver does not provide the <code>drv_probe()</code> routine. In other words, the dec2115x driver does not enumerate the pci bus, nor does it detect a dec2115x device or create an associated device node. When the dec2115x driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate pci bus enumerator driver.</p> <p>The dec2115x driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties and thus, identifies a dec2115x compatible device. The routine checks for the pci vendor and device identifiers matching a dec2115x compatible device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the dec2115x driver name. The parent bus driver uses such a property to determine the name of drivers servicing the device. Via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver, asking it to invoke the <code>drv_int()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if the <code>PROP_DRIVER</code> property is already present in the device node, as the <code>drv_bind()</code> routine will not override existing driver-to-device binding.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports the driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p>

The driver supports all bus events specified by the pci bus driver interface. As a consequence, the driver may be used with a hot-pluggable pci bus (for example, CompactPCI).

The Table below summarizes characteristics of the dec2115x pci bus driver.

driver name:	"sun:pci-dec2115x-(bus, pci)"
hardware:	Intel 2115x pci-to-pci bridge
exported interface:	"bus", "pci" (BUS_CLASS, PCI_CLASS)
exported interface version:	0 (PCI_VERSION_INITIAL)
imported parent interface:	"pci" (PCI_CLASS)
minimal parent interface version:	0 (PCI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver-to-device binding	supported (on a node name basis)
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	supported

The table below lists device node properties used by the dec2115x driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"dev-num"	PCI_PROP_DEV_NUM	PciPropDevNum	m	
"func-num"	PCI_PROP_FUNC_NUM	PciPropFuncNum	m	
"bus-num"	PCI_PROP_BUS_NUM	PciPropBusNum	m	
"sub-bus-num"	PCI_PROP_SUB_BUS_NUM	PciPropSubBusNum	m	
"io-regs"	PCI_PROP_IO_REGS	PciPropIoRegs	o	—
"mem-rgn"	PCI_PROP_MEM_RGN	Pci64PropMemRgn	o	—

Name	Alias	Type	m/o	Default value
"mem-rgn-64"	PCI64_PROP_MEM_RGN	Pci64PropMemRgn	o	—
"intr-conf"	D2115X_PROP_INTR_CONF	D2115xPropIntrConf	o	see text below
"conf"	D2115X_PROP_CONF	D2115xPropConf	o	see text below

The `PCI_PROP_VEND_ID` property specifies the dec vendor identifier (must be 0x1011).

The `PCI_PROP_DEV_ID` property specifies the dec2115x device identifier:

- 0x0022 for dec21150
- 0x0024 for dec21152
- 0x0025 for dec21153
- 0x0026 for dec21154

The `PCI_PROP_DEV_NUM`, and `PCI_PROP_FUNC_NUM` properties specify the configuration space address of the device. This address is the device and function numbers to which the device is connected.

The `PCI_PROP_BUS_NUM` property specifies the secondary bus number.

The `PCI_PROP_SUB_BUS_NUM` property specifies the subordinate bus number.

The dec2115x PCI-to-PCI bridge is a transparent bridge. The `PCI_PROP_IO_REGS`, `PCI_PROP_MEM_RGN` and `PCI64_PROP_MEM_RGN` properties configure the transaction forwarding mechanism (between primary and secondary busses) implemented by the dec2115x bridge.

The `PCI_PROP_IO_REGS` property specifies the programmed and memory-mapped I/O windows assigned for devices residing on the secondary bus. In general, the `PCI_PROP_IO_REGS` property specifies an array which consists of up to two elements (`PciPropIoRegs`). One element specifies a window for programmed I/O accesses (`PCI_PIO_SPACE`) and the other element specifies a window for memory mapped I/O accesses (`PCI_MEM_SPACE`). If one of the windows is not defined (that is, the `PCI_PROP_IO_REGS` property is a single `PciPropIoRegs` structure), the bus bridge will not forward transactions of a corresponding type from primary to secondary busses. If the `PCI_PROP_IO_REGS` property is not specified at all, the bus bridge will not forward I/O access transactions from primary to secondary busses.

The `PCI_PROP_MEM_RGN` property specifies a window (within 32-bit PCI address space) for pre-fetchable memory access. Transactions issued on the primary bus will be claimed by the bus bridge and forwarded to the secondary bus, if they fall into this address window.

The `PCI64_PROP_MEM_RGN` property specifies a window (within 64-bit PCI address space) for pre-fetchable memory access. Transactions issued on the primary bus will be claimed by the bus bridge and forwarded to the secondary bus, if they fall into this address window.

Note that the `PCI_PROP_MEM_RGN` and `PCI64_PROP_MEM_RGN` properties are mutually exclusive. In other words, only one of them can be specified.

Access transactions issued on the secondary bus which do not fall into the windows specified by the `PCI_PROP_IO_REGS` and `PCI_PROP_MEM_RGN` (or `PCI64_PROP_MEM_RGN`) properties will be forwarded by the bus bridge to the primary bus.

The `D2115X_PROP_INTR_CONF` property specifies the pci interrupts routing on the secondary bus. The property value has the `D2115xPropIntrConf` type. `D2115xPropIntrConf` is a two dimensional matrix mapping secondary bus pci interrupts to the primary bus pci interrupts depending on the device number. In other words, for each secondary interrupt (`PCI_INTR_A/B/C/D`) and device number (0..31) pair, the matrix element specifies the primary bus interrupt (`PCI_INTR_A/B/C/D`). If a matrix element is zero, the corresponding interrupt (for the corresponding device number) is not routed to the primary bus. The bus driver will return the `K_EFAIL` error code if an attempt is made to connect a handler to such an interrupt source.

If the `D2115X_PROP_INTR_CONF` property is not specified, a rotated interrupt routing is implemented by the bus driver. There is a one-to-one correspondence between the secondary and primary pci interrupts for the device zero. For the next device, the primary interrupts are rotated and so on. For example, the interrupts routing shown below is applied to the two:

secondary bus interrupt	primary bus interrupt
<code>PCI_INTR_A</code>	<code>PCI_INTR_C</code>
<code>PCI_INTR_B</code>	<code>PCI_INTR_D</code>
<code>PCI_INTR_C</code>	<code>PCI_INTR_A</code>
<code>PCI_INTR_D</code>	<code>PCI_INTR_B</code>

The `D2115X_PROP_CONF` property specifies the configuration of the dec2115x pci-to-pci bridge. The `D2115xPropConf` structure consists of images of the bridge configuration registers. If the `D2115X_PROP_CONF` property is specified, the

driver will program configuration registers according to the images given by the D2115xPropConf structure. Otherwise, the default values will be used. The Table below shows the D2115xPropConf fields and the default values used for the bridge configuration when the D2115X_PROP_CONF property is not specified.

type	name	default	addr	description
uint8_f	cline_size	00	0c	cache line size
uint8_f	plat_timer	00	0d	primary latency timer
uint8_f	slat_timer	00	1b	secondary latency timer
uint8_f	chip_ctl	00	40	chip control
uint8_f	gpio_cfg	00	66	GPIO output enable (*)
uint8_f	serr_dis	00	64	primary SERR# event disable
uint16_f	bridge_ctl	0023	3e	bridge control
uint16_f	arbiter_ctl	0200	43	arbiter control
uint16_f	sclock_ctl	0000	68	secondary clock output control

(*) the gpio_cfg field is ignored for the dec21152 bridge.

The following bits (defined by dec2115x.h) can be set in the chip_ctl field:

- D2115X_CTRL_LINS — live insertion mode
- TRL_NPREF — secondary bus memory pre-fetch disable
- D2115X_CTRL_WDIS — write disconnect on cache line boundary

The following bits (defined by dec2115x.h) can be set in the serr_dis field:

- D2115X_SED_PW_PE — posted write parity error
- D2115X_SED_PW_ND - posted write non delivery
- D2115X_SED_PW_TA - posted write target abort
- D2115X_SED_PW_MA - posted write master abort
- D2115X_SED_DW_ND - delayed write non delivery
- D2115X_SED_DR_ND - delayed read no data

- D2115X_SED_DT_MA - delayed transaction MA

The following bits (defined by pci.h and dec2115.h) can be set in the bridge_ctl field:

- PCI_PCI_CTL_PARERR_RESP - parity error response
- PCI_PCI_CTL_SYSERR - SERR# forwarding
- PCI_PCI_CTL_MABORT_MODE - master abort mode
- D2115X_PCI_CTL_PMTO - primary master time-out
- D2115X_PCI_CTL_SMT0 - secondary master time-out
- D2115X_PCI_CTL_MTO_SERR_EN - master time-out SERR# enable

See the Intel 2115x PCI-to-PCI Bridge Datasheet for programming the bus bridge configuration registers (<http://www.intel.com>).

RESTRICTIONS

Current version of the dec2115x driver does not support the dynamic assignment of the PCI bus resources.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`bus(9DDI)`, `pci(9DDI)`

NAME	dec21x4x – dec21x4x Ethernet device driver
SYNOPSIS	<p>drv/src/net/ether/dec21x4x/dec21x4x.h - hardware related constants</p> <p>drv/src/net/ether/dec21x4x/dec21x4x.c - driver code</p> <p>drv/src/net/ether/dec21x4x/dec21x4xProp.h - driver specific properties</p>
FEATURES	DRV
DESCRIPTION	Device Driver for dec21x4x Ethernet controllers
EXTENDED DESCRIPTION	<p>The dec21x4x Ethernet driver implements the ether device driver interface.</p> <p>The driver works on dec 21040, 21041, 21140 and 21143 Ethernet controllers.</p> <p>The driver uses the pci bus driver interface provided by a parent bus driver. Thus, the driver may be applied to any pci bus interface.</p> <p>The dec21x4x driver does not provide the <code>drv_probe</code> routine. In other words, the dec21x4x driver does not enumerate the pci bus, nor does it detect a dec21x4x device or create an associated device node. When the dec21x4x driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate pci bus enumerator driver.</p> <p>The dec21x4x driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties and thus, identifies a dec21x4x compatible device. The routine checks for the pci vendor and device identifiers matching a dec21x4x compatible device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the dec21x4x driver name. The parent bus driver uses such a property to determine the name of drivers servicing the device. Via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver, asking it to invoke the <code>drv_int()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if the <code>PROP_DRIVER</code> property is already present in the device node, as the <code>drv_bind()</code> routine will not override existing driver-to-driver binding.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports the driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p> <p>The driver supports all bus events specified by the pci bus driver interface. As a consequence, the driver may be used with a hot-pluggable pci bus (for example, CompactPCI).</p> <p>The table below summarizes the characteristics of the dec21x4x Ethernet driver:</p>

driver name:	"sun:pci-dec21x4x-ether"
hardware:	dec 21040/21041/21140/21143
exported interface:	"ether" (ETHER_CLASS)
exported interface version:	0 (ETHER_VERSION_INITIAL)
imported parent interface:	"pci" (PCI_CLASS)
minimal parent interface version:	0 (PCI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	supported

The table below lists device node properties used by the `dec21x4x` driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"dev-num"	PCI_PROP_DEV_NUM	PciPropDevNum	m	
"func-num"	PCI_PROP_FUNC_NUM	PciPropFuncNum	m	
"intr"	PCI_PROP_INTR	PciPropIntr	m	
"ether-addr"	ETHER_PROP_ADDR	EtherPropAddr	o	SROM content
"srom"	DEC21_PROP_SROM	uint8_f[]	o	see text below
"media"	DEC21_PROP_MEDIA	Dec21PropMedia	o	MEDIA_TP
"tx-desc-nb"	DEC21_PROP_TX_DESC_NB	Dec21PropTxDescNb	o	8
"rx-desc-nb"	DEC21_PROP_RX_DESC_NB	Dec21PropRxDescNb	o	8
"tx-buf-size"	DEC21_PROP_TX_BUF_SIZE	Dec21PropTxBufSize	o	1514 bytes
"rx-buf-size"	DEC21_PROP_RX_BUF_SIZE	Dec21PropRxBufSize	o	1514 bytes

The `PCI_PROP_VEND_ID` property specifies the dec pci vendor identifier (must be 0x1011).

The `PCI_PROP_DEV_ID` property identifies the type of controller within the dec21x4x family:

- must be 0x0002 for dec21040
- must be 0x0014 for dec21041
- must be 0x0009 for dec21140
- must be 0x0019 for dec21143

The `PCI_PROP_DEV_NUM`, and `PCI_PROP_FUNC_NUM` properties specify the configuration space address of the device.

The `PCI_PROP_INTR` property specifies the pci interrupt used by the device.

The `PCI_PROP_IO_REGS` property specifies the controller I/O registers range.

The `ETHER_PROP_ADDR` property specifies the controller Ethernet address. If the property is not present, the driver reads the Ethernet address from the controller SROM, and creates the property in the device node.

The `DEC21_PROP_SROM` property specifies the SROM data. This property is optional. If the property is not present, the driver does not access the controller SROM but, instead, uses the property value as the SROM image. The property value must be a byte stream in the 21x4 Serial ROM format.

The `DEC21_PROP_MEDIA` property specifies the default media type. The controller is configured for the default media in case the driver is not able to detect the media type automatically. This property is optional. If the property is not present, the default media type is the Twister Pair 10MB link (`MEDIA_TP`) .

The `DEC21_PROP_TX_DESC_NB` property specifies the number of descriptors to be allocated in the transmit ring. By default, 8 descriptors are used.

The `DEC21_PROP_RX_DESC_NB` property specifies the number of descriptors to be allocated in the receive ring. By default, 8 descriptors are used.

The `DEC21_PROP_TX_BUF_SIZE` property specifies the size of the DMA buffers to be allocated and bound to each descriptor in the transmit ring. By default, each buffer has a size of 1514 bytes, which is the maximum size of an Ethernet frame.

The `DEC21_PROP_RX_BUF_SIZE` property specifies the size of the DMA buffers to be allocated and bound to each descriptor in the receive ring. By default, each buffer has a size of 1514 bytes, which is the maximum size of an Ethernet frame.

Thus, the total size of DMA memory allocated by the driver will be:

- `dec21_PROP_TX_DESC_NB * dec21_PROP_TX_BUF_SIZE` for transmission
- `dec21_PROP_RX_DESC_NB * dec21_PROP_RX_BUF_SIZE` for reception.

Note that the driver is able to send and receive an Ethernet frame using multiple descriptors in a ring. It is not able to share the memory buffer bound to one descriptor between multiple Ethernet frames. In other words, configuring buffer sizes with big values will lead to poor memory usage, for small Ethernet frames.

The table below lists device node properties added by the `dec21x4x` driver, which may be used by the driver clients:

Name	Alias	Type
"link-throughput"	ETHER_PROP_THROUGHPUT	EtherPropThroughput
"ether-addr"	ETHER_PROP_ADDR	EtherPropAddr

The `ETHER_PROP_THROUGHPUT` property specifies the throughput of the Ethernet link.

The `ETHER_PROP_ADDR` property specifies the controller Ethernet address. Note that the driver adds this property to the device node only if it does not already exist.

Link Detection

The media speed (10/100Mb) and mode (half and full-duplex) is detected automatically by the driver in one of the following ways:

- when an external PHY is present on the board, the auto-negotiation capabilities of the transceiver are used to configure the controller in the best mode possible (as detected by the transceiver)
- when an external PHY is present on the board, the auto-negotiation capabilities of the transceiver are used to configure the controller in the best mode possible (as detected by the transceiver)

The default media type (obtained through the `DEC21_PROP_MEDIA` property) is used if either of the following cases occur:

1. the auto-negotiation (using PHY) fails
2. the manual detection (using the test frame) fails

The `DEC21_PROP_NO_AUTO_DETECT` property disables link auto-detection.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

ether(9DDI), pci(9DDI)

NAME	ebus – Sun pci/isa bridge driver
SYNOPSIS	<p>drv/src/isa/ebus/ebus.h - ebus hardware related constants</p> <p>drv/src/isa/ebus/ebus.c - driver code</p> <p>drv/src/isa/ebus/ebusProp.h - driver specific properties</p>
FEATURES	DRV
DESCRIPTION	<p>The ebus bridge driver implements:</p> <ul style="list-style-type: none"> ■ the common bus driver Interface, and ■ the isa bus driver Interface. <p>The driver uses the pci bus driver interface provided by a parent pci bus driver. Thus, the driver may be applied to any bus driver implementing the pci bus interface.</p> <p>The ebus driver does not provide the <code>drv_probe()</code> routine. In other words, the ebus driver does not enumerate the pci bus, nor does it detect an ebus device or create an associated device node. When the ebus driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate pci bus enumerator driver.</p> <p>The ebus driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties and thus identifies an ebus compatible device. The routine checks for the pci vendor and device identifiers matching an ebus compatible device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the ebus driver name. The parent bus driver uses such a property to determine the name of drivers servicing the device. Via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver, asking it to invoke the <code>drv_int()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if the <code>PROP_DRIVER</code> property is already present in the device node, as the <code>drv_bind()</code> routine will not override existing driver-to-driver binding.</p> <p>The driver does not implement the <code>drv_unload()</code> entry. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>The driver supports the <code>DKI_SYS_SHUTDOWN</code> event specified by the common DKI interface.</p> <p>The EBUS driver uses the RID DI to attach/detach ISA interrupts.</p>

The Table below summarizes characteristics of the ebus driver.

driver name:	"sun:(pci, ric)-ebus-(bus, isa)"
hardware:	Sun ebus pci/isa bridge
exported interface:	"bus" (BUS_CLASS)
exported interface version:	0 (BUS_VERSION_INITIAL)
exported interface:	"isa" (ISA_CLASS)
exported interface version:	0 (ISATHER_VERSION_INITIAL)
imported parent interface:	"pci" (PCI_CLASS)
minimal parent interface version:	0 (PCI_VERSION_INITIAL)
imported parent interface:	"ric" (RIC_CLASS)
minimal parent interface version:	0 (RIC_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

The Table below lists device node properties used by the ebus driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"dev-num"	PCI_PROP_DEV_NUM	PciPropDevNum	m	
"func-num"	PCI_PROP_FUNC_NUM	PciPropFuncNum	m	
"io-regs"	PCI_PROP_IO_REGS	PciPropIoRegs	m	
"ebud-ric-map"	EBUS_PROP_RIC_MAP	EbusPropRicMap	m	

The `PCI_PROP_VEND_ID` property specifies the Sun pci vendor identifier (must be 0x108e).

