

Writing FCode 3.x Programs

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

[Part No: 802-6287-10](#)
[Revision A, November 1997](#)



THE NETWORK IS THE COMPUTER™

Copyright 1997 Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, California 94043-1100 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.

Portions of this product may be derived from the UNIX[®] system, licensed from Novell, Inc., and from the Berkeley BSD system, licensed from the University of California. UNIX is a registered trademark in the United States and other countries and is exclusively licensed by X/Open Company Ltd. Third-party software, including font technology in this product, is protected by copyright and licensed from Sun's suppliers.

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).

Sun, Sun Microsystems, the Sun logo, SunSoft, SunDocs, SunExpress, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK[®] and Sun[™] Graphical User Interfaces were 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.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.



Contents

1. SBus Cards and FCode	1
FCode PROM Format	2
Interpreting FCode.....	2
Device Identification	2
Creating and Executing FCode Definitions	3
2. PCI FCode Information	5
PCI FCode PROM Header Format.....	5
The PCI Expansion PROM Header Format	6
PCI Expansion PROM Data Structure Format	6
Format of Physical Address in "reg" Property	7
CPU PROM-Generated Properties.....	8
Adding a PCI Header to a PROM	10
3. Elements of FCode Programming	11
Colon Definitions.....	12
Stack Operations	13

Programming Style	14
Commenting Code.	14
Coding Style	15
Definition Length.	15
Stack Comments.	16
A Minimal FCode Program.	18
FCode Classes.	19
Primitive FCode Functions	20
System FCode Functions.	21
Interface FCode Functions	21
Local FCode Functions	22
4. Debugging and Testing	
FCode Programs	23
Packaging PCI FCode	23
System Flags and FCode Debugging.	24
FCode Source	25
Tokenizing FCode Source	25
FCode Binary Format	27
Testing FCode Programs on the Target Machine	27
Configuring the Target Machine	28
Setting Appropriate Configuration Parameters.	28
Modifying the Expansion Bus Probe Sequence	29
Getting to the Forth Monitor	30
Using the Command Line Editor of the Forth Monitor	30

Using the Forth Monitor to Test FCode Programs.....	30
Using <code>dload</code> to Load from Ethernet.....	31
Using <code>dlbin</code> to Load From Serial Port A.....	32
Using <code>boot</code> to Load From Hard Disk, Diskette, or Ethernet	32
Using <code>d1</code> to Load Forth Over Serial Port A	33
Using the Forth Monitor to Interpret an FCode Program.....	34
Using the Forth Monitor to Browse a Device Node.....	36
Using the Forth Monitor to Test a Device Driver.....	38
Device Node Methods.....	38
Using <code>select-dev</code>	38
Using <code>begin-select-dev</code>	40
Using <code>execute-device-method</code>	41
Using <code>apply</code>	42
Testing FCode Programs in Source Form	42
Producing an FCode PROM.....	42
Exercising an Installed FCode PROM.....	43
<code>select-dev</code> -Generated Errors	43
5. Helpful Testing and Debugging Hints	45
Accessing a PCI Device's Configuration Space Registers.....	45
Base Address Register Setting	45
System Cache Line Size.....	45
Sun Ultra-30 UPA/PCI-Related Nodes.....	46
Finding and Using Physical Addresses.....	47
Controlling PCI Slot Probing on an Ultra-30 UPA/PCI System	49

Using 3.x Tokenizer and 3.x CPU PROMs	50
PCI Device Configuration Register Access	51
Boot Software Roles	51
Enabling Access to a PCI Device's Memory Space Locations	52
Expansion FCode PROM	53
Packaging Error with Ethernet FCode	54
6. Packages	57
Package Instances	58
Package Data	61
Static and Instance-specific Methods	62
Execution Tokens	63
Intra-package Calling Methods	63
Accessing Other Packages	64
Inter-package Calling Methods	66
execute-device-method and apply	69
Plug-in Device Drivers	70
Common Package Methods	71
Basic Methods	71
open	71
close	71
Recommended Methods	72
reset	72
selftest	72
Package Data Definitions	73

Instance Arguments and Parameters	74
Package Addresses	76
Package Mappings	77
nvramrc	77
Modifying Package Properties	78
Standard Support Packages	78
Sun Disk-Label Support Package	79
TFTP Booting Support Package	80
Deblocker Support Package	81
7. Properties	83
Standard FCode Properties	85
Standard Property Names	86
Display Device Properties	86
Network Device Properties	87
Memory Device Properties	87
General Properties For Parent Nodes	87
Properties For PCI Parent Nodes	88
Properties for PCI Child Nodes	88
Detailed Descriptions of Standard Properties	89
#address-cells	89
#size-cells	89
address	90
address-bits	90
alternate-reg	91

assigned-addresses	91
available	92
big-endian-aperture.	92
bus-range	93
big-endian-aperture.	93
character-set	93
class-code	93
compatible	94
class-code	94
depth	94
device-id	95
device-id	95
device_type.	95
devsel-speed.	96
existing	96
fast-back-to-back	97
has-fcode.	97
fast-back-to-back	97
has-fcode.	97
height.	97
interrupts	98
linebytes	99
little-endian-aperture	100
local-mac-address	100

mac-address	100
max-frame-size	101
max-latency	101
min-grant	102
model	102
name	102
page-size	103
power-consumption	104
ranges	105
reg	110
revision-id	113
slot-names	113
status	114
translations	114
vendor-id	115
width	115
Manipulating Properties	115
Property Creation and Modification	115
Property Values	115
Property Encoding	116
Property Retrieval	116
Property Decoding	117
Property-Specific FCodes	118
8. Block and Byte Devices	121

Block Devices	121
Byte Devices	122
Required Methods	122
block-size (-- block-len)	122
load (addr -- size)	122
max-transfer (-- max-len)	123
read (addr len -- actual)	123
read-blocks (addr block# #blocks -- #read)	123
seek (poslow poshigh -- status) for block; (offset file# -- error?) for byte	123
write (addr len -- actual)	124
write-blocks (addr block# #blocks -- #written)	124
Required Properties	125
Device Driver Examples	125
Simple Block Device Driver	126
Extended Block Device Driver	126
Complete Block and Byte Device Driver	134
9. Display Devices.	145
Required Methods	145
Required Properties	146
Device Driver Examples	146
Simple Display Device Driver	146
Extended Display Device Driver	147
Complete Display Device Drive	151

