OpenBoot™ 3.*x* Supplement for PCI

Sun Microsystems Computer Company
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No: 805-1627-10
Revision A, August 1997

Sun microsystems

THE NETWORK IS THE COMPUTER™

Contents

Tables

Preface

The *OpenBoot 3.x Supplement for PCI* is intended to provide OpenBoot PROM developers with information to aid them in development for PCI.

How This Book Is Organized

This manual is divided into the following chapters:

Chapter 1, "PCI FCode Information," has basic information for developers writing FCode for use with PCI.

Chapter 2, "FCode Design Considerations," is material for developers to take note of when designing PCI FCode.

Chapter 3, FCode Debugging," describes how to debug FCode code that is used with PCI.

Chapter 4, Troubleshooting," contains information intended to help you when troubleshooting problems when FCode is used with PCI.

Related Documents

The following documents contain topics that relate to the information in this document.

Application	Title	Part Number
Boot firmware standard	*IEEE 1275 Boot Firmware Standard* (EEE 800-678-4333)	n/a
Writing PCI FCode	*IEEE 1275/PCI Bindings*. Open Firmware (1275) Homepage: `http://playground.sun.com/1275/` IEEE 1275/PCI bindings: `http://playground.sun.com/1275/bindings/pci/`	n/a
FCode descriptions, testing	*Writing FCode 3.x Programs* —call SunExpress 800-873-7869 PostScript available online from SunSoft on the 2.5.1 Device Developer Kit CD Full paper doc set.	SDDK-251-CDB DDDK-251-CDB 802-5895-10
PCI specification	*PCI Local Bus Specification, Rev. 2.1, PCI-to-PCI Bridge Architecture Specification Rev. 1.0* (PCI Special Interest Group, 800-433-5177 or 503-797-4207)	n/a
OpenBoot command reference	*OpenBoot 3.x Command Reference* *OpenBoot 2.x Command Reference* *(Sun Express, 800-873-7869)*	802-3242-10 802-3241-10

Ordering Sun Documents

SunDocs^{SM} is a distribution program for Sun Microsystems technical documentation. Easy, convenient ordering and quick delivery is available from SunExpress. You can find a full listing of available documentation on the World Wide Web: `http://www.sun.com/sunexpress/`

Sun Welcomes Your Comments

Please use the *Reader Comment Card* that accompanies this document. We are interested in improving our documentation and welcome your comments and suggestions.

If a card is not available, you can email or fax your comments to us. Please include the part number of your document in the subject line of your email or fax message.

- Email: `smcc-docs@sun.com`
- Fax: SMCC Document Feedback
  1-415-786-6443

PCI FCode Information 1

This chapter contains basic information for developers writing FCode for use with PCI.

PCI FCode PROM Header Format

The PCI FCode PROM header format is described in Table 1-1.

Table 1-1 PCI FCode PROM Header Format

Header	**Format**
PCI expansion PROM header	28 bytes
PCI data structure	24 bytes
FCode	(8 Byte FCode header + FCode code bytes)

If you are `dload`ing or `boot`ing your FCode image on a Solaris 2.5.1 or 2.6 system, you must replace the `a.out` header by an ELF header.

The `fakeboot` utility can add an ELF header based on parameters that you pass to `fakeboot`. The `addhdr` utility can also add either an `a.out` or ELF header, in addition to adding a PCI header and PCI data structure.

The PCI Expansion PROM Header Format

The PCI expansion PROM header format (28 bytes) is as follows:

Table 1-2 PCI Expansion PROM Header Format

Byte Offset	Value	Description
00	55(h)	PROM signature byte one.
01	aa(h)	PROM signature byte two.
02-03	34 00 (h)	SPARC reserved value
04-17	00 00	Reserved for processor architecture-unique data.
18-19	1c 00	Pointer to PCI data structure (assuming PCI data structure follows immediately after PCI expansion PROM Header).
1A-1B	00 00	Pad bytes.

PCI Expansion PROM Data Structure Format

The PCI expansion PROM data structure (24 bytes) is described in Table 1-2.