The `PCI_PROP_DEV_ID` property specifies the ebus pci device identifier (must be 0x1000).

The `PCI_PROP_DEV_NUM` property specifies the pci device number of the device.

The `PCI_PROP_FUNC_NUM` property specifies the pci function number of the device.

For each ISA device, the `EBUS_PROP_RIC_MAP` property specifies a mapping of the ISA interrupt into a RIC interrupt number offset (ranging from 0x0 to 0x3f). The `EBUS_RIC_OFF_INVALID` value can be used if the corresponding offset is not defined. This property allows the EBUS to take into account the various ISA interrupt RIC offset mappings for different platforms.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`ric(9DDI)` `bus(9DDI)`, `pci(9DDI)` `isa(9DDI)`

NAME	el3 – 3Com etherlinkIII Ethernet device driver										
SYNOPSIS	<p>drv/src/net/ether/el3/el3.h - el3 hardware related constants</p> <p>drv/src/net/ether/el3/el3.c - driver code</p> <p>drv/src/net/ether/el3/el3prop.h - driver specific properties</p>										
FEATURES	DRV										
DESCRIPTION	<p>The el3 Ethernet driver implements the ether device driver interface.</p> <p>The driver is based on the common bus driver interface. Thus, the driver may be applied to any bus driver providing this interface. This means that the el3 driver can be used to drive an ISA board such as the 3C509x series, as well as PCMCIA based boards such as the 3C589 series.</p> <p>The el3 driver does not provide the <code>drv_probe()</code> routine. In other words, the el3 driver does not enumerate the bus, nor does it detect an el3 device or create an associated device node. When the el3 driver is used, associated device nodes should be created either statically, by a boot program, or dynamically, by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The driver does not provide the <code>drv_bind()</code> routine. In other words, the driver does not support dynamic driver-to-device binding. When the el3 driver is used, the driver should be explicitly bound to the device using the <code>PROP_DRIVER</code> property. It is assumed that the device-to-driver binding is done either by a device node creator or by a separate binder driver. Such a driver-to-device binder could be developed for this particular bus architecture.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports the driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p> <p>The driver supports all bus events specified by the common bus driver interface. Consequently, the driver may be used with a hot-pluggable bus (for example, PCMCIA).</p> <p>The Table below summarizes characteristics of the el3 Ethernet driver.</p> <table> <tr> <td>driver name:</td><td>"sun:bus-el3-ether"</td></tr> <tr> <td>hardware:</td><td>3Com etherlinkIII Ethernet controller</td></tr> <tr> <td>exported interface:</td><td>"ether" (ETHER_CLASS)</td></tr> <tr> <td>exported interface version:</td><td>0 (ETHER_VERSION_INITIAL)</td></tr> <tr> <td>imported parent interface:</td><td>"bus" (BUS_CLASS)</td></tr> </table>	driver name:	"sun:bus-el3-ether"	hardware:	3Com etherlinkIII Ethernet controller	exported interface:	"ether" (ETHER_CLASS)	exported interface version:	0 (ETHER_VERSION_INITIAL)	imported parent interface:	"bus" (BUS_CLASS)
driver name:	"sun:bus-el3-ether"										
hardware:	3Com etherlinkIII Ethernet controller										
exported interface:	"ether" (ETHER_CLASS)										
exported interface version:	0 (ETHER_VERSION_INITIAL)										
imported parent interface:	"bus" (BUS_CLASS)										

driver name:	"sun:bus-el3-ether"
minimal parent interface version:	0 (BUS_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	supported

The Table below lists the generic common bus device node properties used by the el3 driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"io-regs"	BUS_PROP_IO_REGS	<bus class specific>	m	
"intr"	BUS_PROP_INTR	<bus class specific>	m	
"el3-id-port"	EL3_PROP_ID_PORT	BUS_PROP_IO_REGS	m	
"el3-cfg"	EL3_PROP_CFG13	E13PropCfg	m	
"el3-tx-start-thr"	EL3_TX_START_THRESHOLD	uint16_fo		
"el3-rx-early-thr"	EL3_RX_EARLY_THRESHOLD	uint16_fo		

The BUS_PROP_IO_REGS property specifies the el3 I/O registers range. Its value is bus class specific.

The BUS_PROP_INTR property specifies the el3 interrupt to be used. Its value is bus class specific.

The Table below lists the specific device node properties used by the el3 driver.

Name	Alias	Type	m/o	Default value
"el3-id-port"	EL3_PROP_ID_PORT	BUS_PROP_IO_REGS	o	
"el3-cfg"	EL3_PROP_CFG	El3PropCfg	m	
"el3-tx-start-thr"	EL3_TX_START_THRESHOLD	uint16_f	o	2000 (disabled)
"el3-rx-early-thr"	EL3_RX_EARLY_THRESHOLD	uint16_f	o	2000 (disabled)

The `EL3_PROP_ID_PORT` property specifies the el3 ID Port I/O register. It is used by the el3 driver to activate boards on an ISA bus. Therefore this property is only used to activate boards connected to an ISA bus. Its value is bus class specific. The el3 driver will use the common bus I/O operations on this property. The default value for this property should suit nearly all configurations, and should be modified in case of conflict with other ISA devices.

Note: This property is not used on any busses other than ISA.

The `EL3_PROP_CFG` property specifies the configuration used by the device. The `El3PropCfg` type is defined as:

```
typedef struct {
    El3WinReg w0o6; /* Window 0 Offset 6 config */
    El3WinReg w0o8; /* Window 0 Offset 8 config */
    char boardPort; /* Output Port */
    char activation; /* Activation Method */
} El3PropCfg;
```

The `w0o6` and `w0o8` fields should not be modified by the user. They contain values used internally by the driver. These values are automatically computed from the `BUS_PROP_IO_REGS` and `BUS_PROP_INTR` properties mentioned above.

The `boardPort` field specifies the output port (media) to be used and must be set to one of the following values:

- `EL3_TP_PORT`
- `EL3_AUI_PORT`
- `EL3_BNC_PORT`

The `activation` field specifies the activation mechanism which should be used. Its value depends on the parent bus and must be set to one of the following values:

- `EL3_ISA_ACTIVATION` for ISA based boards,

■ **EL3_PCMCIA_ACTIVATION** for PCMCIA based boards,

The **EL3_TX_START_THRESHOLD** property specifies the threshold used by the board to start transmission. Its value is the number of bytes which should be present in the TX FIFO to start transmission. By default, this feature is disabled (value > 1792) and transmission will begin once the complete frame has been copied to the TX FIFO.

The **EL3_RX_EARLY_THRESHOLD** property specifies the threshold used by the board to notify the driver that a frame reception has begun. Its value is the number of bytes. Once this amount of bytes is present in the RX FIFO, the board will generate an interrupt to notify the driver. By default, this feature is disabled (value > 1792) and the board will notify once the complete frame has been received.

The Table below lists the device node properties dynamically added by the el3 driver, which may be used by the driver clients.

Name	Alias	Type
"link-throughput"	ETHER_PROP_THROUGHPUT	EtherPropThroughput
"ether-addr"	ETHER_PROP_ADDR	EtherPropAddr

The **ETHER_PROP_THROUGHPUT** specifies the throughput of the Ethernet link. The el3 driver adds this property to the device node and initializes it with the throughput of the Ethernet link to which the device is connected.

The **ETHER_PROP_ADDR** property specifies the device Ethernet address. The el3 driver adds this property to the device node and initializes it with the Ethernet address read in the board EEPROM.

Note: These properties will be added by the el3 driver if, and only if, they do not exist in the device tree at initialization time.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`ether(9DDI)`, `isa(9DDI)`, `bus(9DDI)`

NAME	epic100 – epic100 pci Ethernet device driver								
SYNOPSIS	<p>drv/src/net/ether/epic100/epic100.h - EPIC100 hardware constants</p> <p>drv/src/net/ether/epic100/epic100.c - driver code</p> <p>drv/src/net/ether/epic100/epic100Prop.h - driver specific properties</p>								
FEATURES	DRV								
DESCRIPTION	Implements SMC epic100 Ethernet controller.								
EXTENDED DESCRIPTION	<p>The epic100 Ethernet driver implements the ether device driver interface.</p> <p>The driver uses the pci bus driver interface provided by a parent bus driver. Thus, the driver may be applied to any pci bus interface.</p> <p>The epic100 driver does not provide the <code>drv_probe()</code> routine. In other words, the epic100 driver does not enumerate the pci bus, nor does it detect an epic100 device or create an associated device node. When the epic100 driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate pci bus enumerator driver.</p> <p>The epic100 driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties and thus, identifies an epic100 compatible device. The routine checks for the pci vendor and device identifiers matching an epic100 compatible device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the epic100 driver name. The parent bus driver uses such a property to determine the name of drivers servicing the device. Via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver, asking it to invoke the <code>drv_int()</code> routine on that device. The <code>drv_bind()</code> routine does nothing if the <code>PROP_DRIVER</code> property is already present in the device node. In other words, the <code>drv_bind</code> routine will not override existing driver-to-device binding.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports the driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p> <p>The table below summarizes characteristics of the epic100 Ethernet driver:</p> <table> <tr> <td>driver name:</td><td>sun:pci-epic100-ether"</td></tr> <tr> <td>hardware:</td><td>EPIC100 Ethernet controller</td></tr> <tr> <td>exported interface:</td><td>"ether" (ETHER_CLASS)</td></tr> <tr> <td>exported interface version:</td><td>0 (ETHER_VERSION_INITIAL)</td></tr> </table>	driver name:	sun:pci-epic100-ether"	hardware:	EPIC100 Ethernet controller	exported interface:	"ether" (ETHER_CLASS)	exported interface version:	0 (ETHER_VERSION_INITIAL)
driver name:	sun:pci-epic100-ether"								
hardware:	EPIC100 Ethernet controller								
exported interface:	"ether" (ETHER_CLASS)								
exported interface version:	0 (ETHER_VERSION_INITIAL)								

imported parent interface:	"pci" (PCI_CLASS)
minimal parent interface version:	0 (PCI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	supported

The table below lists device node properties used by the epic100 driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the "default value" column shows the default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"bus-num"	PCI_PROP_BUS_NUM	PciPropBusNum	m	
"dev-num"	PCI_PROP_DEV_NUM	PciPropDevNum	m	
"func-num"	PCI_PROP_FUNC_NUM	PciPropFuncNum	m	
"intr"	PCI_PROP_INTR	PciPropIntr	m	
"tx-desc-nb"	EPIC100_PROP_TX_DESC_NB	uint32_f	o	4
"rx-desc-nb"	EPIC100_PROP_RX_DESC_NB	uint32_f	o	4
"tx-buf-size"	EPIC100_PROP_TX_BUF_SIZE	uint32_f	o	1514 bytes
"rx-buf-size"	EPIC100_PROP_RX_BUF_SIZE	uint32_f	o	1514 bytes

The PCI_PROP_VEND_ID property specifies the SMC PCI vendor identifier (must be 0x10B8).

The PCI_PROP_DEV_ID property specifies the EPIC100 device identifier (must be 0x0005).

The `PCI_PROP_BUS_NUM`, `PCI_PROP_DEV_NUM`, and `PCI_PROP_FUNC_NUM` properties specify the configuration space address of the device. That is the PCI bus, device and function numbers to which the device is connected.

The `PCI_PROP_INTR` property specifies the PCI interrupt used by the device.

The `PCI_PROP_IO_REGS` property specifies the EPIC100 I/O registers range.

The `EPIC100_PROP_TX_DESC_NB` specifies the number of descriptors to allocate in the transmit DMA ring. By default, 4 descriptors are used.

The `EPIC100_PROP_RX_DESC_NB` specifies the number of descriptors to allocate in the receive DMA ring. By default, 4 descriptors are used.

The `EPIC100_PROP_TX_BUF_SIZE` specifies the size of the DMA buffers to allocate and bind to each descriptor in the transmit ring. By default, each buffer has a size of 1514 bytes, which is the maximum size of an Ethernet frame.

The `EPIC100_PROP_RX_BUF_SIZE` specifies the size of the DMA buffers to allocate and bind to each descriptor in the receive ring. By default, each buffer has a size of 1514 bytes, which is the maximum size of an Ethernet frame.

Thus, the total size of DMA memory allocated by the driver will be:

`EPIC100_PROP_TX_DESC_NB * EPIC100_PROP_TX_BUF_SIZE` for transmission, and

`EPIC100_PROP_RX_DESC_NB * EPIC100_PROP_RX_BUF_SIZE` for reception.

Note that the driver is able to send and receive an Ethernet frame using multiple descriptors in a ring. It is not able to share the memory buffer bounded to one descriptor between multiple Ethernet frames. In other words, configuring buffer sizes with big values will lead to poor memory usage, with small Ethernet frames.

The table below lists device node properties added by the Epic100 driver, which may be used by the driver clients.

Name	Alias	Type
link-throughput	ETHER_PROP_THROUGHPUT	EtherPropThroughput
"ether-addr"	ETHER_PROP_ADDR	EtherPropAddr

The ETHER_PROP_THROUGHPUT specifies the throughput of the Ethernet link. The Epic100 driver adds this property to the device and initializes it with the throughput of the Ethernet link to which the device is connected.

The ETHER_PROP_ADDR property specifies the device Ethernet address. The Epic100 driver adds this property to the device and initializes it with the Ethernet address which is in the controller ROM.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`ether(9DDI)`, `pci(9DDI)`

NAME	<code>fccEther</code> – quicc fcc controller Ethernet device driver.						
SYNOPSIS	<p><code>drv/src/net/ether/fcc/fccEther.c</code> - driver code</p> <p><code>drv/src/net/ether/fcc/fccEtherProp.h</code> - driver specific properties</p>						
FEATURES	DRV						
DESCRIPTION	<p>The <code>fccEther</code> Ethernet driver implements the ether device driver interface.</p> <p>The driver works on a quicc fast communication controller (FCC) connected to an MII interface. The driver uses the quicc bus driver interface provided by a parent bus driver.</p> <p>The <code>fccEther</code> driver does not provide the <code>drv_probe()</code> routine. In other words, the <code>fccEther</code> driver does not enumerate the bus, nor does it detect an <code>fccEther</code> device or create an associated device node. When the <code>fccEther</code> driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The <code>fccEther</code> driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties in order to recognize an <code>fccEther</code> compatible device. The routine checks for a <code>QUICC_PROP_CHANNEL</code> property matching an fcc device in Ethernet mode. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node attaching a <code>PROP_DRIVER</code> property to it. The property value specifies the <code>fccEther</code> driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. In other words, via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver asking it to invoke the <code>drv_init()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if a <code>PROP_DRIVER</code> property is already present in the device node. In other words, the <code>drv_bind()</code> routine does not override existing driver-to-device binding.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports the driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p> <p>The driver supports the <code>QUICC_SYS_SHUTDOWN</code> and <code>QUICC_DEV_SHUTDOWN</code> bus events specified by the quicc bus driver interface.</p> <p>The Table below summarizes characteristics of the <code>fccEther</code> Ethernet driver.</p> <table> <tr> <td>driver name:</td><td>"sun:quicc-fcc-ether"</td></tr> <tr> <td>hardware:</td><td>Quicc/fcc connected to MII</td></tr> <tr> <td>exported interface:</td><td>"ether" (ETHER_CLASS)</td></tr> </table>	driver name:	"sun:quicc-fcc-ether"	hardware:	Quicc/fcc connected to MII	exported interface:	"ether" (ETHER_CLASS)
driver name:	"sun:quicc-fcc-ether"						
hardware:	Quicc/fcc connected to MII						
exported interface:	"ether" (ETHER_CLASS)						

exported interface version:	0 (ETHER_VERSION_INITIAL)
imported parent interface:	"quicc" (QUICC_CLASS)
minimal parent interface version:	0 (QUICC_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver-to-device binding:	supported (on channel basis)
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	not supported

The table below lists device node properties used by the fccEther driver.

Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default Value
"mem-rgn"	QUICC_PROP_MEM_RGN	QuiccPropMemRgn	m	
"intr"	QUICC_PROP_INTR	QuiccPropIntr	m	
"channel"	QUICC_PROP_CHANNEL	QuiccPropChannel	m	
"ether-addr"	ETHER_PROP_ADDR	EtherPropAddr	o	ENVIRONMENT
"phy-addr"	MII_PROP_PHY_ADDR	MiiPropPhyAddr	o	0
"tx-desc-nb"	FCCETHER_PROP_TX_DESC_NB	FccEtherPropDescNb	o	8
"rx-desc-nb"	FCCETHER_PROP_RX_DESC_NB	FccEtherPropDescNb	o	8
"tx-buf-size"	FCCETHER_PROP_TX_BUF_SZ	FccEtherPropBufSz	o	1514 bytes
"rx-buf-size"	FCCETHER_PROP_RX_BUF_SZ	FccEtherPropBufSz	o	1514 bytes

The QUICC_PROP_MEM_RGN property specifies the quicc bus memory mapped regions used by the driver:

- index FCCETHER_REG is used to map fcc registers

- index FCCETHER_PARAM is used to map fcc parameters

The QUICC_PROP_INTR property specifies the interrupt used by the device.

The QUICC_PROP_CHANNEL property specifies the channel used by the device.

The ETHER_PROP_ADDR property specifies the Ethernet address. If the property is not present, the driver tries to get the Ethernet address from the "ETHER_ADDR" environment variable, and creates the property in the device node.

The MII_PROP_PHY_ADDR property specifies the address on the MII bus of the transceiver (PHY) to which the FCC device is connected. By default, PHY address 0 is used.

The FCCETHER_PROP_TX_DESC_NB specifies the number of descriptors to be allocated in the transmit ring. By default, 8 descriptors are used.

The FCCETHER_PROP_RX_DESC_NB specifies the number of descriptors to be allocated in the receive ring. By default, 8 descriptors are used.

The FCCETHER_PROP_TX_BUF_SZ specifies the size of the DMA buffers to be allocated and bound to each descriptor in the transmit ring. By default, each buffer has a size of 1514 bytes, which is the maximum size of an Ethernet frame.