10. Memory-Mapped Buses	157
Required Methods	157
decode-unit (addr len -- phys.lo ... phys.hi)	157
dma-alloc (size -- virt)	158
dma-free (virt size --)	158
dma-map-in (virt size cacheable? -- devaddr).....	159
dma-map-out (virt devaddr size --)	159
dma-sync (virt devaddr size --)	160
probe-self (arg-addr arg-len reg-addr reg-len fcode-addr fcode-len --)	160
map-in (phys.lo ... phys.hi size -- virt)	161
map-out (virt size --)	161
SBus Addressing	161
SBus Required Properties	162
Device Driver Examples	162
Basic Hierarchical Device Driver	163
Extended Hierarchical Device Driver	165
Complete Hierarchical Device Driver	173
11. Network Devices	191
Required Methods	192
load (addr -- len).....	192
read (addr len -- actual).....	193
write (addr len -- actual).....	193
Required Device Properties	193

Optional Device Properties.....	193
Device Driver Examples	194
Simple Network Device Example	194
Sample Driver With Test and Debugging Methods.....	196
Bootable Network Device Driver Example.....	209
12. Serial Devices	237
Required Methods	237
install-abort (--).....	237
read (addr len -- actual)	237
remove-abort (--)	238
write (addr len -- actual)	238
Required Properties.....	238
Device Driver Examples	238
Simple Serial FCode Program.....	239
Extended Serial FCode Program	239
Complete Serial FCode Program	241
13. PCI FCode Driver Example	247
14. FCode Dictionary	255
A. FCode Reference	399
FCode Primitives	399
FCodes by Function.....	399
FCodes by Byte Value	424
FCodes by Name	440
B. FCode Memory Allocation	457

C. Coding Style	459
Typographic Conventions.....	459
Use of Spaces	459
if...then...else	460
do...loop	461
begin...while...repeat	461
begin...until...again.....	462
Block Comments	462
Stack Comments.....	462
Return Stack Comments	463
Numbers	463
D. Differences Between FCode 2.x and 3.x	465

Figures

Figure 6-1	An Instance Chain for /iommu/sbus/SUNW,cgsix.....	59
Figure 6-2	An Instance Chain for /iommu/sbus/ledma/le with obp-tftp Support	61
Figure 8-1	Sample Device Tree.....	125
Figure 11-1	QED Device Tree	192

Tables

Table 2-1	PCI FCode PROM Header Format	5
Table 2-2	PCI Expansion PROM Header Format	6
Table 2-3	PCI Expansion PROM Data Structure	6
Table 2-4	Format of Physical Address in "reg" Property	8
Table 3-1	Stack Parameter Abbreviations	16
Table 3-2	FCode Source Word Classes	19
Table 4-1	FCode Binary Format	27
Table 4-2	FCode Header Format	27
Table 4-3	Common Package-related Commands	30
Table 4-4	Commands for Browsing the Device Tree	36
Table 6-1	Package Access FCodes	65
Table 6-2	Manipulating <code>phandles</code> and <code>ihandles</code>	66
Table 6-3	Functions Enabling Calling Other Packages' Methods	66
Table 6-4	Sun Disk Label Package Methods	80
Table 6-5	TFTP Package Methods	81
Table 6-6	Deblocker Package Methods	81

Table 7-1	Standard Device Types	96
Table 7-2	interrupts Property Value Encoding.	99
Table 7-3	Child-Parent Address Relationships	106
Table 7-4	Child-Parent Address Relationships for a PCI Node in a PPCRP Machine	108
Table 7-5	status Property values.	114
Table 7-6	Property-specific FCodes	118
Table 8-1	Required Properties of Block and Byte Devices	125
Table 9-1	Required Display Device Properties.	146
Table 10-1	SBus Required Properties.	162
Table 11-1	Required Network Device Properties.	193
Table 11-2	Optional Network Device Properties	193
Table 12-1	Serial Driver Required Properties.	238
Table 14-1	Escape Sequences in Text Strings.	256
Table A-1	Stack Manipulation	399
Table A-2	Arithmetic Operations	400
Table A-3	Memory Operations	402
Table A-4	Atomic Access	403
Table A-5	Data Exception Tests.	403
Table A-6	Comparison Operations	403
Table A-7	Text Input	404
Table A-8	ASCII Constants.	405
Table A-9	Numeric Input	405
Table A-10	Numeric Primitives.	405
Table A-11	Numeric Output	406
Table A-12	General-purpose Output	406

Table A-13	Formatted Output	407
Table A-14	<code>begin</code> Loops	407
Table A-15	Conditionals	407
Table A-16	Case Statements	408
Table A-17	<code>do</code> Loops	408
Table A-18	Control Words	408
Table A-19	Strings	409
Table A-20	Defining Words	409
Table A-21	Dictionary Compilation	410
Table A-22	Dictionary Search	410
Table A-23	Conversion Operators	410
Table A-24	64-bit Operations	411
Table A-25	Memory Buffers Allocation	412
Table A-26	Miscellaneous Operators	412
Table A-27	Internal Operators (invalid for program text)	413
Table A-28	Virtual Memory Allocation	415
Table A-29	Properties	415
Table A-30	Commonly-used Properties	416
Table A-31	System Version Information	417
Table A-32	Device Activation Vector Setup	417
Table A-33	Self-test Utility Routines	417
Table A-34	Time Utilities	417
Table A-35	Machine-specific Support	418
Table A-36	User-set Terminal Emulation Values	418
Table A-37	Terminal-set Terminal Emulation Values	418