Table 1-3 PCI Expansion PROM Data Structure

Byte offset	Description /(Hex. value)
00-03	signature: P C I R (50 43 49 52)
04-05	vendor id
06-07	device id
08-09	Pointer to Vital Product Data
0A-0B	PCI data structure length (18 00)
0C	PCI data structure revision.
0D	Programming interface code
0E	Subclass code
0F	Class code
10-11	Image length in 512 bytes
12-13	Revision level of code/data

Table 1-3 PCI Expansion PROM Data Structure *(Continued)*

Byte offset	Description /(Hex. value)
14	Code type (01)
15	Indicator byte. For last image (80)
16-17	Reserved (00 00)

Table 1-3 shows a dump of initial bytes in a PCI FCode PROM with an `a.out` header in the first 32 bytes.

Table 1-4 PCI FCode PROM Dump

Hex Addr	Hex Value
00000	01 03 01 07 00 00 46 98 00 00 00 00 00 00 00 00
00010	00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00
00020	55 aa 34 00 00 00 00 00 00 00 00 00 00 00 00 00
00030	00 00 00 00 00 00 00 00 1c 00 00 00 50 43 49 52
00040	8e 10 01 10 00 c0 18 00 00 00 00 02 7e 00 00 01
00050	01 80 00 00 fd 03 18 6e 00 00 46 64 xx xx xx xx

For the PROM above, the vendor id is 0x108e, the device id is 0x1001, the pointer to Vital Product Data is 0xc000, class code is 0x02,subclass code is 0, programming interface code is 0, image length (in 512 bytes) is 0x7e, FCode length is 0x4664 bytes, xx..... are FCode data.

Format of Physical Address in "`reg`" Property

- For PCI, the "`reg`" property value has 5 32-bit numbers:
- - `phys.hi`
- , `phys.mid`,
- `phys.lo`,
- `size.hi`
- , `size.lo`.
- The `size.hi` and `size.lo` are values for a register size the address and type of which are defined by `phys.hi`, `phys.mid`, and `phys.low`.

The format of the physical address in the “`reg`” property is as follows:

phys.hi cell:	*npt000ss bbbbbbbb dddddfff rrrrrrrr*
phys.mid cell:	*hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh*
phys.low cell:	*LLLLLLLL LLLLLLLL LLLLLLLL LLLLLLLL*

where

- *n* is 0 if the address is relocatable; 1 otherwise.
- *p* is 1 if the addressable region is prefetchable; 0 otherwise.
- *t* is 1 if the address is aliased (for non-relocatable I/O), below 1MByte (for memory), or below 64KBytes (for relocatable I/O).
- *ss*=00 ==> configuration space.
- *ss*=01 ==> I/O space.
- *ss*=10 ==> 32 bit memory space.
- *ss*=11 ==> 64 bit memory space.
- *bbbbbbbb* is an 8-bit bus number (assigned by the CPU PROM at probe time).
- *ddddd* is a 5-bit device number.
- *fff* is a 3-bit function number.
- *rrrrrrrr* is an 8-bit register number.
- *hh..hh* is a 32-bit unsigned number, most significant bits.
- *LL..LL* is a 32-bit unsigned number, least significant bits.

PCI Device Configuration Register Access

To find the address to use for configuration register access on your PCI device, look in the format for the physical address of the “`reg`” property. You can use the `phys.hi` cell of the first entry in the "`reg`" property, as the base address for the configuration space.

The first entry in the "`reg`" property must be the configuration space entry (*bbbb.bbbb.dddd.dfff.0000.0000* binary). Use that (or any other method), to obtain the values of *bbbb.bbbb, ddddd* and *fff* for your device. Then use:

```
ok "<parent-pci-bus-node>" select-dev
ok <bbbb.bbbb.dddd.dfff>XX config-l@
```

(`XX` is the offset for that register configuration.) For example, if the bus number is 1000.0001 (0x81), the device number is 0.0000, and the function number is 001 (0x01), then use

```
ok " /pci@1f,2000" select-dev
ok 81.0100 config-l@ (to read device id and vendor id)
ok 81.0104 config-w@ ( to read command register)
ok 81.0130 config-l@ ( to read the expansion PROM base address register)
```

Boot Software Roles

There are three kinds of software that come into play during a boot:
- Operating system kernel,
- FCode operating system kernel driver.

This section describes the normal Solaris boot scenario, including their functions and the order in which they begin.

At power-on, the CPU PROM begins execution. It probes all on-board devices and plug-in cards to interpret the FCodes on all FCode PROMs. In the FCode probing process, some FCode PROMs execute commands to reset the device. But, generally, FCode PROMs generate device properties for respective devices.