The FCCETHER_PROP_RX_BUF_SZ specifies the size of the DMA buffers to be allocated and bound to each descriptor in the receive ring. By default, each buffer has a size of 1514 bytes, which is the maximum size of an Ethernet frame. Thus, the total size of DMA memory allocated by the driver will be:

- $\text{FCCETHER_PROP_TX_DESC_NB} * \text{FCCETHER_PROP_TX_BUF_SZ}$ for transmission, and
- $\text{FCCETHER_PROP_RX_DESC_NB} * \text{FCCETHER_PROP_RX_BUF_SZ}$ for reception

Note that the driver is capable of sending and receiving an Ethernet frame using multiple descriptors in a ring. But it is not able to share the memory buffer, bound to one descriptor, between multiple Ethernet frames. In other words, configuring buffer sizes with big values will lead to poor memory usage, with small Ethernet frames.

The table below lists device node properties added by the fccEther driver, which may be used by the driver clients.

Name	Alias	Type
"link-throughput"	ETHER_PROP_THROUGHPUT	EtherPropThroughput
"ether-addr"	ETHER_PROP_ADDR	EtherPropAddr
"best-out-size"	ETHER_PROP_BEST_OUTSIZE	EtherPropBestOutsize
"best-in-size"	ETHER_PROP_BEST_INSIZE	EtherPropBestInsize

The ETHER_PROP_THROUGHPUT property specifies the throughput of the Ethernet link.

The ETHER_PROP_ADDR property specifies the Ethernet address. Note that the driver adds this property to the device node only if it does not already exist.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`ether(9DDI)`, `quicc(9DDI)`, `mii(9DDI)`

NAME	i8254 – Intel i8254 timer device driver												
SYNOPSIS	<p>drv/src/timer/i8254/i8254.c - driver code</p> <p>drv/src/timer/i8254/i8254Prop.h - driver specific properties</p>												
FEATURES	DRV												
DESCRIPTION	<p>The i8254 driver implements the timer device driver interface.</p> <p>The driver uses the common bus driver interface provided by a parent bus driver. Thus, the driver is generic and may be applied to any bus providing this type of interface.</p> <p>The i8254 driver does not provide the <code>drv_probe()</code> routine. In other words, the i8254 driver does not enumerate the bus, nor does it detect an i8254 device or create an associated device node. When the i8254 driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The driver does not provide the <code>drv_bind()</code> routine. In other words, the driver does not support dynamic driver-to-device binding. When the i8254 driver is used, the driver should be explicitly bound to the device using the <code>PROP_DRIVER</code> property. It is assumed that the device-to-driver binding is done either by a device node creator or by a separate binder driver. Such a driver-to-device binder could be developed for this particular bus architecture.</p> <p>The driver does not implement <code>drv_unload()</code>. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>The driver does not support the <code>BUS_DEV_REMOVAL</code> bus event specified by the common bus driver interface. As a consequence, the driver may not be used with a hot-pluggable bus (for example, PCMCIA).</p> <p>The Table below summarizes the characteristics of the i8254 driver:</p> <table> <tr> <td>driver name:</td><td>"sun:bus-i8254-timer"</td></tr> <tr> <td>hardware:</td><td>Intel i8254 timer chip</td></tr> <tr> <td>exported interfaces:</td><td>"timer" (TIMER_CLASS)</td></tr> <tr> <td>exported interface version:</td><td>0 (TIMER_VERSION_INITIAL)</td></tr> <tr> <td>imported parent interface:</td><td>"bus" (BUS_CLASS)</td></tr> <tr> <td>minimal parent interface version:</td><td>0 (BUS_VERSION_INITIAL)</td></tr> </table>	driver name:	"sun:bus-i8254-timer"	hardware:	Intel i8254 timer chip	exported interfaces:	"timer" (TIMER_CLASS)	exported interface version:	0 (TIMER_VERSION_INITIAL)	imported parent interface:	"bus" (BUS_CLASS)	minimal parent interface version:	0 (BUS_VERSION_INITIAL)
driver name:	"sun:bus-i8254-timer"												
hardware:	Intel i8254 timer chip												
exported interfaces:	"timer" (TIMER_CLASS)												
exported interface version:	0 (TIMER_VERSION_INITIAL)												
imported parent interface:	"bus" (BUS_CLASS)												
minimal parent interface version:	0 (BUS_VERSION_INITIAL)												

device probing (auto-detection):	not supported
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	not supported

The Table below lists device node properties used by the i8254 driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"io-regs"	BUS_PROP_IO_REGS	<bus class specific>	m	
"intr"	BUS_PROP_INTR	<bus class specific>	m	
"timer-freq"	PROP_TIMER_FREQ	<generic>	o	1193180
"timer-conf"	I8254_PROP_CONF	I8254PropConf	m	see text below

The BUS_PROP_IO_REGS property specifies the i8254 I/O registers range. The property value is bus class specific.

The BUS_PROP_INTR property specifies the i8254 interrupt source. The property value is bus class specific.

The PROP_TIMER_FREQ property specifies the i8254 frequency. The property value is board specific.

The I8254_PROP_CONF property specifies the role assigned to each i8254 counter. The I8254PropConf type is defined as:

```
typedef struct {
    CounterConfig  c0;    /* Counter 0 configuration */
    CounterConfig  c1;    /* Counter 1 configuration */
    CounterConfig  c2;    /* Counter 2 configuration */
} El3PropCfg;
```

and the CounterConfig type as:

```
typedef struct {  
    uint8_f    role;  
} CounterConfig;
```

The `role` field can take the following value:

- `SYSTEM_TICK_COUNTER`
- `SYSTEM_SPEAKER_COUNTER`
- `TIMER_COUNTER`
- `RESERVED_COUNTER`

The `SYSTEM_TICK_COUNTER` and `SYSTEM_SPEAKER_COUNTER` must only be assigned to one counter. To configure a counter, as used by the system tick timer, simply set it to `SYSTEM_TICK_COUNTER`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`bus(9DDI)`, `isa(9DDI)`, `timer(9DDI)`

NAME	intro – device drivers				
SYNOPSIS	<pre># include <drv/drv.h> # include <drv/f_drv.h></pre>				
FEATURES	DRV				
DESCRIPTION	Provides device driver services				
EXTENDED DESCRIPTION	<p>The ChorusOS Device Driver Framework provides several drivers. Most of these drivers, unless otherwise noted, are generic (non-platform-specific) drivers, and can be used regardless of platform, as they utilize either Common Bus Driver Interface or Bus Specific (not Platform Specific) services.</p> <p>For further information on the provided drivers please see the appropriate .9drv manpages, listed below.</p>				
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
Interface Stability	Evolving				
SEE ALSO	<p>amd29xxx(9DRV), benchns16550(9DRV), cheerio(9DRV), dec2115x(9DRV), dec21x4x(9DRV), ebus(9DRV), el3(9DRV), epic100(9DRV), fccEther(9DRV), i8254(9DRV), isabios(9DRV), isapci(9DRV), m48txx(9DRV), mc146818(9DRV), ne2000(9DRV), nsl6650(9DRV), pcibios(9DRV), pciconf(9DRV), pcienumo(9DRV), quicc8260(9DRV), quicc8xx(9DRV), raven(9DRV), ric(9DRV), sabre(9DRV), sccEther(9DRV), sccuart(9DRV), simba(9DRV), smc1660(9DRV), smcuart(9DRV), tbDec(9DRV), universe(9DRV), univRemCom(9DRV), vt82c586(9DRV), w83c553(9DRV), z85x30(9DRV)</p>				

NAME	isabios – Intel i386AT generic isa bus driver
SYNOPSIS	<p>drv_f/src/isa/isabios/generic.c - driver code</p> <p>drv_f/src/isa/isabios/genericProp.h - driver specific properties</p>
FEATURES	DRV
DESCRIPTION	<p>The isabios bus driver implements:</p> <ul style="list-style-type: none"> ■ The common bus driver interface, and ■ The isa bus driver interface. <p>The driver works on any Intel i386AT platform.</p> <p>The driver uses the Intel x86 specific dki interface provided by the kernel. Thus, the driver only works with Intel based family products.</p> <p>The isabios driver does not provide the <code>drv_probe()</code> routine. In other words, the isabios driver does not enumerate the bus nor does it detect an isabios device or create an associated device node. When the isabios driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for the particular bus architecture.</p> <p>The isabios driver provides the <code>drv_bind()</code> routine. This routine examines the device node name property (PROP_NODE) in order to recognize an isabios compatible device. The routine checks whether the device node name matches a pre-defined device name. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node attaching a PROP_DRIVER property to the device node. The property value specifies the ric driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. In other words, via the PROP_DRIVER property, the driver gives its name to the parent bus driver asking it to invoke the <code>drv_init()</code> routine on that device. The <code>drv_bind()</code> routine does nothing if a PROP_DRIVER property is already present in the device node. In other words, the <code>drv_bind()</code> routine will not override existing driver-to-device binding.</p> <p>The driver does not implement <code>drv_unload()</code>. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>The driver only supports the DKI_SYS_SHUTDOWN event specified by the common dki interface.</p> <p>The Table below summarizes the characteristics of the isabios isa bus driver:</p>

driver name:	"sun:x86-generic-(bus,isa)"
hardware:	Intel i386AT standard ISA bus
exported interface:	"bus,isa" (BUS_CLASS,ISA_CLASS)
exported interface version:	0 (ISA_VERSION_INITIAL)
imported parent interface:	"x86" (FDKI_CLASS)
minimal parent interface version:	0 (FDKI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

ATTRIBUTES

The isabios driver does not use any specific properties other than its name.
See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`bus(9DDI)`, `isa(9DDI)`

NAME	isapci – Intel i386AT generic pci/isa bridge, isa bus driver
SYNOPSIS	<p>x86/drv_f/src/isa/isapci/generic.c - driver code</p> <p>x86/drv_f/src/isa/isapci/genericProp.h - driver specific properties</p>
FEATURES	DRV
DESCRIPTION	<p>The isapci bus driver implements:</p> <ul style="list-style-type: none"> ■ The common bus driver interface, and ■ The ISA bus driver interface. <p>The driver works on any Intel i386AT platform equipped with a pci bus.</p> <p>The driver uses the pci bus driver interface provided by a parent pci bus driver. Thus, the driver may be applied to any bus driver implementing the pci bus interface.</p> <p>The isapci driver does not provide the <code>drv_probe()</code> routine. In other words, the isapci driver does not enumerate the bus nor does it detect an isapci device or create an associated device node. When the isapci driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for the particular bus architecture.</p> <p>The isapci driver provides the <code>drv_bind()</code> routine. This routine examines the device node name property (PROP_NODE) in order to recognize an isapci compatible device. The routine checks whether the device node name matches a pre-defined device name. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node attaching a PROP_DRIVER property to the device node. The property value specifies the isapci driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. In other words, via the PROP_DRIVER property, the driver gives its name to the parent bus driver asking it to invoke the <code>drv_init()</code> routine on that device. The <code>drv_bind()</code> routine does nothing if a PROP_DRIVER property is already present in the device node. In other words, the <code>drv_bind()</code> routine will not override existing driver-to-device binding.</p> <p>The driver does not implement <code>drv_unload()</code>. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>The driver supports the PCI_SYS_SHUTDOWN event specified by the PCI interface.</p>

The Table below summarizes the characteristics of the isapci ISA bus driver:

driver name:	"sun:pci-generic-(bus,isa)"
hardware:	Intel i386AT standard PCI/ISA bridge
exported interface:	"bus,isa" (BUS_CLASS,ISA_CLASS)
exported interface version:	0 (ISA_VERSION_INITIAL)
imported parent interface:	"pci" (PCI_CLASS)
minimal parent interface version:	0 (PCI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

The isapci driver does not use any specific properties other than its name.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

pci(9DDI), bus(9DDI), isa(9DDI)

NAME	m48txx – SGS m48txx real time clock device driver																
SYNOPSIS	<div>drv/src/rtc/m48txx/m48txx.c - driver code</div> <div>drv/src/rtc/m48txx/m48txxProp.h - driver specific properties</div>																
FEATURES	DRV																
DESCRIPTION	<p>The m48txx driver implements the rtc and nvram device driver interface.</p> <p>The driver uses the common bus driver interface provided by a parent bus driver. Thus, the driver is generic and may be applied to any bus providing this type of interface.</p> <p>The m48txx driver does not provide the <code>drv_probe()</code> routine. In other words, the m48txx driver does not enumerate the bus, nor does it detect an m48txx device or create an associated device node. When the m48txx driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The driver does not provide the <code>drv_bind()</code> routine. In other words, the driver does not support dynamic driver-to-device binding. When the m48txx driver is used, the driver should be explicitly bound to the device using the <code>PROP_DRIVER</code> property. It is assumed that the device-to-driver binding is done either by a device node creator or by a separate binder driver. Such a driver-to-device binder could be developed for this particular bus architecture.</p> <p>The driver supports all bus events specified by the common bus driver interface. As a consequence, the driver may be used with a hot-pluggable bus (for example, PCMCIA).</p> <p>The Table below summarizes characteristics of the m48txx driver:</p> <table> <tr> <td>driver name:</td><td>"sun:bus-m48txx-(nvram,rtc)"</td></tr> <tr> <td>hardware:</td><td>SGS m48txx chip</td></tr> <tr> <td>exported interfaces:</td><td>"rtc" (RTC_CLASS) "nvram" (NVRAM_CLASS)</td></tr> <tr> <td>exported interface versions:</td><td>0 (RTC_VERSION_INITIAL) 0 (NVRAM_VERSION_INITIAL)</td></tr> <tr> <td>imported parent interface:</td><td>"bus" (BUS_CLASS)</td></tr> <tr> <td>minimal parent interface version:</td><td>0 (BUS_VERSION_INITIAL)</td></tr> <tr> <td>device probing (auto-detection):</td><td>not supported</td></tr> <tr> <td>driver unloading:</td><td>supported</td></tr> </table>	driver name:	"sun:bus-m48txx-(nvram,rtc)"	hardware:	SGS m48txx chip	exported interfaces:	"rtc" (RTC_CLASS) "nvram" (NVRAM_CLASS)	exported interface versions:	0 (RTC_VERSION_INITIAL) 0 (NVRAM_VERSION_INITIAL)	imported parent interface:	"bus" (BUS_CLASS)	minimal parent interface version:	0 (BUS_VERSION_INITIAL)	device probing (auto-detection):	not supported	driver unloading:	supported
driver name:	"sun:bus-m48txx-(nvram,rtc)"																
hardware:	SGS m48txx chip																
exported interfaces:	"rtc" (RTC_CLASS) "nvram" (NVRAM_CLASS)																
exported interface versions:	0 (RTC_VERSION_INITIAL) 0 (NVRAM_VERSION_INITIAL)																
imported parent interface:	"bus" (BUS_CLASS)																
minimal parent interface version:	0 (BUS_VERSION_INITIAL)																
device probing (auto-detection):	not supported																
driver unloading:	supported																

system (emergency) shut-down: supported
normal device shut-down: supported
hot-plug (surprise) device removal: supported

The Table below lists device node properties used by the m48txx driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"io-regs"	BUS_PROP_IO_REGS	<bus class specific>	m	
"nvram-layout"	NVRAM_PROP_IO_REGS	<NvramPropChunk[]>	o	

The BUS_PROP_IO_REGS property specifies the m48txx I/O registers range. The property value is bus class specific.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

bus(9DDI), isa(9DDI), rtc(9DDI) nvram(9DDI)

NAME	mc146818 – Motorola mc146818 real time clock device driver										
SYNOPSIS	<p>drv/src/rtc/mc146818/mc146818.c - driver code</p> <p>drv/src/rtc/mc146818/mc146818Prop.h - driver specific properties</p>										
FEATURES	DRV										
DESCRIPTION	<p>The mc146818 driver implements the rtc device driver interface and the timer device driver interface.</p> <p>The driver uses the common bus driver interface provided by a parent bus driver. Thus, the driver is generic and may be applied to any bus providing this type of interface.</p> <p>The mc146818 driver does not provide the <code>drv_probe()</code> routine. In other words, the mc146818 driver does not enumerate the bus, nor does it detect an mc146818 device or create an associated device node. When the mc146818 driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The driver does not provide the <code>drv_bind()</code> routine. In other words, the driver does not support dynamic driver-to-device binding. When the mc146818 driver is used, the driver should be explicitly bound to the device using the <code>PROP_DRIVER</code> property. It is assumed that the device-to-driver binding is done either by a device node creator or by a separate binder driver. Such a driver-to-device binder could be developed for this particular bus architecture.</p> <p>The driver does not implement <code>drv_unload()</code>. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>The driver does not support the <code>BUS_DEV_REMOVAL</code> bus event specified by the common bus driver interface. As a consequence, the driver may not be used with a hot-pluggable bus (for example, PCMCIA).</p> <p>The table below summarizes the characteristics of the mc146818 driver:</p> <table> <tr> <td>driver name:</td><td>"sun:bus-mc146818-(rtc,timer)"</td></tr> <tr> <td>hardware:</td><td>Motorola MC146818 RTC chip</td></tr> <tr> <td>exported interfaces:</td><td>"rtc" (RTC_CLASS)</td></tr> <tr> <td></td><td>"timer" (TIMER_CLASS)</td></tr> <tr> <td>exported interface version:</td><td>0 ((rtc,timer)_VERSION_INITIAL)</td></tr> </table>	driver name:	"sun:bus-mc146818-(rtc,timer)"	hardware:	Motorola MC146818 RTC chip	exported interfaces:	"rtc" (RTC_CLASS)		"timer" (TIMER_CLASS)	exported interface version:	0 ((rtc,timer)_VERSION_INITIAL)
driver name:	"sun:bus-mc146818-(rtc,timer)"										
hardware:	Motorola MC146818 RTC chip										
exported interfaces:	"rtc" (RTC_CLASS)										
	"timer" (TIMER_CLASS)										
exported interface version:	0 ((rtc,timer)_VERSION_INITIAL)										

imported parent interface:	"bus" (BUS_CLASS)
minimal parent interface version:	0 (BUS_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	not supported

The Table below lists device node properties used by the mc146818 driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"io-regs"	BUS_PROP_IO_REGS	<bus class specific>	m	
"intr"	BUS_PROP_INTR	<bus class specific>	m	

The BUS_PROP_IO_REGS property specifies the mc146818 I/O registers range. The property value is bus class specific.