Table A-38	Terminal Emulation Routines*	418
Table A-39	Frame Buffer Parameter Values*	419
Table A-40	Font Operators	420
Table A-41	One-bit Frame Buffer Utilities	420
Table A-42	eight-bit Frame Buffer Utilities	421
Table A-43	Package Support	421
Table A-44	Asynchronous Support	423
Table A-45	Miscellaneous Operations	423
Table A-46	Interpretation	423
Table A-47	Error Handling	423
Table A-48	FCode by Byte Value	424
Table A-49	Tokenizer Directives	436
Table A-50	FCode and Tokenizer Directives by Name	440
Table D-1	FCode Names Changed in Version 3.x	465
Table D-2	FCode 2.x Changed Names and Equivalent FCode 3.x Names	467
Table D-3	FCode 2.x Commands Deleted in FCode 3.x	467
Table D-4	New FCodes Added in 3.x	468
Table D-5	Differently Functioning 3.x FCodes With Changed Byte Values	468
Table D-6	3.x FCodes Related to 64-bit Operations	469
Table D-7	Device-related User Interface Commands Changed in 3.x	470

Preface

This manual, *Writing FCode 3.x Programs* (802-6287), describes how to write, debug, and test FCode programs for SPARC-based systems and PCI or SBus interface card devices. It replaces all previous manuals describing how to write FCode programs. *IEEE Standard 1275-1994* is based on the Sun™ OpenBoot™ 2.x implementation. OpenBoot 3.x from Sun is compliant with *IEEE Standard 1275*.

Throughout this book, the reference to FCode PROM might refer to any type of ROM device (ROM, PROM, FlashPROM, etc.).

Who Should Use This Book

This manual is written for designers of PCI and SBus interface cards and other devices that use the FCode programming language. It is written for those designers who have some familiarity with PCI or SBus card design requirements and Forth programming. The sample code in this book is provided as is without any warranty.

The material in this manual is for developers of FCode applications for PCI or SBus peripherals on OpenBoot 3.x. With proper programming precautions, these applications should run on OpenBoot 3.x and earlier versions. By following the IEEE 1275 standard, the same FCode application can also run on non-SPARC systems which are IEEE 1275-compliant. The FCode language is defined by *IEEE Standard 1275-1994 Standard for Boot Firmware*.

This manual is written for designers who have read and understood the corresponding SBus or PCI specifications and SBus or PCI binding to *IEEE Standard 1275-1994 1.2*.

How This Book Is Organized

- **Chapter 1, “SBus Cards and FCode”**, introduces the basic relationships between FCode device drivers and the hardware that they control.
- **Chapter 2, “PCI FCode Information”**, basic information for developers writing FCode for use with PCI.
- **Chapter 3, “Elements of FCode Programming”**, introduces the basic elements of FCode, stack notation, and programming style.
- **Chapter 4, “Debugging and Testing FCode Programs”**, describes the process of producing FCode programs, from source files to testing working programs.
- **Chapter 5, “Helpful Testing and Debugging Hints”**, is information to consider when you are designing FCode code for PCI.
- **Chapter 6, “Packages”**, describes the basic units of FCode program function.
- **Chapter 7, “Properties”**, describes properties, which define how an FCode device driver program recognizes the hardware that it controls.
- **Chapter 8, “Block and Byte Devices”**, describes the required methods for nonvolatile and sequential-access mass storage devices.
- **Chapter 9, “Display Devices”**, describes writing FCode programs for display devices.
- **Chapter 10, “Memory-Mapped Buses”**, describes addressing and required properties for memory-mapped buses.
- **Chapter 11, “Network Devices”**, describes how to implement network device drivers.
- **Chapter 12, “Serial Devices”**, describes programming requirements for serial devices, and gives examples of serial device drivers.
- **Chapter 13, “PCI FCode Driver Example”**, shows a PCI FCode driver which is an example of a way of dealing with PCI-related registers in FCode.
- **Chapter 14, “FCode Dictionary”**, describes currently-defined FCode words and their functions and use, with brief programming examples.

-
- **Appendix A, “FCode Reference”**, lists all currently-defined FCode words according to functional group, name, and byte value.
 - **Appendix B, “FCode Memory Allocation”**, describes guidelines for memory allocation and de-allocation in FCode.
 - **Appendix C, “Coding Style”**, contains an OpenBoot coding guideline.
 - **Appendix D, “Differences Between FCode 2.x and 3.x”**, discusses the FCodes and macros that have changed between FCode 2.x and FCode 3.x.

Related Books

This manual does not cover all you need to know in order to write FCode drivers. The following sources are also useful.

For further information:

- *IEEE Standard 1275-1994 Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices*
- *IEEE Standard 1275.1-1994 Standard for Boot (Initialization Configuration) Firmware: Instruction Set Architecture (ISA) Supplement for IEEE 1754*
- *IEEE Standard 1275.2-1994 Standard for Boot (Initialization Configuration) Firmware: Bus Supplement for IEEE 1496 (SBus)*
- *IEEE 1496-1993 Standard for Chip and Module Interconnect Bus: SBus 1-55937-353-9*
- *PCI Local Bus Specification, rev. 2.1* available from PCI Special Interest group: 800-433-5177/503-797-4207 and <http://www.pcisig.com>
- *OpenBoot 2.x Command Reference (802-3241)*
- *OpenBoot 3.x Command Reference (802-5837)*
- *PCI -Bus Binding to IEEE STD 1275-1994* is available at <http://playground.sun.com/1275>

The on-line version of this manual and FCode tools are available at

<http://www.sun.com/developers/driver>

Forth and Forth Programming

For further information about Forth and Forth programming:

- *Mastering Forth*, Anita Anderson and Martin Tracy, Brady Communications Co., Inc., 1989
- *ANSI Forth X3J14*
- *Forth: A Text and Reference*, Mahlon G. Kelly and Nicholas Spies. Prentice-Hall, 1986
- *Starting Forth*, Leo Brody. Forth, Inc., second edition, 1987
- *Forth: The New Model*, Jack Woehr. M & T Books, 1992
- Forth Interest Group <http://forth.org/fig.html>

What Typographic Changes Mean

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. system% You have mail.
AaBbCc123	What you type, contrasted with on-screen computer output	system% su Password:
<i>AaBbCc123</i>	Variable: replace with a real name or value	To delete a file, type <code>rm filename</code> .
<i>AaBbCc123</i>	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be root to do this.
[optional]	Optional command-line entry	<code>lpr [-Pprinter]</code>
Code samples are included in boxes and may display the following:		
ok	The OpenBoot Forth Monitor prompt	ok
%	UNIX C shell prompt	system%
\$	UNIX Bourne and Korn shell prompt	system\$
#	Superuser prompt, all shells	system#

Ordering Sun Documents

The SunDocs Order Desk is a distribution center for Sun Microsystems technical documentation. You can use major credit cards and company purchase orders. You can order documentation in the following ways:

In the U.S.A.