Then, the CPU PROM boots over the specified boot device (using its FCode boot driver), loads `bootblk` (or `inetboot` for network booting), and passes control to the `bootblk` code. The `bootblk` code loads the kernel and modules, and passes control to the kernel. The kernel at some point starts to use the OS-device driver.

The order is:
- CPU-PROM
- FCode-PROM
- `bootblk`
- Operating system kernel
- Operating System driver

CPU PROM-Generated Properties

This section discusses the properties created by the motherboard CPU PROM from the information given in the configuration space registers of the PCI device.

The CPU PROM normally generates the following properties in a PCI device node:

- `vendor-id`
- `device-id`
- `revision-id`
- `class-code`

and `devsel-speed` from information in configuration space registers. The `"interrupts"` property is present if the Interrupt Pin register is non-zero. The following properties will be present only if the corresponding capability is available from the device or the corresponding value was non-zero as indicated in the configuration space registers:

- `66mhz-capable`
- `udf-supported`
- `cache-line-size`
- `fast-back-to-back`
- `subsystem-id`
- `subsystem-vendor-id`

`min-grant` and `max-latency` properties are created unless the header type is 01. The CPU PROM also creates the `"assigned-addresses"` property, with entries for each address base register for which an address was assigned.

FCode Design Considerations 2

This chapter contains information to consider when designing FCode code for PCI.

Adding a PCI Header to a PROM

To add a PCI header to your PROM, look at the source code of `fakeboot.c` from DDK 2.5.1, and enhance it to add a PCI header for your PROM. Another approach is to use `addhdr`, available with the Solaris 2.6 DDK.

Accessing a PCI Device's Configuration Space Registers

It isn't necessary to do anything extra to access your device's configuration space registers. They are always accessible.

Base Address Register Setting

The base address registers used in configuration space are set by the CPU PROM.

The CPU PROM (not the PCI card's FCode PROM) allocates the base address for memory andor I/O space on your PCI device and for the FCode PROM.

System Cache Line Size

You must write the system's cash line size into the cash line size configuration space register of your PCI. To do this, look in the cache-line-size register of the configuration space; it refers to the cache line size supported by the PCI device.

Sun Ultra™ 1 UPA/PCI-Related Nodes

The PCI-related nodes on the Sun Ultra™ 1 UPA/PCI system are `/pci@1f,4000` and `/pci@1f,2000`. `pcia` and `pcib` are required for the NVRAM variables `pcia-probe-list` and `pcib-probe-list` and are determined as follows:

Each PCI bus has a property named "`slot-name`" which gives information about slots on that PCI bus and sometimes indicate which NVRAM variable corresponds to it.

To get a value for that property, type the following:

```
ok " </pci-bus-node>" select-dev
ok " slot-name" get-my-property drop decode-int .h cr  type
```

To get a PCI bus at `/pci@1f,2000`, type `the following`:

```
ok " /pci@1f,2000" select-dev
ok " slot-name" get-my-property drop decode-int .h cr  type
```

The display will be something like the following to indicate that the devices

```
6
pcia slot 1pcia slot 2
ok “ /pci@1f.4000” select-dev
ok “ slot-name” get-my-property drop decode-int .h cr  type
4
32-bit slot 2
```

under `/pci@1f,2000` relate to `pcia`.

In a Sun Ultra 1 UPA/PCI with four plug-in PCI slots, only slot 1 is physically present for `pci@1f,2000`. It can also support 66 Mhz., 64 bit PCI devices as shown in the following example.

```
ok " /pci@1f,4000" select-dev
ok " slot-name" get-my-property drop decode-int .h cr  type
34
pcib slot 2pcib slot 4pcib slot 5
```

This example indicates that devices under `/pci@1f,4000` relate to pcib.

The PCI slot in a Sun Ultra 1 UPA/PCI with a 4 plug-in is used as follows :
- Slots 2, 4, and 5 under `/pci@1f,4000` support 33 Mhz., 32 bit PCI devices.
- Slot 3 under `/pci@1f,4000` is used for an on-board SCSI device.

Note that the value of the “`slot-name`" property differs for different systems. Some systems may not differentiate the PCI bus by the value of the "`slot-name`" property.