The BUS_PROP_INTR property specifies the mc146818 interrupt source for the timer interface. The property value is bus class specific.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`bus(9DDI)`, `isa(9DDI)`, `rtc(9DDI)`, `timer(9DDI)`

NAME	ne2000 – ne2000 Ethernet device driver
SYNOPSIS	<p>drv/src/net/ether/ne2000/ne2000.h - ne2000 hardware constants</p> <p>drv/src/net/ether/ne2000/ne2000.c - driver code</p> <p>drv/src/net/ether/ne2000/ne2000Prop.h - driver specific properties</p>
FEATURES	DRV
DESCRIPTION	<p>The ne2000 Ethernet driver implements the ether device driver interface.</p> <p>The driver uses the common bus driver interface provided by a parent bus driver. Thus, the driver may be applied to any bus with this interface.</p> <p>The ne2000 Ethernet driver supports ne2000 boards in both ISA and PCI versions.</p> <p>At initialization, the ne2000 driver is registered twice in the driver registry. The first instance is as a common bus interface and the second as a PCI one.</p> <p>The ne2000 driver does not provide the <code>drv_probe()</code> routine. In other words, the ne2000 driver does not enumerate the PCI bus, nor does it detect an ne2000 device or create an associated device node. When the ne2000 driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate PCI bus enumerator driver.</p> <p>The ne2000 driver, when registered as a pci based device driver, provides the <code>drv_bind()</code> routine. This routine examines the device node properties and thus identifies an ne2000 compatible device. The routine checks for the PCI vendor and device identifiers matching an ne2000 compatible device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the ne2000 driver name. The parent bus driver uses such a property to determine the name of drivers servicing the device. Via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver, asking it to invoke the <code>drv_int()</code> routine on that device.</p> <p>The <code>drv_bind()</code> routine does nothing if the <code>PROP_DRIVER</code> property is already present in the device node. In other words, the <code>drv_bind()</code> routine will not override existing driver-to-device binding.</p> <p>The driver does not implement <code>drv_unload()</code>. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>The Table below lists device node properties used by the ne2000 driver.</p>

driver name:	"sun:bus-ne2000-ether"
hardware:	NE2000 Ethernet controller
exported interface:	"ether" (ETHER_CLASS)
exported interface version:	0 (ETHER_VERSION_INITIAL)
imported parent interface:	"bus" (BUS_CLASS) "pci" (PCI_CLASS)
minimal parent interface version:	0 (BUS_VERSION_INITIAL) 0 (PCI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	not supported

Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the "default value" column shows the default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default Value
"io-regs"	BUS_PROP_IO_REGS	<bus class specific>	m	
"intr"	BUS_PROP_INTR	<bus class specific>	m	
"ether-addr"	ETHER_PROP_ADDR	EtherPropAddr	o	Rom content

The BUS_PROP_INTR property specifies the bus interrupt used by the device.

The BUS_PROP_IO_REGS property specifies the ne2000 I/O registers range.

The Table below lists device node properties added by the ne2000 driver, which may be used by the driver clients.

Name	Alias	Type
"link-throughput"	ETHER_PROP_THROUGHPUT	EtherPropThroughput
"ether-addr"	WETHER_PROP_ADDR	EtherPropAddr

ETHER_PROP_THROUGHPUT specifies the throughput of the Ethernet link. The ne2000 driver adds this property to the device node and initializes it with the throughput of the Ethernet link to which the device is connected.

ETHER_PROP_ADDR specifies the device Ethernet address. The ne2000 driver adds this property to the device node and initializes it with the Ethernet address found in the controller ROM.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`ether(9DDI)`, `bus(9DDI)`, `pci(9DDI)`

NAME	ns16550 – generic ns16x50 compatible uart device driver						
Synopsis	<p>drv/src/uart/ns16550/ns16550.h - ns16x50 hardware related constants</p> <p>drv/src/uart/ns16550/ns16550.c - driver code</p> <p>drv/src/uart/ns16550/ns16550Prop.h - driver specific properties</p>						
FEATURES	DRV						
DESCRIPTION	Implements the ns16550 device driver interface.						
EXTENDED DESCRIPTION	<p>The <code>ns16550</code> <code>uart</code> driver implements the <code>uart</code> device driver interface. The driver works on a <code>uart</code> device which is software compatible with the <code>ns16x50</code> chip.</p> <p>The driver uses the common bus driver interface provided by a parent bus driver. Thus, the driver is generic and may be applied to any bus providing this type of interface.</p> <p>The <code>ns16x50</code> driver does not provide the <code>drv_probe()</code> routine. In other words, the <code>ns16x50</code> driver does not enumerate the bus, nor does it detect an <code>ns16x50</code> device or create an associated device node. When the <code>ns16x50</code> driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The driver does not provide the <code>drv_bind()</code> routine. In other words, the driver does not support dynamic driver-to-device binding. When the <code>ns16x50</code> driver is used, the driver should be explicitly bound to the device using the <code>PROP_DRIVER</code> property. It is assumed that the device-to-driver binding is done either by a device node creator or by a separate binder driver. Such a driver-to-device binder could be developed for this particular bus architecture.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p> <p>The driver supports all bus events specified by the common bus driver interface. As a consequence, the driver may be used with a hot-pluggable bus (for example, PCMCIA). The Table below summarizes the characteristics of the <code>ns16x50</code> <code>uart</code> driver:</p> <table> <tr> <td>driver name:</td><td>"chorus@bus-ns16550-uart"</td></tr> <tr> <td>hardware:</td><td>NS16x50 compatible UART chip</td></tr> <tr> <td>exported interface:</td><td>"uart" (UART_CLASS)</td></tr> </table>	driver name:	"chorus@bus-ns16550-uart"	hardware:	NS16x50 compatible UART chip	exported interface:	"uart" (UART_CLASS)
driver name:	"chorus@bus-ns16550-uart"						
hardware:	NS16x50 compatible UART chip						
exported interface:	"uart" (UART_CLASS)						

exported interface version:	0 (UART_VERSION_INITIAL)
imported parent interface:	"bus" (BUS_CLASS)
minimal parent interface version:	0 (BUS_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	supported

The table below lists device node properties used by the ns16550 driver.

Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default Value
"io-regs"	BUS_PROP_IO_REGS	<bus class specific>	m	
"intr"	BUS_PROP_INTR	<bus class specific>	m	
"clock-freq"	PROP_CLOCK_FREQ	PropClockFreq	o	1843200

The BUS_PROP_IO_REGS property specifies the ns16x50 I/O registers range. The property value is bus class specific.

The BUS_PROP_INTR property specifies the ns16x50 interrupt source. The property value is bus class specific.

The PROP_CLOCK_FREQ property specifies the input clock frequency in Hz. By default, the input clock frequency is 1843200 Hz. Note that when programming the timer divisor for a given baud rate, the driver assumes the 16x clock factor. The formula relating the baud rate to the timer divisor is shown below.

$$\text{<divisor>} = (\text{<clock-freq>} + (8 * \text{<baud-rate>})) / (16 * \text{<baud-rate>})$$

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

uart(9DDI), bus(9DDI)

NAME	pcibios – Intel i386AT generic pci bridge, pci bus driver
SYNOPSIS	<p>x86/drv_f/src/pci/pcibios/generic.c - driver code</p> <p>x86/drv_f/src/pci/pcibios/genericProp.h - driver specific properties</p>
FEATURES	DRV
DESCRIPTION	<p>The pcibios bus driver implements:</p> <ul style="list-style-type: none"> ■ The common bus driver interface, and ■ The pci bus driver interface. <p>The driver works on any Intel i386AT platform equipped with a pci bus.</p> <p>The driver uses the x86 specific dki interface provided by the kernel. Thus, the driver only works with Intel family based products.</p> <p>The pcibios driver does not provide the <code>drv_probe()</code> routine. In other words, the pcibios driver does not enumerate the bus nor does it detect a pcibios device or create an associated device node. Then the pcibios driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for the particular bus architecture.</p> <p>The pcibios driver provides the <code>drv_bind()</code> routine. This routine examines the device node name property (PROP_NODE) in order to recognize a pcibios compatible device. The routine checks whether the device node name matches a pre-defined device name. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node attaching a PROP_DRIVER property to the device node. The property value specifies the ric driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. In other words, via the PROP_DRIVER property, the driver gives its name to the parent bus driver asking it to invoke the <code>drv_init()</code> routine on that device. The <code>drv_bind()</code> routine does nothing if the PROP_DRIVER property is already present in the device node. In other words, the <code>drv_bind()</code> routine will not override existing driver-to-device binding.</p> <p>The driver does not implement <code>drv_unload()</code>. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>The driver supports the DKI_SYS_SHUTDOWN event specified by the common dki interface.</p> <p>The table below summarizes characteristics of the pcibios pci bus driver.</p>

driver name:	"sun:x86-bios-(bus,pci)"
hardware:	Intel i386AT standard PCI bridge
exported interface:	"bus,pci" (BUS_CLASS,PCI_CLASS)
exported interface version:	0 (PCI_VERSION_INITIAL)
imported parent interface:	"x86" (FDKI_CLASS)
minimal parent interface version:	0 (FDKI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

ATTRIBUTES

pcibios driver does not use any specific properties other than its name.
See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`pci(9DDI)`, `bus(9DDI)`

NAME	pciconf – pci configuration space parser driver
SYNOPSIS	drv/src/pci/pciconf/pciconf.c - driver code
FEATURES	DRV
DESCRIPTION	The pciconf driver implements no interface, it is a probe-only driver.
	The driver uses the pci bus driver interface provided by a parent bus driver. Thus, the driver may be applied to any pci bus which provides this interface.
	The aim of this driver is to browse the pci bus configuration space.
	The driver provides the drv_unload() entry and supports the driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.
	For each device found, pciconf displays all relevant pci header fields as specified in the table below :

"base-addr[5]"		x
"cardbus-cis"		x
"sub-vendor-id"		x
"sub-id"		x
"exp-rom-base"		x
"intr-line"		x
"intr-pin"		x
"min-gnt"		x
"max-lat"		x
"prim-bus"	x	
"sec-bus"	x	
"sub-bus"	x	
"pci-la"	x	
"t-timer"	x	
"io-base"	x	
"io-limit"	x	
"pci-status"	x	
"mem- io-base"	x	
"mem-io-limit"	x	
"mem-base"	x	
"mem-limit"	x	
"mem-base-hi"	x	
"mem-limit-hi"	x	
"io-base-hi"	x	
"io-limit-hi"	x	
"exp-rom-base"	x	
"intr-line"	x	
"intr-pin"	x	
"pci-control"	x	

NOTE : A "pci-to-pci bridge" is a pci device which makes it possible to add a pci bus into the system.

The table below summarizes characteristics of the pciconf driver.

driver name:	"sun:pci-parser-"
hardware:	none
exported interface:	none
exported interface version:	none
imported parent interface:	"pci" (PCI_CLASS)
minimal parent interface version:	0 (PCI_VERSION_INITIAL)
device probing (auto-detection):	supported
driver unloading:	supported
system (emergency) shut-down:	not supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

ether(9DDI), pci(9DDI)

NAME	pcienum – pci enumerator driver
SYNOPSIS	drv/src/pci/pcienum/pcienum.c - driver code
FEATURES	DRV
DESCRIPTION	<p>The pcienum driver does not implement any interface, it is a probe-only driver.</p> <p>The driver uses the pci bus driver interface provided by a parent bus driver. Thus, the driver may be applied to any pci bus which provides this interface.</p> <p>The aim of this driver is to scan the pci bus to detect all connected devices and to update the device tree.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports the driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p> <p>The pci standard properties that the driver can create are the following :</p> <ul style="list-style-type: none"> ■ PCI_PROP_DEV_NUM ■ PCI_PROP_FUNC_NUM ■ PCI_PROP_VENDOR_ID ■ PCI_PROP_DEV_ID <p>The pci device specific properties that the driver can create are the following :</p> <ul style="list-style-type: none"> ■ PCI_PROP_INTR ■ PCI_PROP_IO_REGS ■ PCI_PROP_MEM_RGN ■ PCI64_PROP_MEM_RGN <p>For each device declared in the device tree, pcienum verifies that the node corresponds to the hardware configuration. If a problem appears, the node is deleted. If there is no problem, pcienum will complete the node if necessary.</p> <p>After this first check, pcienum scans the pci bus. If pcienum detects a device that is undeclared in the device tree, pcienum will create a node and will complete it.</p> <p>Each time that the pcienum driver creates/deletes a node/property in the device tree, a message is printed on the console.</p> <p>The node's names created by pcienum are formatted as follows :</p> <pre>pci<vendorid>,<deviceid>@<devicenumber>,<devicefunction></pre>

To suppress messages from the pcienum driver, you must comment out the PCIENUM_TRACE declaration in the pcienum.c file.

The table below summarizes characteristics of the pcienum driver.

driver name:	"sun:pci-pci-enumerator"
hardware:	none
exported interface:	none
exported interface version:	none
imported parent interface:	"pci" (PCI_CLASS)
minimal parent interface version:	0 (PCI_VERSION_INITIAL)
device probing (auto-detection):	supported
driver unloading:	supported
system (emergency) shut-down:	not supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

pciconf(9DRV), pci(9DDI)

NAME	quicc8260 – quicc bus driver for Motorola mpc8260 micro-controllers.
SYNOPSIS	<p>drv_f/src/quicc/8260/quicc8260.h - mpc8260 specific constants</p> <p>drv_f/src/quicc/8260/quicc8260.c - driver code</p> <p>drv_f/src/quicc/8260/quicc8260Prop.h - driver specific properties</p>
FEATURES	DRV
DESCRIPTION	<p>The quicc8260 bus driver implements:</p> <ul style="list-style-type: none"> ■ the common bus driver interface ■ the quicc bus driver interface <p>The driver works on Motorola mpc8260 micro-controllers and provides an abstraction of the CPM peripheral bus and local and PowerPC busses.</p> <p>The driver uses the PowerPC specific dki interface provided by the kernel. Thus, the driver works only with the PowerPC family of products.</p> <p>The quicc8260 driver does not provide the <code>drv_probe()</code> routine. In other words, the quicc8260 driver does not enumerate the bus, nor does it detect a quicc8260 device or create an associated device node. When the quicc8260 driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The quicc8260 driver provides the <code>drv_bind()</code> routine. This routine examines the device node property <code>PROP_NODE</code> in order to recognize a quicc8260 compatible device. This routine checks whether the device node name matches a pre-defined device name. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the quicc8260 driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. In other words, via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver asking it to invoke the <code>drv_init()</code> routine on that device. Note that the <code>drv_bind()</code> routine is not active if a <code>PROP_DRIVER</code> property is already present in the device node. In other words, the <code>drv_bind()</code> routine does not override existing driver-to-device binding.</p> <p>The driver does not provide the <code>drv_unload()</code> entry. Thus, the driver component cannot be unloaded, even if it has been dynamically loaded at run time.</p> <p>The driver supports the <code>DKI_SYS_SHUTDOWN</code> event specified by the common DKI interface.</p>

The table below summarizes characteristics of the quicc8260 bus driver.

driver name:	"sun:powerpc-mpc8260-(bus, quicc)"
hardware:	Motorola MPC8260 micro-controllers
exported interface:	"quicc" (QUICC_CLASS)
exported interface version:	0 (QUICC_VERSION_INITIAL)
imported parent interface:	"powerpc" (FDKI_CLASS)
minimal parent interface version:	0 (FDKI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver to device binding:	supported (on node name basis)
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

The Table below lists device node properties used by the quicc8260 driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default Value
"imap-rgn"	QUICC_PROP_IMAP_RGN	QuiccPropMemRgn	m	
"dpram-rgn"	QUICC_PROP_DPRAM_RGN	QuiccPropMemRgn	m	
"clock-freq"	QUICC_PROP_CLOCK_FREQ	QuiccPropClockFreq	m	
"brg-clock-freq"	QUICC_PROP_BRG_CLOCK_FREQ	QuiccPropClockFreq	m	
"cpm-clock-freq"	QUICC_PROP_CPMCLK	QuiccPropClockFreq	m	
"mem-rgn"	QUICC_PROP_MEM_RGN	QuiccPropMemRgn	o	see text
"pin-config"	QUICC_8260_PROP_PIN_CONFIG	Quicc8260PropPinConfig	m	
"intr-config"	QUICC_8260_PROP_INTR_CONFIG	Quicc8260PropIntrConfig	o	see text

The `QUICC_PROP_IMAP_RGN` property specifies the internal memory region mapping.

The `QUICC_PROP_DPRAM_RGN` property specifies the Dual Ported RAM memory region mapping.

The `QUICC_PROP_CLOCK_FREQ` property specifies the local bus clock frequency.

The `QUICC_PROP_CPM_CLOCK_FREQ` property specifies the CPM processor clock frequency.

The `QUICC_PROP_BRG_CLOCK_FREQ` property specifies the BRGs input clock frequency.

The `QUICC_PROP_MEM_RGN` property specifies the regions in the internal memory map where allocation of DMA regions is allowed. This property is an array that declares the ranges of memory, in the internal memory map, which are free for DMA usage (`dma_alloc()`/`dma_free()`). By default, the entire DPRAM region of the internal memory map is usable for DMA allocation purposes.

The `QUICC_8260_PROP_PIN_CONFIG` property specifies the configuration and routing of the parallel I/O ports' pins and internal controllers' clocks. This property provides the values to be set in internal map registers which manage the configuration and routing of parallel I/O port pins. For each port (A, B, C, D), the following registers must be provided:

- pin assignment register
- pin special option register
- data direction register
- open drain register
- data register value.

In addition, the following clock configuration registers must be provided:

- CMX FCC clock route register is used to configure FCC controllers in multiplexed or non-multiplexed mode (NMSI), and to select FCC clocks.
- CMX SCC clock route register is used to configure SCC controllers in multiplexed or non-multiplexed mode (NMSI), and to select SCC clocks.
- CMX SMC clock route register is used to configure SMC controllers in multiplexed or non-multiplexed mode (NMSI), and to select SMC clocks

The `QUICC_8260_PROP_INTR_CONFIG` property specifies the interrupts configuration. This property defines :

- SIU interrupts modes configuration (*sicr* property field)
- SIU interrupts priorities (*siprr* property field)
- CPM FCCs and MCCs relative priorities (*scprrr_h* property field)
- CPM SCCs relative priorities (*scprrr_1* property field)

The *sicr* property field should be set using the following macro:

```
SIU_INTR_CONFIG(siu_prio_mode, scc_prio_mode, highest)
```

siu_prio_mode indicates the PIT, TMCNT and IRQ1–5 priority mode to be used:

- **SIU_PRIO_GROUPED**: interrupts are grouped by priority
- **SIU_PRIO_SPREAD**: interrupts are spread by priority among other sources

scc_prio_mode indicates the SCC priority mode to use in:

- **SCC_PRIO_GROUPED**: SCC interrupts are grouped by priority in higher priorities
- **SCC_PRIO_SPREAD**: SCC interrupts are spread by priority among other sources

highest indicates a value of type *QuiccPropIntr* which should be assigned to the highest priority.