Call 1-800-247-0250

Fax 1-801-373-6798

Outside the U.S.A.

Call 1-801-342-3450

Fax 1-801-373-6798

World Wide Web: <http://www.sun.com/sundocs/catalog.html>

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. 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

Each SBus card must have a PROM whose contents identify the device and its characteristics.

The SBus card's PROM may also include an optional software driver that lets you use the card as a boot device or a display device during booting.

In addition to designing hardware, the process of developing SBus devices may include writing, testing, and installing FCode drivers for the device. These drivers, if present, perform three functions:

- Exercising the device during development, and verifying its functionality
- Providing the necessary driver to be used by the system boot PROM during power-up
- Providing device configuration information

In practice, these functions overlap substantially. The same code needed by the system boot PROM usually serves to significantly test the device as well, although additional code may be desired to fully verify proper behavior of the device. The PROM code is used before and during the boot sequence. After the boot sequence finishes, and while not using the OpenBoot Forth Monitor, most SBus device use is through Solaris drivers.

SBus device PROMs must be written in the FCode programming language, which is similar to ANS Forth. FCode is described in more detail in Chapter 3, "Elements of FCode Programming".

FCode PROM Format

An FCode PROM begins at address 0 in the SBus card's physical address space. Its size can range from 30 bytes up to 32K bytes. Typical sizes are 60 bytes (for a simple card that identifies itself but does not need a driver) and 5 to 15K bytes (for a card with a boot driver). It is good practice to make FCode boot drivers as short as is practical.

An FCode PROM must be organized as follows:

- Header (8 bytes: consisting of magic number, version number, length, checksum).
- Body (FCode program; 0 or more bytes).
- End Token (either End0, a zero byte, or End1, an alternative all 1s byte).

Interpreting FCode

For each SBus slot, the FCode program is interpreted during bootup as follows:

- Location 0 of the SBus PROM is read with an 8-bit or 32-bit access. If there is no response (as when there is no card in that slot), the slot is subsequently ignored.
- If the high-order byte of the value returned from the first access is not the FCode magic numbers 0xfd or 0xf1, the slot is subsequently ignored.
- If the high-order byte is 0xfd or 0xf1, the PROM is assumed to contain a valid FCode program. The FCode is then interpreted by starting at location 0 and reading one byte at a time, executing a procedure associated with each FCode value.
- Interpretation ceases when the FCode 0x00 or 0xff (End0 or End1) is encountered.

Device Identification

An FCode PROM must identify its device. This identification must include, at a minimum, the device name, used to link the device to its Solaris driver. Identification information may include additional characteristics of the device for the benefit of the operating system and the CPU boot PROM.

In most systems, the CPU's FCode interpreter will store each device's identification information in a *device tree* that has a node for each device. Each *device node* has a *property list* that identifies and describes the device. The property list is created as a result of interpreting the program in the FCode PROM.

Each property must have a name and a value. The name is a string and the value is an array of bytes, which may encode strings, numbers, and various other data types.

See Chapter 7, "Properties" for more information.

Creating and Executing FCode Definitions

Many FCode programs create executable routines, called *colon definitions (or methods)* that typically read from and write to device locations to control device functions. These definitions are also stored in the device tree node for that device.

Once defined, these routines may typically be executed under any of the following circumstances:

- Interactively at the OpenBoot `ok` prompt
- By the OpenBoot system (for using this boot or display system during system start-up)
- Automatically during FCode interpretation (for power-on initialization or other purposes)

PCI FCode Information



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 as follows:

Table 2-1 PCI FCode PROM Header Format

Header	Format
a.out header	32 bytes (needed by some utilities)
PCI expansion PROM header	28 bytes
PCI data structure	24 bytes
FCode	(8 Byte FCode header + FCode code bytes)

The `a.out` header is required in order to download an FCode image using `dload` or `boot` on a Solaris™ 1.x (SunOS 4.x) system if, for instance, during test/development of your FCode driver, you don't want to use a physical PROM.

If you are `dloading` or `booting` your FCode image on a Solaris 2.x system, you must replace the `a.out` header by an ELF header.

The `fakeboot` utility can add either an `a.out` or an ELF header based on parameters that you pass to `fakeboot`.

The PCI Expansion PROM Header Format

The PCI expansion PROM header format (28 bytes) is as follows:
(Values are shown in hexadecimal.)

Table 2-2 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 as follows:
(Values are shown in hexadecimal.)

Table 2-3 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

Table 2-3 PCI Expansion PROM Data Structure (Continued)

Byte Offset	Description /(Hex. value)
12-13	Revision level of code/data
14	Code type (01)
15	Indicator byte. For last image (80)
16-17	Reserved (00 00)

The following is a dump of initial bytes in a PCI FCode PROM with an a.out header in the first 32 bytes.

Code Example 2-1 PCI FCode PROM Dump

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 f1 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 of which the address and type are defined by phys.hi, phys.mid, and phys.low.