In different releases of the PROM for the same system, the value of the "`slot-name`" property may also change. You may need to refer to your system documentation for details about using PCI buses on the system.

Alternatively, you can find which NVRAM variable refers to which PCI bus by setting the NVRAM variables to different values or by plugging PCI cards in different slots.

Using Physical Addresses

When you need to find and use physical addresses to access, for instance, configuration space registers on a Sun Ultra 1 UPA/PCI system, use MMU bypassing by selecting with the correct ASI space. The arguments for the space `{c,d,w,l,x}` command includes an address and ASI code (and data for a write operation.)

On Sun Ultra 1 UPA/PCI systems, the PCI device configuration registers are viewed using the following address:

```
1fe.0100.0000 + x
```

where the 32-bit value of `X` is represented in bit format as:

```
bbbb.bbbb.dddd.dfff.rrrr.rrrr
```

where

bbbb.bbbb is an 8-bit bus number
dddd.d is a five bit device number
fff is a three-bit function number
rrrr.rrrr is an eight-bit register number

For example, if the bus number is 81, device number is 0 and function number is 1, then x will be 81.0100, giving you a configuration register base.

This will give you access to the 0th configuration register at 1fe.0181.0100 (physical address). On Sun Ultra 1 UPA/PCI systems, you can use ASI 0x15 for a non-cacheable address being accessed by MMU bypass. If you are accessing a little-endian device, use `ASI 0x1d`.

You can get bus number, device number, and function number from `my-space` after selecting that device or from the "`reg`" property value for that device. Look in IEEE 1275/PCI binding for the "`reg`" property format.

In general, to get physical addresses for registers in any space, (configuration space, 32- bit memory space, and others) use the `map-in` command. `map-in` requires `phys.lo`, `phys.mid`, `phys.hi`, and length arguments. `You can take the phys.lo`, `phys.mid`, and `phys.hi` numbers from the corresponding "`reg`" property.

In the case of configuration space, getting the physical address is easy since `phys.lo` and `phys.mid` are always zero. `phys.hi` is just the configuration space address.

Table 2-1 is an example of getting the physical address of the configuration space registers, using onboard Ethernet on a Sun Ultra 1 UPA/PCI system:

Code Example 2-1 Configuration Space Registers Physical Address

```
ok " /pci@1f,4000/network@1,1" begin-select-dev
ok pwd

/pci@1f,4000/network@1,1
ok .properties
.
.
reg  (Config Space ---->)      00000900 00000000 00000000 00000000 00000000
(32bit memory space ---->) 02000910 00000000 00000000 00000000 00007020
.
.
.
ok 0 0 900 100 " map-in" $call-parent constant my-cfg-vaddr
ok my-cfg-vaddr .
fff80900
ok my-cfg-vaddr map?
VA:fff80900
G:0 W:1 P:1 E:1 CV:0 CP:0 L:0 Soft1:1 PA[40:13]:ff00800 PA:1fe01000000
Diag:0 Soft2:0 IE:0 NFO:0 Size:0 V:1
PA:1fe01000900
```

This results in the physical address for the base of configuration registers as `1fe.0100.0900` for this device. For plug-in PCI devices, the registers' physical address may vary if the device is plugged into a different slot, or if other devices are present. Similarly, use the "`reg`" entry for memory or I/O space, to find a physical address for those spaces.

Controlling PCI Slot Probing on an Ultra 1 UPA/PCI System

You can control probing of PCI slots on your Sun Ultra 1 UPA/PCI system as follows: during normal system initialization on the Sun Ultra 1 UPA/PCI system, there are NVRAM variables that indicate to the CPU PROM which slots to probe and in what order.

On the Sun Ultra 1 UPA/PCI system the slots are: `pcia-probe-list` and `pcib-probe-list`. The default value for `pcia-probe-list` is 1,2; for `pcib-probe-list`, it is 5,4,3,2. To disable slot 4 probing on `pcib` during normal initialization after a reset, change `pcib-probe-list` to:

```
ok setenv pcib-probe-list 5,3,2
```

To probe slot 4 on `pcib` manually after a reset, type:

```
ok 4 probe-pci-slot /pci@1f,4000
```

Note that not all CPU PROMs have the `probe-pci-slot` command. In future PROMs, this command may not work in the same way or may be eliminated.