The *siprr* property field should be set using the following macro:

```
SIU_INTR_PRIORITY(p1, p2, p3, p4, p5, p6, p7, p8)
```

where *px* is a value of type *QuiccPropIntr* in [QUICC8260_TMCNT_INTR, QUICC8260_IRQ5_INTR] to be assigned to SIU priority position <XSIUx> (p1=highest, p8=lowest)

The *scprrr_h* property field should be set using the following macro:

```
CPM_FAST_INTR_PRIORITY (p1, p2, p3, p4, p5, p6, p7, p8)
```

where *px* is a value of type *QuiccPropIntr* in [QUICC8260_FCCI_INTR, QUICC8260_MCC2_INTR], or —1, to be assigned to SIU priority position <XCCx> (p1=highest, p8=lowest).

The *scprrr_1* property field should be set using the following macro:


```
CPM_SCC_INTR_PRIORITY(p1, p2, p3, p4, p5, p6, p7, p8)
```

where `px` is a value of type `QuiccPropIntr` in `{QUICC8260_SCC1_INTR, QUICC8260_SCC4_INTR}`, or `-1`, to be assigned to SIU priority position `<YCCx>` (`p1`=highest, `p8`=lowest).

By default, interrupts are configured as follows:

```
SIU_INTR_CONFIG(SIU_PRIO_SPREAD, SCC_PRIO_SPREAD, QUICC8260_TMCNT_INTR)
    SIU_INTR_PRIORITY(QUICC8260_TMCNT_INTR, QUICC8260_PIT_INTR,
        QUICC8260_IRQ1_INTR, QUICC8260_IRQ2_INTR,
        QUICC8260_IRQ3_INTR, QUICC8260_IRQ4_INTR,
        QUICC8260_IRQ5_INTR, QUICC8260_IRQ6_INTR)
    CPM_FAST_INTR_PRIORITY(QUICC8260_FCC1_INTR, QUICC8260_FCC2_INTR,
        QUICC8260_FCC3_INTR, QUICC8260_MCC1_INTR,
        QUICC8260_MCC2_INTR, -1, -1, -1)
    CPM_SCC_INTR_PRIORITY(QUICC8260_SCC1_INTR, QUICC8260_SCC2_INTR,
        QUICC8260_SCC3_INTR, QUICC8260_SCC4_INTR,
        -1, -1, -1, -1)
```

RESTRICTIONS

The current version of the quicc8260 driver does not support dynamic assignment of quicc bus resources.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`quicc(9DDI)`, `bus(9DDI)`

NAME	quicc8xx – quicc bus driver for Motorola mpc8xx micro-controllers.
SYNOPSIS	<p>drv_f/src/quicc/8xx/quicc8xx.h - mpc8xx specific constants</p> <p>drv_f/src/quicc/8xx/quicc8xx.c - driver code</p> <p>drv_f/src/quicc/8xx/quicc8xxProp.h - driver specific properties</p>
FEATURES	DRV
DESCRIPTION	<p>The quicc8xx bus driver implements:</p> <ul style="list-style-type: none"> ■ the common bus driver interface ■ the quicc bus driver interface <p>The driver works on Motorola mpc8xx micro-controllers and provides an abstraction of the CPM peripheral bus and the external U-BUS.</p> <p>The driver uses the PowerPC specific dki interface provided by the kernel. Thus, the driver works only with PowerPC family products.</p> <p>The quicc8xx driver does not provide the <code>drv_probe()</code> routine. In other words, the quicc8xx driver does not enumerate the bus, nor does it detect a quicc8xx device or create an associated device node. When the quicc8xx driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The quicc8xx driver provides the <code>drv_bind()</code> routine. This routine examines the device node property <code>PROP_NODE</code> in order to recognize a quicc8xx compatible device. This routine checks whether the device node name matches a pre-defined device name. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the quicc8xx driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. In other words, via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver asking it to invoke the <code>drv_init()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if a <code>PROP_DRIVER</code> property is already present in the device node. In other words, the <code>drv_bind()</code> routine does not override existing driver-to-device binding.</p> <p>The driver does not provide the <code>drv_unload()</code> entry. Thus, the driver component cannot be unloaded, even if it has been dynamically loaded at run time.</p> <p>The driver supports the <code>DKI_SYS_SHUTDOWN</code> event specified by the common dki interface.</p>

The Table below summarizes characteristics of the quicc8xx bus driver.

driver name:	"sun:powerpc-mpc8xx-(bus, quicc)"
hardware:	Motorola MPC8xx micro-controllers
exported interface:	"quicc" (QUICC_CLASS)
exported interface version:	0 (QUICC_VERSION_INITIAL)
imported parent interface:	"powerpc" (FDKI_CLASS)
minimal parent interface version:	0 (FDKI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver to device binding:	supported (on node name basis)
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

The Table below lists device node properties used by the quicc8xx driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default Value
"imap-rgn"	QUICC_PROP_IMAP_RGN	QuiccPropMemRgn	m	
"dpram-rgn"	QUICC_PROP_DPRAM_RGN	QuiccPropMemRgn	m	
"clock-freq"	QUICC_PROP_CLOCK_FREQ	QuiccPropClockFreq	m	
"brg-clock-freq"	QUICC_PROP_BRG_CLOCK_FREQ	QuiccPropClockFreq	m	
"cpm"-clock-freq	QUICC_PROP_CPMCLK	QuiccPropClockFreq	m	
"mem-rgn"	QUICC_PROP_MEM_RGN	QuiccPropMemRgn	o	see text
"pin-config"	QUICC_8xx_PROP_PIN_CONFIG	Quicc8xxPropPinConfig	m	
"intr-config"	QUICC_8xx_PROP_INTR_CONFIG	Quicc8xxPropIntrConfig	o	see text

The `QUICC_PROP_IMAP_RGN` property specifies the internal memory region mapping.

The `QUICC_PROP_DPRAM_RGN` property specifies the Dual Ported RAM memory region mapping.

The `QUICC_PROP_CLOCK_FREQ` property specifies the local bus clock frequency.

The `QUICC_PROP_CPM_CLOCK_FREQ` property specifies the CPM processor clock frequency.

The `QUICC_PROP_BRG_CLOCK_FREQ` property specifies the BRGs input clock frequency.

The `QUICC_PROP_MEM_RGN` property specifies the regions in the internal memory map where allocation of DMA regions is allowed. This property is an array that declares the ranges of memory, in the internal memory map, which are free for DMA usage (`dma_alloc()`/`dma_free()`). By default, the entire DPRAM region of the internal memory map is usable for DMA allocation purposes.

The `QUICC_8XX_PROP_PIN_CONFIG` property specifies the configuration and routing of the parallel I/O ports. This property provides the values to be set in internal map registers which manage the configuration and routing of parallel I/O port pins. For each port (A, B, C, D), the following registers must be provided:

- Pin assignment register.
- Data direction register.
- Open drain, or special option when existing for a port.
- Data register value. Note that port data register is set only if the provided value is not zero.

In addition, the following SI registers must be provided:

- SI mode register is used to configure SMC in multiplexed or non-multiplexed mode (NMSI), and to select SMC clocks.
- SI clock route register is used to configure SCC devices in multiplexed or non-multiplexed mode (NMSI), and to select SCC clocks

The `QUICC_8XX_PROP_INTR_CONFIG` property specifies the interrupts configuration. This property defines :

- the priority (ranging from lowest `SIU_PRIO_0` to highest `SIU_PRIO_7`) of each SIU internal source of interrupt :

- time base
- periodic interrupt timer
- real time clock
- PMCIA card A interrupt request and status changed sources
- PMCIA card B interrupt request and status changed sources
- cpm interrupt sources
- the interrupt configuration for CPM interrupt sources

The "cpmIntrConfig" property field should be set using the following macro:

```
CPM_INTR_CONFIG(scc_prio_mode, scc_p0, scc_p1, scc_p2, scc_p3, highest)
```

`scc_prio_mode` indicates the SCC priority mode to use in:

- `SCC_PRIO_GROUPED`: SCC interrupts are grouped by priority in higher priorities
- `SCC_PRIO_SPREAD`: SCC interrupts are spread by priority among other sources

`scc_p0, 1, 2, 3` indicates which SCC device (in 1 to 4) is assigned to each of the four available SCC priority levels (from lowest `scc_p0` to highest `scc_p3`).

`highest` indicates a value of type `QuiccPropIntr` which should be assigned to the highest CPM priority.

Default settings for SIU interrupt priorities are:

time base	not set
periodic interrupt timer	not set
real time clock	not set
PMCIA card A interrupt request	not set
PMCIA card A status changed	not set
PMCIA card B interrupt request	not set
PMCIA card B status changed	not set
cpm	SIU_PRIO_4

RESTRICTIONS

cpm interrupt configuration is set to: CPM_INTR_CONFIG(SCC_PPIO_SPREAD, 4, 3, 2, 1, QUICC_PC15_INTR).

The current version of the quicc8xx driver does not support dynamic assignment of quicc bus resources. The quicc8xx driver does not implement I/O pin management services (io_pin_attach).

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

quicc(9DDI), bus(9DDI)

NAME	raven – Motorola pci host bridge, pci bus driver
Synopsis	<div>drv_f/src/pci/raven/raven.h - RAVEN hardware related constants</div> <div>drv_f/src/pci/raven/raven.c - driver code</div> <div>drv_f/src/pci/raven/ravenProp.h - driver specific properties</div>
FEATURES	DRV
DESCRIPTION	<p>The raven bridge driver implements:</p> <ul style="list-style-type: none"> ■ The common bus driver interface, ■ The pci bus driver interface. <p>The driver works on a Motorola raven pci host bridge and uses the PowerPC specific dki interface provided by the kernel.</p> <p>Thus, the driver is designed to work only with PowerPC family products.</p> <p>The raven driver does not provide the <code>drv_probe()</code> routine. In other words, the raven driver does not enumerate the pci bus, nor does it detect a raven device or create an associated device node. When the raven driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate pci bus enumerator driver.</p> <p>The raven driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties and thus, identifies a raven compatible device. The routine checks for the pci vendor and device identifiers matching a raven compatible device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the raven driver name. The parent bus driver uses such a property to determine the name of drivers servicing the device. Via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver, asking it to invoke the <code>drv_init()</code> routine on that device. The <code>drv_bind()</code> routine does nothing if the <code>PROP_DRIVER</code> property is already present in the device node. In other words, the <code>drv_bind()</code> routine will not override existing driver-to-device binding.</p> <p>The driver does not implement <code>drv_unload()</code>. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>The driver supports the <code>DKI_SYS_SHUTDOWN</code> event specified by the common dki interface.</p>

The Table below summarizes characteristics of the raven pci bus driver:

driver name:	"sun:powerpc-raven-(bus,pci)"
hardware:	Motorola RAVEN PCI host bridge
exported interface:	"bus,pci" (BUS_CLASS,PCI_CLASS)
exported interface version:	0 (PCI_VERSION_INITIAL)
imported parent interface:	"powerpc" (FDKI_CLASS)
minimal parent interface version:	0 (FDKI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver-to-device binding:	supported (on PCI dev/vend id basis)
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

The Table below lists device node properties used by the raven driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default Value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"bus-num"	PCI_PROP_BUS_NUM	PciPropBusNum	m	
"dev-num"	PCI_PROP_DEV_NUM	PciPropDevNum	m	
"func-num"	PCI_PROP_FUNC_NUM	PciPropFuncNum	m	
"io-regs"	PCI_PROP_IO_REGS	PciPropIoRegs	m	
"mem-rgn"	PCI_PROP_MEM_RGN	PciPropMemRgn	m	
"pci-map"	RAVEN_PROP_PCI_MAP	RavenPropPciMap	m	
"ppc-map"	RAVEN_PROP_PPC_MAP	RavenPropPpcMap	m	
"intr-map"	RAVEN_PROP_INTR_MAP	RavenPropIntrMap	m	

Name	Alias	Type	m/o	Default Value
"mpic-intr"	RAVEN_PROP_MPIC_INTR	RavenPropMpicIntr	o	see below
"ppc-timeout"	RAVEN_PROP_PPC_TIMEOUT	RavenPropMpicIntr	o	256 us

The `PCI_PROP_VEND_ID` property specifies the Motorola pci vendor identifier (must be 0x1057).

The `PCI_PROP_DEV_ID` property specifies the raven pci device identifier (must be 0x4801).

The `PCI_PROP_BUS_NUM`, `PCI_PROP_DEV_NUM`, and `PCI_PROP_FUNC_NUM` properties specify the configuration space address of the device. That is the pci bus, device and function numbers to which the device is connected (these should be all zero).

The `PCI_PROP_IO_REGS` property specifies the RAVEN's MPIC I/O registers range.

The `PCI_PROP_MEM_RGN` and `RAVEN_PROP_PCI_MAP` properties specify the raven configuration for pci bus slave map decoders. These properties are arrays which must be used to configure the 4 pci slave map decoders in the raven.

Each index (#0 to #3) of `PCI_PROP_MEM_RGN` defines a pci memory address range that will be translated into PowerPC bus cycles by the raven.

Each index (#0 to #3) of `RAVEN_PROP_PCI_MAP` defines:

- The PowerPC bus address into which the associated pci region start address must be translated.
- The attributes to be set in the PSATTx RAVEN register for this pci slave map decoder.

The `RAVEN_PROP_PPC_MAP` property specifies the raven configuration for PowerPC bus slave map decoders. This property is an array which must be used to configure the 4 PowerPC slave map decoders in the raven.

Each index from #0 to #3 defines a PowerPC bus address range that will be translated into pci bus cycles by the raven, with the following elements:

- The PowerPC bus range is defined by its start address and size.
- The pci bus address into which the PowerPC start address must be translated.

- The attributes to be set in the MSATTx RAVEN register for this PowerPC slave map decoder.

The index 5# defines the location of the RAVEN's MPC registers from the PowerPC bus (only the PowerPC bus range is used) (should be at 0xFEFE0000 or 0xFEFF0000 with a size of 0x80).

The `RAVEN_PROP_INTR_MAP` property specifies, for each pci device number, a mapping of pci interrupt pins (INTA#, INTB#, INTC#, INTD#) into a RAVEN MPIC interrupt line (ranging from 0# to 25#). This property allows the raven to take into account the various pci interrupt routings for different platforms.

The `RAVEN_PROP_MPIC_INTR` property specifies the configuration for each interrupt source of the raven MPIC. For each MPIC interrupt source, this property indicates:

- The type of the interrupt (low/high level, or positive/negative edge) .
Note that this field is used only for external interrupt sources.
The default value is `OPIC_INTR_LOW_LEVEL`.
- The relative priority of the interrupt ranging from 0 (disabled) to 15 (highest).
An interrupt can be interrupted only by an interrupt of higher priority.
The default value is 1.

The `RAVEN_PROP_PPC_TIMEOUT` property specifies the configuration of the raven PowerPC bus timeout value to be set in the `raven` GCSR register.

The default value is 256 microseconds.

RESTRICTIONS

The current version of the raven driver does not support dynamic assignment of PCI bus resources.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`pci(9DDI)`, `bus(9DDI)`

NAME	ric – Sun reset, interrupt and clock controller												
SYNOPSIS	<p>drv_f/src/pci/ric/ric.c - driver code</p> <p>drv_f/src/pci/ric/ricProp.h - driver specific properties</p>												
FEATURES	DRV												
DESCRIPTION	<p>The ric driver implements a ric device specific interface.</p> <p>This driver implements the ric device specific interrupt handling interface for the Sun UltraSPARC II CPU using the kernel's UltraSPARC-specific dki interface.</p> <p>The ric driver does not provide the <code>drv_probe()</code> routine. In other words, the ric driver does not enumerate the bus nor does it detect a ric device or create an associated device node. When the ric driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for the particular bus architecture.</p> <p>The ric driver provides the <code>drv_bind()</code> routine. This routine examines the device node name property (PROP_NODE) in order to recognize a ric compatible device. The routine checks whether the device node name matches a pre-defined device name. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node attaching a PROP_DRIVER property to the device node. The property value specifies the ric driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. In other words, via the PROP_DRIVER property, the driver gives its name to the parent bus driver asking it to invoke the <code>drv_init()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if a PROP_DRIVER property is already present in the device node., so the <code>drv_bind</code> routine does not override existing driver-to-device binding.</p> <p>The driver does not implement the <code>drv_unload()</code> entry. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>The table below summarizes characteristics of the ric pci bus driver.</p> <table> <tr> <td>driver name:</td><td>"sun:usparc-ric-ric"</td></tr> <tr> <td>hardware:</td><td>Sun ric controller</td></tr> <tr> <td>exported interface:</td><td>"ric" (RIC_CLASS)</td></tr> <tr> <td>exported interface version:</td><td>0 (RIC_VERSION_INITIAL)</td></tr> <tr> <td>imported parent interface:</td><td>"usparc" (FDKI_CLASS)</td></tr> <tr> <td>minimal parent interface version:</td><td>0 (PCI_VERSION_INITIAL)</td></tr> </table>	driver name:	"sun:usparc-ric-ric"	hardware:	Sun ric controller	exported interface:	"ric" (RIC_CLASS)	exported interface version:	0 (RIC_VERSION_INITIAL)	imported parent interface:	"usparc" (FDKI_CLASS)	minimal parent interface version:	0 (PCI_VERSION_INITIAL)
driver name:	"sun:usparc-ric-ric"												
hardware:	Sun ric controller												
exported interface:	"ric" (RIC_CLASS)												
exported interface version:	0 (RIC_VERSION_INITIAL)												
imported parent interface:	"usparc" (FDKI_CLASS)												
minimal parent interface version:	0 (PCI_VERSION_INITIAL)												

driver name:	"sun:usparc-ric-ric"
device probing (auto-detection):	not supported
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

ATTRIBUTESSee `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME	sabre – sun pci host bridge, pci bus driver
SYNOPSIS	<p>drv_f/src/pci/sabre/sabre.h - SABRE hardware related constants</p> <p>drv_f/src/pci/sabre/sabre.c - driver code</p> <p>drv_f/src/pci/sabre/sabreProp.h - driver specific properties</p>
FEATURES	DRV
DESCRIPTION	<p>The sabre bridge driver implements the common bus driver interface and the pci bus interface.</p> <p>The driver uses the UltraSPARC specific DKI interface provided by the kernel. Thus, the driver is designed to work only with the Sun UltraSPARC family of products.</p> <p>The sabre driver does not provide the <code>drv_probe</code> routine. In other words, the sabre driver does not enumerate the pci bus, nor does it detect a sabre device or create an associated device node. When the sabre driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate pci bus enumerator driver.</p> <p>The sabre driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties and thus, identifies a sabre compatible device. The routine checks for the pci vendor and device identifiers matching a sabre compatible device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the sabre driver name. The parent bus driver uses such a property to determine the name of drivers servicing the device. Via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver, asking it to invoke the <code>drv_int()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if the <code>PROP_DRIVER</code> property is already present in the device node, as the <code>drv_bind()</code> routine will not override existing driver-to-driver binding.</p> <p>The driver does not implement the <code>drv_unload()</code> entry. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>The driver supports the <code>DKI_SYS_SHUTDOWN</code> event specified by the common DKI interface.</p> <p>The table below summarizes characteristics of the sabre pci bus driver.</p>

driver name:	"sun:usparc-sabre-pci"
hardware:	Sun sabre pci host bridge
exported interface:	"bus" (BUS_CLASS)
exported interface version:	0 (BUS_VERSION_INITIAL)
exported interface:	"pci" (PCI_CLASS)
exported interface version:	0 (PCI_VERSION_INITIAL)
imported parent interface:	"usparc" (FDKI_CLASS)
minimal parent interface version:	0 (FDKI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

The table below lists device node properties used by the sabre driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"bus-num"	PCI_PROP_BUS_NUM	PciPropBusNum	m	

The `PCI_PROP_VEND_ID` property specifies the Sun pci vendor identifier (must be 0x108e).