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

Table 2-4 Format of Physical Address in “reg” Property

```
phys.hi cell:    npt000ss bbbbbbbb ddddfff 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.

CPU PROM-Generated Properties

This section describes the properties created by the motherboard CPU PROM from the information 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`

It also generates `devsel-speed` from the information from the configuration space registers. The `"interrupts"` property is present if the Interrupt Pin register is non-zero. The following properties are present only if the corresponding capability is available from the device or if 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`

The `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 is assigned.

Adding a PCI Header to a PROM

You can see an example in test-pci.fth or add a PCI header using the following example:

```
tokenizer[
\ to add vendor id, device id and class code
h# 108e h# 1001 h# 020000 pci-header
\ to add vital product data
h# c000 pci-vpd-offset
\ to add pci-code-revision
h# 1234 pci-code-revision
ltokenizer
fcode-version3
-----
-----
-----
end0
```

FCode is a computer programming language defined by *IEEE Standard 1275-1994 Standard for Boot Firmware*. FCode is semantically similar to ANS Forth, but is encoded as a sequence of binary byte codes representing a defined set of Forth definitions.

FCode has these characteristics:

- The source format is machine and system independent.
- The binary format (FCode) is machine, system, and position independent.
- The binary format is compact.
- The binary format can be interpreted easily and efficiently.
- Programs are easy to develop and debug.
- The source format can easily be translated to binary format.
- The binary format can be translated back to source format.

Forth commands are called *words*, and are roughly analogous to procedures in other languages. Unlike other languages, such as C, which have operators, syntactic characters and procedures, in Forth every word is a procedure.

A Forth word is named by a sequence of between one to 31 printable characters. A Forth program is written as a sequence of Forth word names separated by one or more white space characters (such as spaces, tabs, or line

terminators). Forth uses a left-to-right reverse Polish notation, like some scientific calculators. The basic structure of Forth is: do this, now do that, now do something else, and so on.

New Forth words are defined as sequences of previously existing words. Subsequently, new words may be used to create still more words.

FCode is a byte-coded translation of a Forth program. Translating Forth source code to FCode involves replacing the Forth word names (stored as text strings) with their equivalent FCode numbers. The tokenized FCode takes up less space in PROM than the text form of the Forth program from which it was derived, and can be interpreted more easily and rapidly than the text form.

For purposes of this manual, the term FCode indicates both binary-coded FCode and the Forth programs written as ASCII text files for later conversion to binary-coded FCode.

Except where a distinction between the two forms is explicitly stated, the use of FCode in this manual can be assumed to apply equally to both FCode and Forth.

Colon Definitions

Three concepts are critical to understanding FCode (or Forth):

- A *colon definition* creates a new word with the same behavior as a sequence of existing words. A colon definition begins with a colon and ends with a semicolon.
- Once a new word has been created, it is immediately available, either for direct execution or for use in future colon definitions.
- Most parameter passing is done through a pushdown, last-in, first-out *stack*.

Normally, the action associated with an FCode Function is performed when the FCode Function is encountered. This is called *interpret state*. However, the state may switch between interpret state to *compile state*.

In interpret state, FCode Functions are executed as they are encountered. Interpret state operates until encountering a “:”. The word “:” does the following:

- Allocates an FCode Number and associates it with the name immediately following the colon.

- Switches to compile state.

In compile state, FCodes are saved for later execution, rather than being executed immediately. The sequence thus compiled is installed in the action table as a new word, and can be used later in the same way as if it were a built-in word.

Compile state continues until a “;” is read. The word “;” does the following:

- Compiles an end-of-procedure FCode word
- Switches to interpret state

After compilation, the newly-assigned FCode word can be either interpreted or compiled as part of yet another new word.

If you define a new word having the same spelling as an existing word, the new definition supersedes the older one(s), but only for subsequent usages of that word.

Here’s an example of a colon definition for a new FCode word `dac!`

```
: dac! ( data offset -- ) dac + r1! ;
```

Stack Operations

Each FCode word is specified by its effect on the stack and any side effects, such as accessing memory. Many FCode words affect the stack, by removing arguments from it, performing some operation, and putting the result(s) back on the stack.

A *stack comment*, included in the colon definition, describes the effect on the stack of the execution of an FCode word.

In the previous example, the stack comment, beginning with “(” and ending with “)”, shows that `dac!` takes two parameters from the stack, and doesn’t replace them with anything when it’s done.

Stack comments can be put anywhere in a colon definition. They should be included wherever their use will enhance the clarity of the definition.

The rightmost argument is on top of the stack, with any preceding arguments “under it”. In other words, arguments are pushed onto the stack in left to right order, leaving the most recent argument (the rightmost one in the stack diagram) on the top.

In a stack diagram, parameters shown to the left of the double dashes are expected to be on the stack prior to the execution of the word. Parameters shown to the right of the double dashes are those which are left on the stack after execution of the word. Stack comments use the same convention but detail changes to the stack during execution of the word.

Stack comments and stack diagrams are essentially the same thing. Stack diagrams show the net effect to the stack of any Forth word. Stack comments are embedded in the definition of a word and are used to convey intermediate stack results or changes.

A series of words that describe the behavior of `dac!` follow the stack comment in the preceding example. Executing `dac!` is the same as executing the list of words in its colon definition.

Note that FCode words are separated by spaces, tabs, or newlines, so in the previous example, “(data” is *not* the same as “(data”. Any visible character is part of a word, and not a separator.

Programming Style

Some people have described Forth as a write-only language. While it sometimes ends up that way due to poorly-written or uncommented code, it *is* possible to write Forth (and FCode) programs that can be read easily and understood. Well-written Forth programs will meet these criteria. See Appendix C, “Coding Style” for detailed information about the style used in the existing OpenBoot FCode source base.

Although case is not significant, by convention FCode is written in lower case.

Commenting Code

Comment code extravagantly, then consider adding more comments. The comments will help with maintenance of the code, and don’t add to the final size of the resulting FCode PROM.

Adopt the useful convention of using “()” for stack comments and “\” for other descriptive text and comments.

In comments, describe the purpose of the Forth words, any interface assumptions and requirements, and unusual aspects of the algorithm you used. Try to avoid simply translating low-level details of the code into English. Comments like, “increment the variable” are rarely helpful.

Coding Style

By studying the examples in this book, you can note the indentation and phrasing style that is widely used in OpenBoot source code. Adoption of this style will allow your Forth code to be read more easily by the many programmers who are accustomed to the style.

Definition Length

Keep word definitions short. If your definition exceeds half a page, it should be rewritten as two or more smaller definitions. This will help to make each definition more readable. Readable code is easier to maintain.

A *good* size for a word definition is one or two lines of code. Keeping definitions short and limited in functionality improves readability, speeds debugging and increases the likelihood that the word will be reusable. Remember: reuse of Forth words is a principal contributor to compact PROM images.

Stack Comments

Always include stack comments in word definitions. It can be useful to compare intended function with what the code really does. Here's an example of a word definition with acceptable style.