Using 3.x Tokenizer and 3.x CPU PROMs

When using the 3.x Tokenizer while testing FCode under CPU OpenBoot PROM 3.x versions, make sure that you use OpenBoot PROMs version 3.1 or later. You will need the following NVRAM patch for PROMs prior to 3.1

Code Example 2-2 NVRAM Patch for PROMS Prior to Version 3.1

```
ok nvedit
0:: nl-move( src dst len -- ) rot n->l rot n->l rot n->l
(move);
1: ['] nl-move is move
2: ['] l>>a 2 la+ dup l@ h# 1000 invert and swap l!
3: ['] lrshift 2 la+ dup l@ h# 1000 invert and swap l!
4: ^C
ok nvstore
ok setenv use-nvramrc? true
ok reset-all
```

Note that while using the 2.x or 3.x tokenizer, literals or numbers that have bit 31 set to 1 will extend this bit (1) to bit 63 on 3.x CPU PROMs. For example, the following code will give a value of `ffff.ffff.8000.000`

```
8000.0000 constant xxx
```

When such words or constants are used in address manipulation or otherwise, your code should clip them to a 32 bit value:

Get a real 8000.0000 by:

```
ff ff ff ff bljoin constant x-num
: clip-num ( n -- l )  x-num and ;
8000.0000 clip-num constant  xxx
```

or

use "`xxx clip-num`" wherever "`xxx`" is being used.

FCode Debugging 3

This chapter contains examples for creating a PCI FCode package and using debugging flags in debugging FCode.

Packaging PCI FCode

The following is an example of testing a new version of an FCode program when the developer creates a new package:

```
ok 4000 dload /stand/mydev.fcode
ok 0 0 " 4,0”  " /pci@1f,2000" begin-package
ok 4020 1 byte-load
ok end-package
```

However, when performing an `ls`, it is obvious that there are now *two* packages corresponding to the card:

```
ok l
ffd70c00 pci108e,1001@4,0
ffd6e860 pci108e,1001@4,0
ok
```

To override the original package so that the downloaded code is executed, remove the PCI card PROM. The CPU PROM will create a device node for the card, but the `name` property will have a value of `pci<DDDD>,<VVVV>`.

Create a name property for your device in your downloaded code with a different value than the one created by the CPU PROM. Then refer to your device by its full device path.

System Flags and FCode Debugging

The following is an example of how to use Sun systems debugging flags to aid in debugging FCode.

Set the NVRAM variable `fcode-debug?` to true to keep the headers for words that are preceded with headers in your code.

Some CPU PROMS have a `fcode-verbose` variable to display each FCode as it is being read at probe time by the CPU PROM token interpreter.

To turn it on, before you probe your FCode, type:

```
ok true to fcode-verbose?
<probe-your-card>
```

To set it from NVRAMRC, type:

```
ok nvedit
0: true to fcode-verbose?
^C
ok nvstore
ok setenv use-nvramrc? tru
ok reset-all
```

Some CPU PROMs have `pcimsg?` and `probemsg?` variables to give additional PCI-related information. You can turn them on in the way as described in what to do before you probe your FCode. `pcimsg?` controls the display of all accesses to PCI configuration space. `probemsg?` controls the display of probing status information, including physical allocation.

Note that not all CPU PROMs have `pcimsg?` and `probemsg?`. In future PROMs, this command may not work in the same way or may be eliminated.

Troubleshooting 4

This chapter contains examples for troubleshooting your PCI FCode.

Enabling Access to a PCI Device's Memory Space Locations

Problem:

When loading FCode, memory space locations can't be accessed.

Solution:

Look in the format for the physical address of the `reg` property. Then locate the values of `bbbb.bbbb,ddddd` and `fff` for your device by using:

```
ok " <parent-pci-bus-node>" select-dev
ok 3 <bbbb.bbbb.dddd.dfff>04 config-w!
```

This will write to the configuration space command register and thus enable access to memory and I/O space. This sets bit[0] and bit[1] of the command register. In the same way, you may set other bits in the command register, if required. If the Sbus number is 1000.0001 (0x81), the device number is 0.0000, and the function number is 001 (0x01), you will then use:

```
ok " /pci@1f,2000" select-dev
ok 81.0104 config-w@ 3 or 81.0104 config-w!
```

Normally, your FCode driver's `open` routine should enable such access. FCode can use the value returned by `my-space` and add an offset of 4 to get the address of the command register. It can then set various bits in the command register to enable the desired access. Use the `close` routine to disable that access.