The `PCI_PROP_DEV_ID` property specifies the sabre pci device identifier (must be 0xa000).

The `PCI_PROP_BUS_NUM` property specifies the pci bus number of the device. That is the pci bus number to which the device is connected (should be zero).

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`pci(9DDI)` `bus(9DDI)`

NAME	sccEther – quicc scc controller Ethernet device driver.														
SYNOPSIS	<p>drv/src/net/ether/scc/sccEther.c - driver code</p> <p>drv/src/net/ether/scc/sccEtherProp.h - driver specific properties</p>														
FEATURES	DRV														
DESCRIPTION	<p>The sccEther Ethernet driver implements the ether device driver interface.</p> <p>The driver works on a quicc serial communication controller (SCC) connected to an external SIA and a transceiver function. The driver uses the quicc bus driver interface provided by a parent bus driver.</p> <p>The sccEther driver does not provide the <code>drv_probe()</code> routine. In other words, the sccEther driver does not enumerate the bus, nor does it detect an sccEther device or create an associated device node. When the sccEther driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The driver does not provide the <code>drv_bind()</code> routine. In other words, the driver does not support dynamic driver-to-device binding. When the sccEther driver is used, the driver should be explicitly bound to the device using the <code>PROP_DRIVER</code> property. It is assumed that the device-to-driver binding is done either by a device node creator or by a separate binder driver. Such a driver-to-device binder could be developed for this particular bus architecture.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports the driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p> <p>The driver supports the <code>QUICC_SYS_SHUTDOWN</code> and <code>QUICC_DEV_SHUTDOWN</code> bus events specified by the quicc bus driver interface.</p> <p>The table below summarizes characteristics of the sccEther Ethernet driver.</p> <table> <tr> <td>driver name:</td><td>"sun:quicc-scc-ether"</td></tr> <tr> <td>hardware:</td><td>Quicc/scc connected to 10B-T</td></tr> <tr> <td>exported interface:</td><td>"ether" (ETHER_CLASS)</td></tr> <tr> <td>exported interface version:</td><td>0 (ETHER_VERSION_INITIAL)</td></tr> <tr> <td>imported parent interface:</td><td>"quicc" (QUICC_CLASS)</td></tr> <tr> <td>minimal parent interface version:</td><td>0 (QUICC_VERSION_INITIAL)</td></tr> <tr> <td>device probing (auto-detection):</td><td>not supported</td></tr> </table>	driver name:	"sun:quicc-scc-ether"	hardware:	Quicc/scc connected to 10B-T	exported interface:	"ether" (ETHER_CLASS)	exported interface version:	0 (ETHER_VERSION_INITIAL)	imported parent interface:	"quicc" (QUICC_CLASS)	minimal parent interface version:	0 (QUICC_VERSION_INITIAL)	device probing (auto-detection):	not supported
driver name:	"sun:quicc-scc-ether"														
hardware:	Quicc/scc connected to 10B-T														
exported interface:	"ether" (ETHER_CLASS)														
exported interface version:	0 (ETHER_VERSION_INITIAL)														
imported parent interface:	"quicc" (QUICC_CLASS)														
minimal parent interface version:	0 (QUICC_VERSION_INITIAL)														
device probing (auto-detection):	not supported														

driver-to-device binding:	not supported
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	not supported

The table below lists device node properties used by the sccEther driver.

Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default Value
"mem-rgn"	QUICC_PROP_MEM_RGN	QuiccPropMemRgn	m	
"intr"	QUICC_PROP_INTR	QuiccPropIntr	m	
"channel"	QUICC_PROP_CHANNEL	QuiccPropChannel	m	
"ether-addr"	ETHER_PROP_ADDR	EtherPropAddr	o	ENVIRON-MENT
"tx-desc-nb"	SCCETHER_PROP_TX_DESC_NB	ScsEtherPropDescNb	m	8
"rx-desc-nb"	SCCETHER_PROP_RX_DESC_NB	ScsEtherPropDescNb	m	8
"tx-buf-size"	SCCETHER_PROP_TX_BUF_SZ	ScsEtherPropBufSz	o	1514 bytes
"rx-buf-size"	SCCETHER_PROP_RX_BUF_SZ	ScsEtherPropBufSz	o	1514 bytes

The QUICC_PROP_MEM_RGN property specifies the quicc bus memory mapped regions used by the driver:

- index SCCETHER_REG is used to map scc registers
- index SCCETHER_PARAM is used to map scc parameters

The QUICC_PROP_INTR property specifies the interrupt used by the device.

The QUICC_PROP_CHANNEL property specifies the channel used by the device.

The `ETHER_PROP_ADDR` property specifies the Ethernet address. If the property is not present, the driver tries to get the Ethernet address from the `"ETHER_ADDR"` environment variable, and creates the property in the device node.

The `SCCETHER_PROP_TX_DESC_NB` specifies the number of descriptors to be allocated in the transmit ring. By default, 8 descriptors are used.

The `SCCETHER_PROP_RX_DESC_NB` specifies the number of descriptors to be allocated in the receive ring. By default, 8 descriptors are used.

The `SCCETHER_PROP_TX_BUF_SZ` specifies the size of the DMA buffers to be allocated and bound to each descriptor in the transmit ring. By default, each buffer has a size of 1514 bytes, which is the maximum size of an Ethernet frame.

The `SCCETHER_PROP_RX_BUF_SZ` specifies the size of the DMA buffers to be allocated and bound to each descriptor in the receive ring. By default, each buffer has a size of 1514 bytes, which is the maximum size of an Ethernet frame. Thus, the total size of DMA memory allocated by the driver will be:

- `SCCETHER_PROP_TX_DESC_NB * SCCETHER_PROP_TX_BUF_SZ` for transmission, and
- `SCCETHER_PROP_RX_DESC_NB * SCCETHER_PROP_RX_BUF_SZ` for reception.

Note that the driver is capable of sending and receiving an Ethernet frame using multiple descriptors in a ring. But it is not able to share the memory buffer, bound to one descriptor, between multiple Ethernet frames. In other words, configuring buffer sizes with big values will lead to poor memory usage, with small Ethernet frames.

The table below lists device node properties added by the `sccEther` driver, which may be used by the driver clients.

Name	Alias	Type
"link-throughput"	<code>ETHER_PROP_THROUGHPUT</code>	<code>EtherPropThroughput</code>
"ether-addr"	<code>ETHER_PROP_ADDR</code>	<code>EtherPropAddr</code>

The `ETHER_PROP_THROUGHPUT` property specifies the throughput of the Ethernet link.

The `ETHER_PROP_ADDR` property specifies the Ethernet address. Note that the driver adds this property to the device node only if it does not already exist.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`ether(9DDI)`, `quicc(9DDI)`

NAME	sccUart – quicc scc controller uart device driver				
SYNOPSIS	<p>drv/src/uart/scc/sccUart.c - driver code</p> <p>drv/src/uart/scc/sccUartProp.h - driver specific properties</p>				
FEATURES	DRV				
DESCRIPTION	<p>The sccuart driver implements the uart device driver interface. The driver works on a quicc “serial communication controller” device connected to an RS232 port.</p> <p>The driver uses the quicc bus driver interface provided by a parent bus driver.</p> <p>The sccuart driver does not provide the <code>drv_probe()</code> routine. In other words, the sccuart driver does not enumerate the quicc bus in order to detect an scc device and create an associated device node. When the sccuart driver is used, associated device nodes are created either statically by a boot program or dynamically by a quicc bus enumerator driver.</p> <p>The sccuart driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties in order to recognize an sccuart compatible device. The routine checks for a <code>QUICC_PROP_CHANNEL</code> property matching an scc device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node attaching a <code>PROP_DRIVER</code> property to it. The property value specifies the sccuart driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. In other words, via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver asking it to invoke the <code>drv_init()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if a <code>PROP_DRIVER</code> or a <code>PROP_DBG_LINK</code> property is already present in the device node. In other words, the <code>drv_bind()</code> routine does not override existing driver-to-device binding.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports the unloading driver component. This allows the driver component to be unloaded if it is no longer in use. Note that the driver component can be unloaded only if it has been dynamically loaded at run time. In other words, a resident driver component (embedded in the system bootable image) cannot be unloaded although the unloading is supported by the driver.</p> <p>The driver supports the <code>QUICC_SYS_SHUTDOWN</code> and <code>QUICC_DEV_SHUTDOWN</code> bus events specified by the quicc bus driver interface.</p> <p>The table below summarizes characteristics of the sccuart uart driver.</p> <table> <tr> <td>driver name:</td><td>"sun@quicc-scc-uart"</td></tr> <tr> <td>hardware:</td><td>quicc /scc connected to RS232</td></tr> </table>	driver name:	"sun@quicc-scc-uart"	hardware:	quicc /scc connected to RS232
driver name:	"sun@quicc-scc-uart"				
hardware:	quicc /scc connected to RS232				

exported interface:	"uart" (UART_CLASS)
exported interface version:	0 (UART_VERSION_INITIAL)
imported parent interface:	"quicc" (QUICC_CLASS)
minimal parent interface version:	0 (UART_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver to device binding:	supported (on channel basis)
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	not supported

The table below lists device node properties used by the sccuart driver.

Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the "Default Value" column shows the default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default Value
"mem-rgn"	QUICC_PROP_MEM_RGN	QuiccPropMemRgn[]	m	
"brg"	QUICC_PROP_BRG	QuiccPropBrg	m	
"intr"	QUICC_PROP_INTR	QuiccPropIntr	m	
"channel"	QUICC_PROP_CHANNEL	QuiccPropChannel	m	
"io-pin"	QUICC_PROP_IO_PIN	QuiccPropIoPin[]	m	
"rx-fifo-sz"	SCCUART_PROP_RX_FIFO_SZ	SzUartPropFifoSz	o	8 bytes
"tx-fifo-sz"	SCCUART_PROP_TX_FIFO_SZ	SzUartPropFifoSz	o	2 bytes

The QUICC_PROP_MEM_RGN property specifies the memory regions used to map the scc controller registers and parameters: The property value is an array of QuiccPropMemRgn structures:

- index SCCUART_REG is used to map scc registers
- index SCCUART_PARAM is used to map scc parameters

The `QUICC_PROP_BRG` property specifies the baud rate generator used as the Rx and Tx clocks.

The `QUICC_PROP_INTR` property specifies the interrupt used by the scc controller.

The `QUICC_PROP_CHANNEL` property specifies the cpm scc channel used.

The `QUICC_PROP_IO_PIN` property specifies the parallel port I/O pins to be used for uart control and modem signals. The property value is an array of `QuiccPropIoPin` structures:

- index `SCCUART_CTS` defines CTS input signal I/O pin and interrupt source
- index `SCCUART_DSR` defines DSR input signal I/O pin and interrupt source
- index `SCCUART_RI` defines RI input signal I/O pin and interrupt source
- index `SCCUART_DTR` defines DTR output signal I/O pin and interrupt source
- index `SCCUART_RTS` defines RTS output signal I/O pin and interrupt source

Note that all entries of the property are mandatory. If a given signal is not physically connected to a given platform, the "pin" field for the entry associated to this signal should be zero. In such a case the driver will not manage this signal.

The `SCCUART_PROP_RX_FIFO_SZ` property specifies the size in bytes of the receive FIFO buffer.

The `SCCUART_PROP_TX_FIFO_SZ` property specifies the size in bytes of the transmit FIFO buffer.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`uart(9DDI)`, `quicc(9DDI)`

NAME	simba – Sun advanced pci-to-pci bridge driver
SYNOPSIS	<p>drv/src/pci/simba/simba.h - simba hardware related constants</p> <p>drv/src/pci/simba/simba.c - driver code</p> <p>drv/src/pci/simba/simbaProp.h - driver specific properties</p>
FEATURES	DRV
DESCRIPTION	<p>The simba driver implements the pci bus interface.</p> <p>The driver works with the Sun UltraSPARC II CPU. The driver uses the UltraSPARC specific DKI interface provided by the kernel. Thus, the driver is designed to work with the Sun UltraSPARC II CPU.</p> <p>The simba driver does not provide the <code>drv_probe()</code> routine. In other words, the simba driver does not enumerate the pci bus, nor does it detect a simba device or create an associated device node. When the simba driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate pci bus enumerator driver.</p> <p>The simba driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties and thus, identifies a simba compatible device. The routine checks for the pci vendor and device identifiers matching a simba compatible device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the simba driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. Via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver, asking it to invoke the <code>drv_int()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if the <code>PROP_DRIVER</code> property is already present in the device node, as the <code>drv_bind()</code> routine will not override existing driver-to-driver binding.</p> <p>The driver does not implement the <code>drv_unload()</code> entry. This means that the driver component cannot be unloaded even if it has been dynamically loaded at run time.</p> <p>The driver supports the <code>DKI_SYS_SHUTDOWN</code> event specified by the common DKI interface.</p> <p>The SIMBA driver uses the RIC DDI to attach/detach PCI interrupts.</p> <p>The table below summarizes characteristics of the simba pci bus driver.</p>

driver name:	"sun: (pci, ric)-simba-pci"
hardware:	Sun simba pci host bridge
exported interface:	"pci" (PCI_CLASS)
exported interface version:	0 (PCI_VERSION_INITIAL)
imported parent interface:	"pci" (PCI_CLASS)
minimal parent interface version:	0 (PCI_VERSION_INITIAL)
imported parent interface:	"ric" (RIC_CLASS)
minimal parent interface version:	0 (RIC_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	not supported
system (emergency) shut-down:	supported
normal device shut-down:	not supported
hot-plug (surprise) device removal:	not supported

The table below lists device node properties used by the simba driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"bus-num"	PCI_PROP_BUS_NUM	PciPropBusNum	m	
"dev-num"	PCI_PROP_DEV_NUM	PciPropDevNum	m	
"func-num"	PCI_PROP_FUNC_NUM	PciPropFuncNum	m	
"simba-ric-map"	SIMBA_PROP_RIC_MAP	SimbaPropRicMap	m	

The `PCI_PROP_VEND_ID` property specifies the Sun pci vendor identifier (must be 0x108e).

The `PCI_PROP_DEV_ID` property specifies the simba pci device identifier (must be 0x5000).

The `PCI_PROP_BUS_NUM` property specifies the secondary pci bus number.

The `PCI_PROP_DEV_NUM` property specifies the PCI device number of the device.

The `PCI_PROP_FUNC_NUM` property specifies the PCI function number of the device.

For each PCI device number, the `SIMBA_PROP_RIC_MAP` property specifies a mapping of PCI interrupt pins (`INTA#`, `INTB#`, `INTC#`, `INTD#`) into a RIC interrupt number offset (ranging from `0x0` to `0x3f`). The `SIMBA_RIC_OFF_INVALID` value can be used if the corresponding offset is not defined. This property allows the SIMBA to take into account the various PCI interrupt RIC offset mappings for different platforms.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`pci(9DDI)` `ric(9DDI)`

NAME	smc1660 – implements the isa Ethernet device driver interface.												
SYNOPSIS	<p>drv/src/net/ether/smc1660/smc1660.h - smc1660 hardware constants</p> <p>drv/src/net/ether/smc1660/smc1660.c - driver code</p> <p>drv/src/net/ether/smc1660/smc1660Prop.h - driver specific properties</p>												
FEATURES	DRV												
DESCRIPTION	<p>The smc1660 Ethernet driver implements the ether device driver interface.</p> <p>The smc1660 Ethernet driver supports the following boards: WD80X3, SmcElite16, SmcEtherEZ, SmcEliteUltra which are based on the DP8390/NS32490D Network Interface Controller.</p> <p>The driver uses the isa bus driver interface provided by a parent bus driver. Thus, the driver may be applied to any isa bus using this interface.</p> <p>The smc1660 driver does not provide the <code>drv_probe()</code> routine. In other words, the smc1660 driver does not enumerate the bus, nor does it detect an smc1660 device or create an associated device node. When the smc1660 driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The driver does not provide the <code>drv_bind()</code> routine. In other words, the driver does not support dynamic driver-to-device binding. When the smc1660 driver is used, the driver should be explicitly bound to the device using the <code>PROP_DRIVER</code> property. It is assumed that the device-to-driver binding is done either by a device node creator or by a separate binder driver. Such a driver-to-device binder could be developed for this particular bus architecture.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p> <p>The Table below summarizes characteristics of the smc1660 Ethernet driver.</p> <table> <tr> <td>driver name:</td><td>"sun:isa-smc1600-ether"</td></tr> <tr> <td>hardware:</td><td>SMC1660 Ethernet controller</td></tr> <tr> <td>exported interface:</td><td>"ether" (ETHER_CLASS)</td></tr> <tr> <td>exported interface version:</td><td>0 (ETHER_VERSION_INITIAL)</td></tr> <tr> <td>imported parent interface:</td><td>"isa" (ISA_CLASS)</td></tr> <tr> <td>minimal parent interface version:</td><td>0 (ISA_VERSION_INITIAL)</td></tr> </table>	driver name:	"sun:isa-smc1600-ether"	hardware:	SMC1660 Ethernet controller	exported interface:	"ether" (ETHER_CLASS)	exported interface version:	0 (ETHER_VERSION_INITIAL)	imported parent interface:	"isa" (ISA_CLASS)	minimal parent interface version:	0 (ISA_VERSION_INITIAL)
driver name:	"sun:isa-smc1600-ether"												
hardware:	SMC1660 Ethernet controller												
exported interface:	"ether" (ETHER_CLASS)												
exported interface version:	0 (ETHER_VERSION_INITIAL)												
imported parent interface:	"isa" (ISA_CLASS)												
minimal parent interface version:	0 (ISA_VERSION_INITIAL)												

device probing (auto-detection):	not supported
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	not supported

The Table below lists device node properties used by the smc1660 driver.

Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the "Default Value" column shows the default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default Value
"intr"	ISA_PROP_INTR	IsaPropIntr	m	
"io-regs"	ISA_PROP_IO_REGS	IsaPropIoRegs	m	
"mem-rgn"	ISA_PROP_MEM_RGN	IsaPropMemRgn	m	

The ISA_PROP_INTR property specifies the isa interrupt used by the device.

The ISA_PROP_IO_REGS property specifies the smc1660 I/O registers range.

The ISA_PROP_MEM_RGN property specifies the smc1660 memory range.

The Table below lists device node properties added by the smc1660 driver, which may be used by the driver clients.

Name	Alias	Type
"link-throughput"	ETHER_PROP_THROUGHPUT	EtherPropThroughput
"ether-addr"	ETHER_PROP_ADDR	EtherPropAddr

The ETHER_PROP_THROUGHPUT property specifies the throughput of the Ethernet link. The smc1660 driver adds this property to the device node and initializes it with the throughput of the Ethernet link to which the device is connected.

The `ETHER_PROP_ADDR` property specifies the device Ethernet address. The `smc1660` driver adds this property to the device node and initializes it with the Ethernet address which is in the controller ROM.

`ipctest` is supported by the `smc1660` driver.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`ether(9DDI)`, `isa(9DDI)`

NAME	smcuart – quicc smc controller uart device driver				
SYNOPSIS	<p>drv/src/uart/smc/smcUart.c - driver code</p> <p>drv/src/uart/smc/smcUartProp.h - driver specific properties</p>				
FEATURES	DRV				
DESCRIPTION	<p>The smcuart driver implements the uart device driver interface. The driver works on a quicc “serial management controller” device in uart mode.</p> <p>The driver uses the quicc bus driver interface provided by a parent bus driver.</p> <p>The smcuart driver does not provide the <code>drv_probe()</code> routine. In other words, the smcuart driver does not enumerate the quicc bus in order to detect an smc device and create an associated device node. When the smcuart driver is used, associated device nodes are created either statically by a boot program or dynamically by a quicc bus enumerator driver.</p> <p>The smcuart driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties in order to recognize an smcuart compatible device. The routine checks for a <code>QUICC_PROP_CHANNEL</code> property matching an smc device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node attaching a <code>PROP_DRIVER</code> property to it. The property value specifies the smcuart driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. In other words, via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver asking it to invoke the <code>drv_init()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if a <code>PROP_DRIVER</code> or a <code>PROP_DBG_LINK</code> property are already present in the device node. In other words, the <code>drv_bind()</code> routine does not override existing driver-to-device binding.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports the unloading driver component. This allows the driver component to be unloaded if it is no longer in use. Note that the driver component can be unloaded only if it has been dynamically loaded at run time. In other words, a resident driver component (embedded in the system bootable image) cannot be unloaded although unloading is supported by the driver.</p> <p>The driver supports the <code>QUICC_SYS_SHUTDOWN</code> and <code>QUICC_DEV_SHUTDOWN</code> bus events specified by the quicc bus driver interface.</p> <p>The table below summarizes characteristics of the smcuart uart driver.</p> <table> <tr> <td>driver name:</td><td>"sun@quicc-smc-uart"</td></tr> <tr> <td>hardware:</td><td>quicc /smc device</td></tr> </table>	driver name:	"sun@quicc-smc-uart"	hardware:	quicc /smc device
driver name:	"sun@quicc-smc-uart"				
hardware:	quicc /smc device				

exported interface:	"uart" (UART_CLASS)
exported interface version:	0 (UART_VERSION_INITIAL)
imported parent interface:	"quicc" (QUICC_CLASS)
minimal parent interface version:	0 (UART_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver to device binding:	supported (on channel basis)
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	not supported

The table below lists device node properties used by the smcuart driver.

Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the "Default Value" column shows the default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default Value
"mem-rgn"	QUICC_PROP_MEM_RGN	QuiccPropMemRgn	m	
"brg"	QUICC_PROP_BRG	QuiccPropBrg	m	
"intr"	QUICC_PROP_INTR	QuiccPropIntr	m	
"channel"	QUICC_PROP_CHANNEL	QuiccPropChannel	m	
"rx-fifo-sz"	SMCUART_PROP_RX_FIFO_SZ	SmcUartPropFifoSz	o	8 bytes
"tx-fifo-sz"	SMCUART_PROP_TX_FIFO_SZ	SmcUartPropFifoSz	o	2 bytes

The QUICC_PROP_MEM_RGN property specifies the quicc bus memory mapped regions used by the driver:

- Index SMCUART_REG is used to map smc registers.
- Index SMCUART_PARAM is used to map smc parameters.
- Index SMCUART_BASE is used to map the smc parameter base pointer in case parameters are accessed indirectly. In such cases the driver dynamically

allocates DPRAM memory for the parameters and makes the base pointer point to the allocated memory. Otherwise, if parameters must be directly mapped, the “size” field of this property element must be zero.

The QUICC_PROP_INTR property specifies the interrupt used by the device.

The QUICC_PROP_CHANNEL property specifies the channel used by the device.

The QUICC_PROP_BRG property specifies the baud rate generator to be used as clocks in the device.

The SMCUART_PROP_RX_FIFO_SZ property specifies the size in bytes of the receive FIFO.

The SMCUART_PROP_TX_FIFO_SZ property specifies the size in bytes of the transmit FIFO.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

uart(9DDI), quicc(9DDI)

NAME	tbDec – PowerPC timebase and decrementer timer device driver
SYNOPSIS	<p>ddi/ddi_f.h - PowerPC specific properties</p> <p>drv_f/src/timer/tbDec/tbDec.c - driver code</p> <p>drv_f/src/timer/tbDev/tbDecProp.h - driver specific properties</p>
FEATURES	DRV
DESCRIPTION	<p>The tbDec timer driver implements the timer device driver interface.</p> <p>The driver works with PowerPC on-chip timebase and decrementer registers. Note that only the free-run mode of the <code>timer</code> interface is available for the PowerPC timebase, thus allowing this driver to be shared.</p> <p>The driver uses the PowerPC specific dki interface provided by the kernel. Thus, the driver only works with PowerPC family products.</p> <p>The tbDec driver does not provide the <code>drv_probe()</code> routine. In other words, the tbDec driver does not enumerate the bus nor does it detect a tbDec device or create an associated device node. When the tbDec driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for the particular bus architecture.</p> <p>The tbDec driver provides the <code>drv_bind()</code> routine. This routine examines the device node name property (<code>PROP_NODE</code>) in order to recognize a tbDec compatible device. The routine checks whether the device node name matches a pre-defined device name. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the tbDec driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. In other words, via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver asking it to invoke the <code>drv_init()</code> routine on that device. The <code>drv_bind()</code> routine does nothing if the <code>PROP_DRIVER</code> property is already present in the device node. In other words, the <code>drv_bind()</code> routine will not override existing driver-to-device binding.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p> <p>The driver supports the <code>DKI_SYS_SHUTDOWN</code> event specified by the common dki interface.</p> <p>The Table below summarizes the characteristics of the tbDec <code>TIMER</code> driver:</p>

driver name:	"sun:powerpc-(timebase,dec)-timer"
hardware:	PowerPC timebase and decrementer
exported interface:	"timer" (TIMER_CLASS)
exported interface version:	0 (TIMER_VERSION_INITIAL)
imported parent interface:	"powerpc" (FDKI_CLASS)
minimal parent interface version:	0 (FDKI_VERSION_INITIAL)
device probing (auto-detection):	not supported
device-to-driver binding:	supported (on a node name basis)
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	not support

The Table below lists device node properties used by the tbDec driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	type	m/o	default value
"tb-freq"	PPC_PROP_TIME_BASE_FREQ	PpcPropTimeBaseFreq	o	16665000 Hz

The PPC_PROP_TIME_BASE_FREQ property specifies the timebase and decrementer clock frequency in Hz. By default the frequency is set to 16665000 Hz.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

timer(9DDI)

NAME	universe – TUNDRA Universe PCI to VME bus bridge driver
SYNOPSIS	<div> <div>drv/src/vme/universe/universe.c</div> <div>- driver code</div> </div> <div> <div>drv/src/vme/universe/universe.h</div> <div>- hardware specific properties</div> </div> <div> <div>drv/src/vme/universe/universeProp.h</div> <div>- driver specific properties</div> </div>
FEATURES	DRV
DESCRIPTION	<p>The universe driver implements:</p> <ul style="list-style-type: none"> ■ the VME bus driver interface, and ■ the BUSCOM (local) device driver interface <p>The driver works on a TUNDRA Universe PCI to VME bus bridge controller.</p> <p>The driver uses the PCI bus device driver interface provided by a parent bus driver.</p> <p>The universe driver does not provide the <code>drv_probe()</code> routine. In other words, the universe driver does not enumerate the bus in order to detect a compatible device and create an associated device node. When the universe driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. This type of enumerator driver could be developed for a particular (PCI/VME) bus architecture.</p> <p>The universe driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties in order to recognize a universe compatible device. The routine checks for PCI vendor/device identifier properties that match the universe device ones. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to it. The property value specifies the universe driver name. The parent bus driver uses this property to determine the name of the driver servicing the device. In other words, it is through the <code>PROP_DRIVER</code> property that the driver gives its name to the parent bus driver when asking to invoke the <code>drv_init()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if a <code>PROP_DRIVER</code> property is already present in the device node; <code>drv_bind()</code> does not override an existing driver-to-device binding.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports the driver component unloading. This allows the driver component to be unloaded if it is no longer being used. Note that the driver component can be unloaded only if it has been dynamically loaded at run time. In other words, a resident driver component (embedded in the system bootable image) cannot be unloaded although unloading is supported by the driver.</p>

The driver supports all events specified by the PCI bus driver interface except for VGA palette related events.

The table below summarizes characteristics of the universe bus driver.

driver name:	"sun:pci-universe-vme"
hardware:	TUNDRA Universe PCI-to-VME bridge
exported interface:	"vme" (VME_CLASS)
exported interface version:	0 (VME_VERSION_INITIAL)
imported parent interface:	"pci" (PCI_CLASS)
minimal parent interface version:	0 (PCI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver-to-device binding:	supported (on PCI vendor/device ids)
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	supported

The table below lists device node properties used by the universe vme bus driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

name	alias	type	m/o	default value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"dev-num"	PCI_PROP_DEV_NUM	PciPropDevNum	m	
"func-num"	PCI_PROP_FUNC_NUM	PciPropFuncNum	m	
"intr"	PCI_PROP_INTR	PciPropIntr	m	
"io-regs"	PCI_PROP_IO_REGS	PciPropIoRegs	m	
"vme-map"	UNIVERSE_PROP_VME_MAP	UniversePropVmeMap	m	
"sysfail"	UNIVERSE_PROP_SYSFAIL	none	o	not managed

"dma-burst"	VME_PROP_DMA_BURST	VmePropDmaBurst	o	no maximum
"dma-off"	VME_PROP_DMA_OFF_TIMER	VmePropDmaOffTimer	o	0

The PCI_PROP_VEND_ID property specifies the TUNDRA pci vendor identifier (must be 0x10E3).

The PCI_PROP_DEV_ID property specifies the universe pci device identifier (must be 0x0000).

The PCI_PROP_DEV_NUM and PCI_PROP_FUNC_NUM properties specify the configuration space address of the device. That is, the device and function numbers to which the device is connected on the PCI bus.

The PCI_PROP_INTR property specifies the pci interrupt(s) used by the universe device.

The PCI_PROP_IO_REGS property specifies the pci address space allocated to the universe's I/O registers.

The UNIVERSE_PROP_VME_MAP property specifies the universe configuration for pci bus slave map decoders. This property is an array. Each element in this array defines a pci space address range which will be translated into vme bus cycles by the universe. The maximum number of elements in the array depends on the universe device version. The first universe version defines array indexes in the range [0,3]. The universe II version accepts indexes in the range [0,7].

Each element in the array defines:

- a PCI region:
 - PCI address space (I/O, memory)
 - start address
 - region size
- a VME region:
 - VME address space (short, standard, extended)
 - start address
 - region size
 - region mapping attributes (AM codes)

The universe driver assumes that the UNIVERSE_PROP_VME_MAP property sums up VME space resources allocated to all universe's child devices on the underlying VME bus.

The UNIVERSE_PROP_SYSFAIL property indicates whether the VME SYSFAIL signal is handled by the driver or not. By default, the SYSFAIL signal is ignored.

The VME_PROP_DMA_BURST and VME_PROP_DMA_OFF_TIMER properties specify the maximum DMA burst size and minimum VME bus off timer, respectively. Together, these properties define the behavior of the universe during a DMA transfer by specifying the:

- size boundary at which the DMA transfer will be split into multiple bursts (no maximum by default), and the
- minimum time that the VME bus will be unused between bursts (minimum off time is zero, by default).

The table below lists device node properties used by the universe buscom (local) device driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

name	alias	type	m/o	default value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"buscom-remote"	BUSCOM_PROP_REMOTE	no value	m	
"buscom-monarch"	BUSCOM_PROP_MONARCH	no value	m	
"buscom-id"	BUSCOM_PROP_ID	BusComPropId	m	
"io-regs"	VME_PROP_IO_REGS	VmePropIoRegs	m	
"mem-rgn"	VME_PROP_MEM_RGN	VmePropMemRgn	m	

The BUSCOM_PROP_REMOTE property flags the device node as a "remote" universe device (indicating that it is located on another VME host board). This property must be absent for the universe local communication driver to run on that device.

The BUSCOM_PROP_MONARCH property flags the device node as being the bus communication "monarch". If present, this property indicates that the associated device must be booted and made accessible on the vme bus before all other devices. Typically, on the vme bus, this device is the vme bus system controller. This property should be used by the universe client application to choose the bus device on which any shared vital data should be located.

The BUSCOM_PROP_ID is the unique identifier (UID) of the device on the underlaying vme bus. It can be used as a location identifier by the universe client application that is implementing communication protocols over the vme bus.

The VME_PROP_IO_REGS property specifies the vme address range allocated to the local universe I/O registers. The universe local communication driver will respond to the addresses, specified in the range, allowing univRemCom drivers running on other CPU boards to access local universe registers through the VME bus.

The VME_PROP_MEM_RGN property specifies the vme address range dedicated to the local universe bus communication memory. The universe local communication driver will respond to the addresses, specified in the range, allowing univRemCom drivers running on other boards to access local communication memory through the VME bus.

RESTRICTIONS

The universe driver does not support dynamic resource allocation.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`pci(9DDI)`, `vme(9DDI)`

NAME	univRemCom – TUNDRA Universe vme bus remote communication driver
SYNOPSIS	<p>drv/src/buscom/universe/univRemCom.c - driver code</p> <p>drv/src/buscom/universe/univRemComProp.h - driver specific properties</p>
FEATURES	DRV
DESCRIPTION	<p>The univRemCom driver implements the BUSCOM (remote) device driver interface. The driver works on a TUNDRA Universe PCI to VME bus bridge controller.</p> <p>The driver uses the VME bus driver interface provided by a parent bus driver.</p> <p>The univRemCom driver does not provide the <code>drv_probe()</code> routine. In that the univRemCom driver does not enumerate the bus in order to detect a compatible device and create an associated device node. When the univRemCom driver is used, associated device nodes should be created either statically, by a boot program, or dynamically, by a separate VME bus enumerator driver.</p> <p>The univRemCom driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties in order to recognize a universe compatible device. The routine checks for PCI vendor/device identifier properties matching the TUNDRA universe device ones, and for a <code>BUSCOM_PROP_REMOTE</code> property, indicating that the node corresponds to a remote universe device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to it. The property value specifies the univRemCom driver name. The parent bus driver uses this property to determine the name of the driver servicing the device. It is through the <code>PROP_DRIVER</code> property that the driver gives its name to the parent bus driver to ask it to invoke the <code>drv_init()</code> routine for that device. Note that the <code>drv_bind()</code> routine does nothing if a <code>PROP_DRIVER</code> property is already present in the device node; the <code>drv_bind()</code> routine does not override an existing driver-to-device binding.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports driver component unloading. This allows the driver component to be unloaded if it is no longer needed. Note that the driver component can only be unloaded if it has been dynamically loaded at run time. In that a resident driver component (embedded in the system bootable image) cannot be unloaded although unloading is supported by the driver.</p> <p>The driver supports all events specified by the VME bus driver interface except for the <code>VME_ACFAIL</code> and <code>VME_ARBITRATION_TIMEOUT</code> events.</p> <p>The table below summarizes characteristics of the univRemCom driver.</p>

driver name:	"sun:vme-universe-buscom <remote>"
hardware:	TUNDRA Universe PCI-to-VME bridge
exported interface:	"buscom" (BUSCOM_CLASS)
exported interface version:	0 (BUSCOM_VERSION_INITIAL)
imported parent interface:	"vme" (VME_CLASS)
minimal parent interface version:	0 (VME_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver-to-device binding:	supported (on PCI vendor/device ids)
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	supported

The table below lists device node properties used by the universe driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default Value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"buscom-remote"	BUSCOM_PROP_REMOTE	no value	m	
"buscom-monarch"	BUSCOM_PROP_MONARCH	no value	m	
"buscom-id"	BUSCOM_PROP_ID	BusComPropId	m	
"io-regs"	VME_PROP_IO_REGS	VmePropIoRegs	m	
"mem-rgn"	VME_PROP_MEM_RGN	VmePropMemRgn	m	

The PCI_PROP_VEND_ID property specifies the TUNDRA PCI vendor identifier (must be 0x10E3).