```

\ xyz-map  establishes a virtual-to-physical mapping for each of
the
\ useful addressable regions on the board

0 value status-register

: xyz-map  ( -- )

\ Base-address Size create-mapping then save virtual address

  my-address 4 map-low                ( virtaddr )
  to status-register                  ( )
  my-address 10.0000 d+ frame-buf-size map-low  ( virtaddr )
  to frame-buffer-adr                 ( )

;

```

Note the stack diagram following the word `xyz-map`, and the use of stack comments in the word's definition code.

Stack diagrams are generally written using descriptive parameter names to clarify correct usage. See the table below for stack parameter abbreviations used in this manual

Table 3-1 Stack Parameter Abbreviations

Notation	Description
	Alternate stack results shown with space, e.g. (input -- addr len false result true).
	Alternate stack items shown without space, e.g. (input -- addr len 0 result).
??? or ?	Unknown stack item(s)
...	Unknown stack item(s). If used on both sides of a stack comment, means the same stack items are present on both sides.
< > <space>	Space delimiter. Leading spaces are ignored.
a-addr	Variable-aligned address

Table 3-1 Stack Parameter Abbreviations (Continued)

Notation	Description
addr	Memory address (generally a virtual address).
addr len	Address and length for memory region
byte bxxx	8-bit value (low order byte in a 32-bit word).
char	7-bit value (low order byte), high bit unspecified.
cnt or len or size	Count or length
dxxx	Double (extended-precision) numbers. 2 stack items, hi quadlet on top of stack.
<eol>	End-of-line delimiter
false	0 (false flag)
ihandle	Pointer for an instance of a package
n or n1 or n2 or n3 nxxx	Normal signed values (32-bit)
nu or nul	Signed or unsigned values (32-bit)
<nothing>	Zero stack items
phandle	Pointer for a package
phys	Physical address (actual hardware address)
phys.lo phys.hi	Lower/upper cell of physical address
pstr	Packed string
quad or qxxx	Quadlet (32-bit value)
qaddr	Quadlet (32-bit) aligned address
{text}	Optional text. Causes default behavior if omitted.
"text<delim>"	Input buffer text, parsed when command is executed. Text delimiter is enclosed in <>.
true	-1 (true flag)
uxxx	Unsigned value, positive values (32-bit)
virt	Virtual address (address used by software)
waddr	Doublet (16-bit) aligned address
word or wxxx	Doublet (16-bit value, low order two bytes in a 32-bit word)
x or xl	Arbitrary stack item
x.lo x.hi	Low/high significant bits of a data item

Table 3-1 Stack Parameter Abbreviations (Continued)

Notation	Description
xt	Execution token
xxx?	Flag. Name indicates usage (e.g. done? ok? error?).
xyz-str xyz-len	Address and length for unpacked string
xyz-sys	Control-flow stack items, implementation-dependent
(C: --)	Compilation stack diagram
(--) or (E: --)	Execution stack diagram
(R: --)	Return stack diagram

A Minimal FCode Program

If a peripheral bus card is not needed during the boot process, a minimal FCode program that merely declares the name of the device and the location and size of on-board registers will often suffice.

A minimum program for an SBus device is:

```

fcode-version1
" SUNW,bison" encode-string " name" property

my-address h# 20.0000 + my-space encode-phys
h# 100 encode-int encode+
" reg" property

end0

```

Note the following about this SBus example:

- `my-address` and `my-space` each leave only a single number on the stack representing the *phys.lo phys.hi* address representation of an SBus node. (The value of `#address-cells` is 2 for SBus which is reflected by this format.)
- An offset of 0x200000 is being added to the value returned by `my-address`.
- The *size* argument of `" reg"` is a single number (since `#size-cells` is 1 for SBus) reflecting SBus's 32-bit address space.

The example program above creates a " name" property that will be used to identify the device whose value is "SUNW,bison". Begin the name attribute's value with an identification of your company. The preferred form of this identification is an *organizationally unique identifier* (OUI), a sequence of six uppercase hexadecimal digits assigned by the IEEE Registration Authority Committee. OUIs are guaranteed to be unique world-wide. (For more information about obtaining an OUI, please see the glossary entry for name in *IEEE Standard 1275-1994 Standard for Boot Firmware*.)

As an alternative to the OUI, you may use a sequence of from one to five uppercase letters representing the stock symbol of your company on any stock exchange whose symbols do not conflict with the symbols of the New York Stock Exchange and the NASDAQ Exchange. All stock exchanges in the United States satisfy this requirement. If a non-US company's stock is traded on US stock exchanges by "depository equivalents", those symbols also satisfy this requirement.

The preceding SBus example program can also be written using the following shorthand form. The FCode program generated will be equivalent to the minimum SBus program given above.

```
fcode-version1
" SUNW,bison" name
my-address h# 20.0000 + my-space h# 100 reg
end0
```

FCode Classes

There are four general classes of FCode source words:

Table 3-2 FCode Source Word Classes

Primitives	These words generally correspond directly to conventional Forth words, and implement functions such as addition, stack manipulation, and control structures.
------------	--

Table 3-2 FCode Source Word Classes

System	These are extension words implemented in the boot PROMs, and implement functions such as memory allocation and device property reporting.
Interface	These are specific to particular types of devices, and implement functions such as <code>draw-character</code> for a display device.
Local	These are private words, implemented and used only by the device that created the definition.

Each FCode primitive is represented in a peripheral card’s PROM as a single byte. Other FCodes are represented in the PROM as two consecutive bytes. The first byte, a value from 1 to 0x0f, may be thought of as an escape code.

One-byte FCode numbers range in value from 0x10 to 0xfe. Two-byte FCode numbers begin with a byte in the range 0x01 to 0x0f, and end with a byte in the range 0x00 to 0xff. The single-byte values 0x00 and 0xff signify “end of program” (either value will do; conventionally, 0x00 is used).

Currently-defined FCodes are listed in functional groups, in alphabetic order by name and in numeric order by FCode value in Appendix A, “FCode Reference”.

Primitive FCode Functions

There are more than 300 primitive FCode functions, most of which exactly parallel ANS Forth words, divided into three groups:

- FCode words that generate a single FCode byte
- tokenizer macros
- tokenizer directives

Primitive FCode functions that have an exact parallel with standard ANS Forth words are given the same name as the equivalent ANS Forth word. Chapter 14, “FCode Dictionary”, contains further descriptions of primitive FCodes.

There are about another 70 tokenizer macros, most of which also have direct ANS Forth equivalents. These are convenient source code words translated by the tokenizer into short sequences of FCode primitives.

Tokenizer directives are words that generate no FCodes, but are used to control the interpretation process. Tokenizer directives include the following words:

- decimal, hex, and octal
- d#, h#, and o#
- headers and headerless
- \ and (
- .(
- alias

System FCode Functions

System FCode functions are used by all classes of FCode drivers for various system-related functions. System FCode functions can be either *service* words or *configuration* words.

- Service words are available to the device's FCode driver when needed for functions such as memory mapping or diagnostic routines.
- Configuration words are included in the driver to document characteristics of the driver itself. These *properties* are made available for use by the operating system.

Interface FCode Functions

Interface FCode functions are standard routines used by the workstation's CPU to perform the functions of the peripheral card's device. Different classes of devices will each use only the appropriate set of interface FCodes.

For example, in order to display a character on the screen, OpenBoot calls the interface FCode `draw-character`. Previously, the FCode driver for the device controlling that screen must have assigned a device-specific implementation to `draw-character`. It does this as follows:

```
: my-draw ( char -- ) \ "local" word to draw a character.
  ...                \ Definition contents.
;                    \ end of my-draw definition.
: my-install ( -- )  \ local word to install all interfaces.
  ...
  ['] my-draw to draw-character
  ...
;
```

When `my-install` executes, `draw-character` is assigned the behavior of `my-draw`.

Local FCode Functions

Local FCode functions are assigned to words defined in the body of an FCode program. There are over 2000 FCode byte values allocated for local FCodes. The byte values are meaningful only in the context of a particular driver. Different drivers reuse the same set of byte values.

Debugging and Testing FCode Programs



Packaging PCI FCode

Example: In trying to test a new version of an FCode program, a developer creates a new package as follows:

```
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 doing an `ls`, it is obvious that there are now TWO packages corresponding to the card:

```
ok ls
ffd70c00 my-network@4,0
ffd6e860 my-network@4,0
ok
```

To get rid of or override the original package so that the downloaded code is executed, remove the PCI card PROM. The CPU PROM will still create a device node for the card, but the "name" property will have a value of the form

```
"pci<DDDD>, <VVVV>"
```

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

System Flags and FCode Debugging

Example: When using Sun systems debugging flags to aid in debugging FCode, set the NVRAM variable `fcode-debug?` to true to keep the headers for those words in your source preceded by headers.

Also, some CPU PROMS have a variable named "`fcode-verbose?`" to display each FCode as it is being read at probe time by your CPU PROM's token interpreter. To turn it on, before you probe your FCode, do:

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

To set it from NVRAMRC, do:

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

Some CPU PROMs have `pcimsg?` and `probemsg?` variables to give additional PCI-related information. You can turn them ON in a way similar to that described above. `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?`. Also, in future PROMs, the behavior of `pcimsg?`, `probemsg?` and `fcode-verbose?` may change, including the possibility of deletion.

FCode Source

An FCode source file is essentially a Forth language source code file. The basic Forth words available to the programmer are listed in Chapter 14, “FCode Dictionary”.

FCode programs have the following format:

```
\ Title comment describing the program that follows
fcode-version3
< body of the FCode program >
end0
```

`fcode-version3` is a macro which directs the tokenizer to create an FCode header. For a description of the FCode header see “FCode Binary Format”. `fcode-version3` produces a header including the `start1` FCode. The macros `fcode-version3` and `fcode-version2` produce a header containing the `start1` FCode. Whereas `fcode-version1` produces a header with `version1` FCode. To use all IEEE 1275 compliant Fcodes, use `fcode-version3`. (Since OpenBoot version 1 systems only recognize `version1`, plug-in device FCode that must run in OpenBoot version 1 systems must use `fcode-version1`.)

`end0` is an FCode that marks the end of an FCode program. It must be at the end of the program or erroneous results may occur.

The comment in the first line is not necessary in many cases but it is recommended since it allows some OpenBoot tools to recognize the file as a Forth source file.

Tokenizing FCode Source

The process of converting FCode source to FCode binary is referred to as *tokenizing*. A tokenizer program converts FCode source words to their corresponding byte-codes, as indicated in Chapter 14, “FCode Dictionary”. A tokenizer program with instructions describing its use is available from

<http://www.sun.com/developers/driver>

under the tools icon as Fcode tokenizer and tools.

An FCode program's source can reside in multiple files. The `fload` tokenizer directive directs the tokenizer input stream to load another file. `fload` acts like an `#include` statement in C. When `fload` is encountered, the tokenizer begins processing the file named by the `fload` directive. When the named file is completed, tokenizing continues with the file that issued the `fload`. `fload` directives may be nested.

Typically, the tokenizer produces a file in the following format:

- Header - 32 bytes
- FCode header - 8 bytes
- FCode binary - remainder of file

The header has the following format:

- 4 bytes - 0x01030107 (hexadecimal)
- 4 bytes - Size in bytes of the FCode binary
- 4 bytes - 0x0
- 4 bytes - 0x0
- 4 bytes - 0x0
- 4 bytes - Load point of the file
- 4 bytes - 0x0
- 4 bytes - 0x0

You can use this file to load either an FCode PROM or system memory for debugging as described in "Using the Forth Monitor to Test FCode Programs".

The load point of the file is not used when burning an FCode PROM, but is used by Forth Monitor commands that load FCode files into system memory. The tokenizer available from SunExpress sets the load point to be the recommended 0x4000 address.