Expansion FCode PROM

Problem:

A developer is unable to access his expansion FCode PROM. How can access to it be enabled?

Solution:

To enable access, look in the format for the physical address of the `reg` property. Then obtain the values of `bbbb.bbbb`, `ddddd` and `fff` for your device by using:

```
ok "<parent-pci-bus-node>" select-dev
ok <bbbb.bbbb.dddd.dfff>04 config-w@ 3 or
<bbbb.bbbb.dddd.dfff>04
config-w!
ok <bbbb.bbbb.dddd.dfff>30 config-l@ 1 or
<bbbb.bbbb.dddd.dfff>30 config-l!
```

This will first enable memory and I/O space access. Then, it will read the value from the expansion PROM configuration space base address register (at offset 0x30) *or* 1 to it, and write the value in the expansion PROM base address register to enable access to your FCode PROM. This sets bit[0] of the expansion PROM base address register. If the bus number is 1000.0001 (0x81), the device number is 00000, and the function number is 001 (0x01), use example:

```
ok " /pci@1f,2000" select-dev
k 81.0104 config-w@ 3 or 81.0104 config-w!
ok 81.0130 config-l@ 1 or 81.0130 config-l!
```

If the FCode needs to access PROM data (for example, to access Vital Product Data stored in the PROM) then the FCode should enable PROM access by using the value returned by `my-space` and adding an offset of 0x30 as the register address. The FCode should read the value from the address, *or* 1, and write it to that address.

Also, since the FCode was copied to memory, then the devices memory and I/O spaces may not be enabled. The FCode must then enable them, using the following FCode:

```
ok my-space h# 30 + dup config-l@ 1 or swap config-l! (enable PROM
access)
ok my-space h# 4 + dup config-w@ 3 or swap config-w!  (enable
I/O,memory access)
```

Using the previous example, you can disable expansion PROM access as:

```
ok my-space h# 30 + dup config-l@ 1 invert and swap
config-l!(disable PROM access)
```

Packaging Error With Ethernet FCode

Problem:

When you try to load the FCode from Ethernet, the code seems to load without errors. However, when you build the package, the following error is displayed:

```
ok 4000 dload /stand/cheerio.o
Boot device: /pci@1f,4000/network@1,1:,|stand|cheerio.o  File
and args:
ok 0 0 " 0,1" " /pci@1f,2000" begin-package
ok 4000 1 byte-load
Unimplemented FCode token before address 4004
Warning: FCode sequence resulted in a net stack depth change of 1
ok
```

Solution:

One error may be due to the PCI header attached to the PROM image.

Dump the download image beginning at 4000, for example, `4000 60 dump, and` see where `fd` starts. It is the beginning of the FCode data for the byte-load. For instance, if the FCode data starts at `x`, use the address `x` in

```
ok X 1 byte-load
```

`fd` is the beginning of the FCode header and is 8 bytes long:

```
fd, <tokenizer-version>,<2 reserved bytes/checksum>,<4byte of
FCode length>
```

Note – If you begin your FCode source with `fcode-version1`, the first FCode data is `fd`, but if you use `fcode-version2`, the first FCode data is `f1`.

`Another error may be due to:`

`my-address` is two 32-bit numbers for PCI and only one 32-bit number for SBus.

Change your FCode to handle two numbers returned from `my address`.

To do this, use the following code:

```
my-address     constant my-bus-addr-mid  constant my-bus-addr-low

: my-bus-addr (-- paddr.low paddr.mid )
my-bus-addr-low my-bus-addr-mid
;
```

Then use `my-bus-addr` to create the `reg` property.

select-dev *Errors*

To debug your FCode/device for errors while using select-dev on the device, do the following:

Add a dummy open method to your device node's FCode if you want to select the device to map in the device, look at the ok prompt, look at the device registers, and so on:

```
ok dev /pci..../<device-node>
ok : open true ;
```

This may generate the following message about open not being unique:

```
ok device-end
```

Now you can use select-dev to select your device. Then use "map-in" $call-parent to map in the device registers, and examine them. (The -endianness may differ from what you think. Verify the way that the device is mapped with map? Also, verify that rl@ and other register access words return the data in the way you expect.

Index