The PCI_PROP_DEV_ID property specifies the Universe PCI device identifier (must be 0x0000).

The `BUSCOM_PROP_REMOTE` property flags the device node as a "remote" universe device (that is, a device located on another VME host board). This property must be present for the `univRemCom` to run on that device.

The `BUSCOM_PROP_MONARCH` property flags the device node as being the bus communication "monarch". If present, this property indicates that the associated device must be booted and made accessible on the VME bus before all other devices. Typically, on the VME bus this device is the VME bus system controller. This property should be used by the `univRemCom` client application to choose the bus device on which any shared vital data should be located.

The `BUSCOM_PROP_ID` is the unique identifier (UID) of the device on the underlying VME bus. It can be used as a location identifier by the `univRemCom` client application implementing communication protocols over the VME bus.

The `VME_PROP_IO_REGS` property specifies the VME address range allocated to the remote universe I/O registers. This address range is used by the `univRemCom` driver to access universe registers through the VME bus.

The `VME_PROP_MEM_RGN` property specifies the VME address range dedicated to the remote universe bus communication memory. This property defines the VME address space which will be mapped to a memory space located on the remote VME CPU board. The address range is used by the `univRemCom` driver to access communication memory located on the remote board through the VME bus.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`vme(9DDI)`

NAME	vt82c586 – vt82c586 VIA Technologies pci-to-isa bridge, isa bus driver
SYNOPSIS	<div>drv/src/isa/vt82c586/vt82c586.h - VT82C586 hardware related constants</div> <div>drv/src/isa/vt82c586/vt82c586.c - driver code</div> <div>drv/src/isa/vt82c586/vt82c586PROP.h - driver specific properties</div>
FEATURES	DRV
DESCRIPTION	<p>The vt82c586 pci-to-isa bridge driver implements:</p> <ul style="list-style-type: none"> ■ The common bus driver interface ■ The isa bus driver interface <p>The driver uses the pci bus driver interface provided by a parent pci bus driver. Thus, the driver may be applied to any bus driver implementing the pci bus interface.</p> <p>The vt82c586 driver does not provide the <code>drv_probe</code> routine. In other words, the vt82c586 driver does not enumerate the pci bus, nor does it detect a vt82c586 device or create an associated device node. When the vt82c586 driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate pci bus enumerator driver.</p> <p>The vt82c586 driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties and thus, identifies a vt82c586 compatible device. The routine checks for the pci vendor and device identifiers matching a vt82c586 compatible device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the vt82c586 driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. Via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver, asking it to invoke the <code>drv_int()</code> routine on that device. Note that the <code>drv_bind()</code> routine does nothing if the <code>PROP_DRIVER</code> property is already present in the device node, as the <code>drv_bind()</code> routine will not override existing driver-to-driver binding.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p> <p>The driver supports all bus events specified by the pci bus driver interface. As a consequence, the driver may be used with a hot-pluggable pci bus (for example, CompactPCI).</p>

The table below summarizes characteristics of the vt82c586 isa bus driver.

driver name:	"sun:pci-vt82c586-(bus, isa)"
hardware:	vt82c586 pci-to-isa bridge
exported interface:	"bus", "isa" (BUS_CLASS, ISA_CLASS)
exported interface version:	0 (ISA_VERSION_INITIAL)
imported parent interface:	"pci" (PCI_CLASS)
minimal parent interface version:	0 (PCI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver-to-device binding:	supported (on PCI dev/vend id basis)
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	supported

The Table below lists device node properties used by the vt82c586 driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"dev-num"	PCI_PROP_DEV_NUM	PciPropDevNum	m	
"func-num"	PCI_PROP_FUNC_NUM	PciPropFuncNum	m	
"intr"	PCI_PROP_INTR	PciPropIntr	m	
"conf"	VT82_ISA_PROP_CONF	Vt82IsaPropConf	o	see text below

Note that the driver does not use any PCI_PROP_IO_REGS properties because its I/O range cannot be relocated. The bridge assumes zero based pci/isa I/O addresses.

The `PCI_PROP_VEND_ID` property specifies the VIA Technologies vendor identifier (must be 0x1106).

The `PCI_PROP_DEV_ID` property specifies the vt82C586 device identifier (must be 0x0586).

The `PCI_PROP_DEV_NUM`, and `PCI_PROP_FUNC_NUM` properties specify the configuration space address of the device.

The `PCI_PROP_INTR` property specifies the pci interrupt used by the device. Note that this interrupt is used for all 16 isa IRQ's.

The `VT82_ISA_PROP_CONF` property specifies the configuration of the vt82C586 pci-to-isa bridge. The `Vt82IsaPropConf` structure consists of images of the vt82C586 configuration registers. If the `VT82_ISA_PROP_CONF` property is specified, the driver will program configuration registers according to the images given by the `Vt82IsaPropConf` structure. Otherwise, the default values will be used. The Table below shows the `Vt82IsaPropConf` fields and the default values used for the bridge configuration when the `VT82_ISA_PROP_CONF` property is not specified.

type	name	default	addr	description
uint8_f	bct1	00	40	isa Bus Control Register
uint8_f	tmode	20	41	isa Test Mode Register
uint8_f	cct1	02	42	isa Clock Control Register
uint8_f	rct1	00	43	ROM Decode Control Register
uint8_f	kct1	00	44	Keyboard Controller Control Register
uint8_f	dct1	00	45	Type F DMA Control Register
uint8_f	mct1i	01	46	Miscellaneous
uint8_f	ide_ir	04	4a	ide Interrupt Routing Register

uint8_f	dma_holebot	00	4c	pci Memory Hole Bottom Addr Register
uint8_f	dma_holetop	00	4d	pci Memory Hole Top Addr Register
uint16_f	dma_ct1	f100	4e	DMA/Master Mem Acc Control Register
uint8_f	pirq_sel	00	54	pci IRQ Edge/Level Selection Register
uint8_f	ir1	00	55	PnP Routing for MIRQD Register
uint8_f	ir2	a5	56	PnP Routing for PIRQA/B Register
uint8_f	ir3	00	57	PnP Routing for MIRQ1/PIRQC Register
uint8_f	ir4	00	58	PnP Routing for MIRQ2 Register
uint8_f	mirq	00	59	MIRQ Pin Configuration Register
uint8_f	xd	00	5a	XD Power-On Strap Option Register
uint8_f	rtctm	00	5b	Internal RTC Test Mode Register
uint8_f	dma	00	5c	DMA Control Register

RESTRICTIONS

See the vt82c586b pci Integrated Peripheral Controller Datasheet for programming the bus bridge configuration registers (<http://www.via.com.tw>).

The current version of the vt82c586 driver does not support the dynamic assignment of the isa bus resources.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`bus(9DDI)`, `pci(9DDI)` `isa(9DDI)`

NAME	w83c553 – winbond pci/isa bridge, isa bus driver
SYNOPSIS	<div>drv/src/isa/w83c553/w83c553.h - w83c553 hardware related constants</div> <div>drv/src/isa/w83c553/w83c553.c - driver code</div> <div>drv/src/isa/w83c553/w83c553Prop.h - driver specific properties</div>
FEATURES	DRV
DESCRIPTION	Implements the common bus and isa bus driver interfaces.
EXTENDED DESCRIPTION	<p>The w83c553 bridge driver implements:</p> <ul style="list-style-type: none"> ■ The common bus driver interface. ■ The ISA bus driver interface. <p>The driver works on a winbond w83c553F PCI to ISA bus bridge.</p> <p>The driver uses the PCI bus driver interface provided by a parent PCI bus driver. Thus, the driver may be applied to any bus driver implementing the PCI bus interface.</p> <p>The driver provides a very simple <code>drv_probe()</code> entry. In other words, the driver does not implement true device probing/auto-detection. Rather, it searches the device tree, looking for its device name or its PCI vendor/device identifiers, to find device nodes to which it can be bound.</p> <p>When the w83c553 driver is used, the associated device node has to be created either statically or by another driver. For instance, a probe-only driver may be developed for a PCI bus architecture in order to dynamically discover a w83c553 device residing on the PCI bus and to create an associated device node.</p> <p>The w83c553 driver provides the <code>drv_bind()</code> routine. This routine examines the device node properties and thus, identifies a w83c553 compatible device. The routine checks for the PCI vendor and device identifiers matching a w83c553 compatible device. If the check is positive, the <code>drv_bind()</code> routine binds the driver to the device node, attaching a <code>PROP_DRIVER</code> property to the device node. The property value specifies the w83c553 driver name. The parent bus driver uses such a property to determine the name of the driver servicing the device. Via the <code>PROP_DRIVER</code> property, the driver gives its name to the parent bus driver, asking it to invoke the <code>drv_int()</code> routine on that device. The <code>drv_bind()</code> routine does nothing if the <code>PROP_DRIVER</code> property is already present in the device node. In other words, the <code>drv_bind()</code> routine will not override existing driver-to-device binding.</p>

The driver provides the `drv_unload()` entry and supports driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time. The driver supports all bus events specified by the pci bus driver interface. As a consequence, the driver may be used with a hot-pluggable pci bus (for example, CompactPCI).

The Table below summarizes the characteristics of the w83c553 isa bus driver.

driver name:	"sun:pci-w83c553-(bus,isa)"
hardware:	Winbond W83C553F PCI/ISA bridge
exported interface:	"bus,isa" (BUS_CLASS,ISA_CLASS)
exported interface version:	0 (ISA_VERSION_INITIAL)
imported parent interface:	"pci" (PCI_CLASS)
minimal parent interface version:	0 (PCI_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	supported

The Table below lists device node properties used by the w83c553 driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"vend-id"	PCI_PROP_VEND_ID	PciPropVendId	m	
"dev-id"	PCI_PROP_DEV_ID	PciPropDevId	m	
"bus-num"	PCI_PROP_BUS_NUM	PciPropBusNum	m	
"dev-num"	PCI_PROP_DEV_NUM	PciPropDevNum	m	
"func-num"	PCI_PROP_FUNC_NUM	PciPropFuncNum	m	
"intr"	PCI_PROP_INTR	PciPropIntr	m	
"intr-conf"	W83C553_PROP_INTR_CONF	W83c553PropIntrConf	o	see text below

Note that the driver does not use any `PCI_PROP_IO_REGS` properties because its I/O range cannot be relocated. The bridge assumes zero based PCI/ISA I/O addresses, and uses dynamic resource allocation when available.

The `PCI_PROP_VEND_ID` property specifies the winbond PCI vendor identifier (must be 0x10ad).

The `PCI_PROP_DEV_ID` property specifies the winbond W83C553F PCI device identifier (must be 0x0565).

The `PCI_PROP_BUS_NUM`, `PCI_PROP_DEV_NUM`, and `PCI_PROP_FUNC_NUM` properties specify the configuration space address of the device. That is, the pci bus, device and function numbers to which the device is connected.

The `PCI_PROP_INTR` property specifies the w83c553 PCI interrupt used. Note that this interrupt is used for all 16 isa interrupts.

The `W83C553_PROP_INTR_CONF` property specifies the routing and priority configuration for w83c553 interrupt requests. This property defines the following fields:

- Routing of the PCI interrupts (A,B,C,D) to ISA interrupt requests. This field should be set using the `W83C553_PCI_INTR_ROUTING()` macro. By default, pci interrupts are not routed to any ISA interrupt requests (IRQs). It is assumed that they are handled directly by a PCI interrupt controller.
- Routing of the primary and secondary IDE interrupts to ISA IRQs. This field should be set using the `W83C553_IDE_INTR_ROUTING()` macro. By default, the IDE primary interrupt is routed to ISA IRQ #14 and the IDE secondary interrupt is routed to ISA IRQ #15.
- Relative priority of each ISA interrupt request. This field is an array [0..15] which specifies the priority of each ISA IRQ ranging from the lowest priority 0 to the highest priority 15. An in-service interrupt cannot be interrupted by another interrupt having a lower priority value. By default, all interrupt requests have the same priority. Thus, any in-service interrupt may be interrupted by the others

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`pci(9DDI)`, `bus(9DDI)`, `isa(9DDI)`

NAME	z85x30 – generic z85x30 hardware related constants			
SYNOPSIS	<div>drv/src/uart/z8530/z8530.h - z85x30 hardware related constants</div> <div>drv/src/uart/z8530/z8530.c - driver code</div> <div>drv/src/uart/z8530/z8530PROP.h - driver specific properties</div>			
FEATURES	DRV			
DESCRIPTION	<p>The z85x30 duart driver implements the uart device driver interface. The driver works on duart device which is software compatible with the z85x30 chip.</p> <p>The driver uses the common bus driver interface provided by a parent bus driver. Thus, the driver may be applied to any bus providing such an interface.</p> <p>The z85x30 driver does not provide the <code>drv_probe()</code> routine. In other words, the z85x30 driver does not enumerate the bus, nor does it detect a z85x30 device or create an associated device node. When the z85x30 driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.</p> <p>The driver does not provide the <code>drv_bind()</code> routine. In other words, the driver does not support dynamic driver-to-device binding. When the z85x30 driver is used, the driver should be explicitly bound to the device using the <code>PROP_DRIVER</code> property. It is assumed that the device-to-driver binding is done either by a device node creator or by a separate binder driver. Such a driver-to-device binder could be developed for this particular bus architecture.</p> <p>The driver provides the <code>drv_unload()</code> entry and supports driver component unloading. This allows the driver component to be unloaded if it is no longer being used and if it has been dynamically loaded at run time.</p> <p>Because the z85x30 controller provides two serial interfaces (channel A and B), the z85x30 device node creator should specify channels supported by the driver instance. The supported channels are specified as child device nodes (with respect to the parent z85x30 node) with predefined names. The node names are shown in the Table below:</p> <div>Table 1</div> <table><tr><td>child node name</td><td>alias</td><td>description</td></tr></table>	child node name	alias	description
child node name	alias	description		

"a"	Z8530_CHAN_A_NAME	z85x30 channel A supported
"b"	Z8530_CHAN_B_NAME	z85x30 channel B supported

The driver instance will support a channel if, and only if, a corresponding channel (child) node is attached to the z85x30 device node.

The driver implements the `drv_unload()` entry which allows the unloading of the driver component when it is dynamically loaded at run time.

The driver supports all bus events specified by the common bus driver interface. As a consequence, the driver may be used with a hot-pluggable bus (for example, PCMCIA).

The Table below summarizes characteristics of the z85x30 duart driver.

driver name:	"sun:bus-z85x30—uart"
hardware:	z85x30 compatible DUART chip
exported interface:	"uart" (UART_CLASS)
exported interface version:	0 (UART_VERSION_INITIAL)
imported parent interface:	"bus" (BUS_CLASS)
minimal parent interface version:	0 (BUS_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	supported
system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	supported

The table below lists device node properties used by the z85x30 driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the column "default value" shows a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"io-regs"	BUS_PROP_IO_REGS	<bus class specific>	m	
"intr"	BUS_PROP_INTR	<bus class specific>	m	
"clock-freq"	PROP_CLOCK_FREQ	PropClockFreq	o	10000000

The BUS_PROP_IO_REGS property specifies the z85x30 I/O registers range. The property value is bus class specific.

The BUS_PROP_INTR property specifies the z85x30 interrupt source. The property value is bus class specific.

The PROP_CLOCK_FREQ property specifies the input clock frequency in Hz. By default, the input clock frequency is 10 MHz. Note that, programming the time constant for a given baud rate, the driver assumes the 16x clock factor. The formula relating the baud rate to the time constant is shown below.

$$\langle \text{time-constant} \rangle = ((\langle \text{clock-freq} \rangle / 16) + (\text{baud-rate})) / (\langle \text{baud-rate} \rangle * 2)$$

RESTRICTIONS

Only a subset of modem signals is supported by the z85x30 controller. The current version of the z85x30 driver supports only modem signals provided by the z85x30 chip:

- RTS (request to send)
- CTS (clear to send)
- CD (carrier detect)

SEE ALSO

bus(9DDI), uart(9DDI)

Index

A

amd29xxx — amd29xxx compatible flash driver 11

B

benchns16550 — ns16x50 device driver 13

C

cheerio—Sun cheerio 10/100Mbps Ethernet device driver 15

D

dec2115x — dec21150/21152/21153/21154 pci-to-pci bridge family, pci bus drivers 18

dec21x4x — dec21040/21041/21143 Ethernet device driver 24

E

ebus — SUN ebus pci/isa bridge driver 29

el3—3Com etherlinkIII Ethernet device driver 32

epic100 — epic100 pci Ethernet device driver 36

F

fccEther — quicc fcc controller Ethernet device driver 40

I

i8254 — Intel i8254 timer device driver 44

intro — device drivers 47

isabios — Intel i386AT generic isa bus driver 48

isapci — Intel i386AT generic PCI/ISA bridge, ISA bus driver 50

M

m48txx — SGS m48txx real time Clock device driver 52

mc146818 — Motorola mc146818 real time clock device driver 54

N

ne2000 — ne2000 Ethernet device driver 56

ns16550 — generic ns16x50 compatible uart device driver 59

P

pcibios — Intel i386AT generic pci bridge, pci bus driver 62

pciconf—pci configuration space parser driver 64

pcienum—pci enumerator driver 67

Q

quicc8260 — quicc bus driver 69

quicc8xx — quicc bus driver 74

R

raven — Motorola pci host bridge, pci bus driver 79
ric — SUN reset, interrupt and clock controller 83

S

sabre — SUN PCI host bridge driver 85
sccEther — SCC ethernet device driver 88
sccUart — quicc scc controller uart device driver 92
simba — SUN ADVANCED PCI BRIDGE DRIVER 95
smc1660 — smc1660 isa Ethernet device driver 98
smcuart — smc uart device driver 101

T

tbDec — PowerPC timebase and decrementer timer device driver 104

U

universe — TUNDRA Universe PCI to VME bus bridge driver 106
univRemCom — TUNDRA Universe vme bus remote communication driver 111

V

vt82c586 — VIA Technologies pci-to-isa bridge, isa bus driver 114

W

w83c553 — winbond pci/isa bridge, isa bus driver 119

Z

z85x30 — generic z85x30 hardware related constants 122