FCode Binary Format

The format of FCode binary that is required by the OpenBoot *FCode evaluator* is as follows:

Table 4-1 FCode Binary Format

Element	Structure
FCode header	Eight bytes
Body	0 or more bytes
End byte-code	1 byte, the <code>end0</code> byte-code

The format of the FCode header is:

Table 4-2 FCode Header Format

Byte(s)	Content
0	One of the FCodes: <code>start0</code> , <code>start1</code> , <code>start2</code> , <code>start4</code> , <code>version1</code>
1	reserved
2 and 3	reserved
4 through 7	count of bytes in the FCode binary image including the header

Testing FCode Programs on the Target Machine

Once you have created the FCode binary, you can test it using the OpenBoot Forth Monitor. The Forth Monitor provides facilities to allow you to load your program into system memory and direct the FCode evaluator to interpret it from there. This allows you to debug your FCode without having to create a PROM and attach it to your plug-in board for each FCode revision during the debug process. See the *OpenBoot 3.x Command Reference* for complete documentation of the use of the Forth Monitor.

The FCode testing process generally involves the following steps:

1. Configuring the target machine. This includes installing the hardware associated with the FCode program in the target machine and powering-up the machine to the Forth Monitor.

2. Loading the FCode program into memory from a serial line, a network, a hard disk, or a floppy disk.
3. Interpreting the FCode program to create a device node(s) on the OpenBoot device tree.
4. Browsing the device node(s) to verify proper FCode interpretation.
5. Exercising the FCode program's device driver *methods* compiled into the device node, if any.

If the FCode program does not include any methods which involve using the actual hardware (for example, a driver which only publishes properties) then the program can be tested without installing the hardware.

Configuring the Target Machine

Setting Appropriate Configuration Parameters

Before powering-down the target machine to install the target hardware, a few NVRAM configuration variables should be set to appropriate values. You can set them from the Forth Monitor as follows:

```
ok setenv auto-boot? false
ok setenv fcode-debug? true
```

Setting `auto-boot?` to `false` tells OpenBoot not to boot the OS on a machine reset but rather to enter the Forth Monitor at the `ok` prompt.

Setting `fcode-debug?` to `true` tells the OpenBoot FCode evaluator to save the names of words created by interpreting FCode words which were tokenized with `headers on`. This is in addition to words defined after the tokenizer processed an `external` directive (in other words, words whose names are always saved). `fcode-debug?` defaults to `false` to conserve RAM space in normal machine operation.

Modifying the Expansion Bus Probe Sequence

The start-up sequence in the machine's OpenBoot implementation normally examines all expansion buses for the presence of plug-in devices and their on-board FCode PROM programs. It then invokes the FCode evaluator to interpret any programs found. This process is called *probing*.

When using the Forth Monitor to load and interpret an FCode program in system memory, it is better to configure OpenBoot to avoid probing that device automatically. The probing can then be done manually (as explained later) from the Forth Monitor.

Configuring an OpenBoot implementation to avoid probing a given slot on a given expansion bus can be done in various implementation-dependent ways. That is, they will be different for different systems and different expansion buses.

Many machines with SBus have an NVRAM configuration variable named `sbus-probe-list`. It defines which SBus card slots will be probed during start up and the order in which they will be probed.

For example, a machine with four SBus slots might have the `sbus-probe-list` configuration variable set to a default value of 0123. Setting `sbus-probe-list` to 013 directs OpenBoot during start-up to probe first SBus slots 0, 1, and 3. This leaves SBus slot 2 unprobed, free for use by the device under development.

Methods to prevent probing a given slot for other types of expansion buses can involve using the NVRAMRC script. Among other uses, an NVRAMRC script can:

- Patch an implementation-specific OpenBoot word that defines the bus's probe sequence
- Modify a property of the expansion bus's device node that describes the sequence.

After the FCode program is debugged and programmed in PROM on the device, you can do a full system test (including automatic probing of the new device), by restoring the expansion bus probing configuration to the default.

Getting to the Forth Monitor

After completing the configuration described above, power the machine down and install the device. Then power the system up. The display should stop scrolling at the `ok` prompt, ready to accept Forth Monitor commands.

Using the Command Line Editor of the Forth Monitor

Refer to the *OpenBoot 3.x Command Reference* for a list and description of the line-editing commands available with the Forth Monitor.

Using the Forth Monitor to Test FCode Programs

Directions for using the Forth Monitor to download files to system memory are provided in the *OpenBoot 3.x Command Reference*. Common package-related commands are shown below.

Table 4-3 Common Package-related Commands

Command	Stack Diagram	Function
<code>begin-package</code>	(arg-addr arg-len reg-addr reg-len path-addr path-len --)	Initializes device tree for executing FCode.
<code>end-package</code>	(--)	Completes a device tree entry and returns to the Forth Monitor environment.
<code>open-dev</code>	(path-addr path-len --)	Opens the specified device node and all of its parents.
<code>device-end</code>	(--)	Closes the current node and returns to the Forth Monitor environment.
<code>select-dev</code>	(path-addr path-len --)	Opens the specified device node and all of its parents, and makes the device the current instance.

Table 4-3 Common Package-related Commands (Continued)

Command	Stack Diagram	Function
<code>unselect-dev</code>	(--)	Closes the specified device node and all of its parents, and unselects the active package and current instance leaving none selected.
<code>set-args</code>	(arg-addr arg-len reg-addr reg-len --)	Sets values returned by <code>my-args</code> , <code>my-space</code> and <code>my-address</code> for the current node.
<code>execute-device-method</code>	(... path-addr path-len cmd-addr cmd-len -- ... ok?)	Executes the named command in the specified device tree node.

Using `dload` to Load from Ethernet

`dload` loads files over Ethernet at a specified address, as shown below.

```
ok 4000 dload filename
```

In the above example, *filename* must be relative to the server's root. Use 4000 (hex) as the address for `dload` input.

Note – One can use any value other than 4000 as long as it has been properly mapped.

FCode programs loaded with `dload` must be in the format described in “Tokenizing FCode Source”.

`dload` uses the trivial file transfer protocol (TFTP), so the server may need to have its permissions adjusted for this to work.

Using `dlbin` to Load From Serial Port A

`dlbin` may be used to load files over serial line A. Connect the target system's serial port A to a machine that is able to transfer a file on request. The following example assumes a `tip` window setup on a Sun system which will provide the FCode file. (See the *OpenBoot 3.x Command Reference* for information on setting `tip` connections.)

1. At the `ok` prompt, type:

```
ok dlbin
```

2. In the `tip` window of the other system, send the file:

```
~C (local command) cat filename  
(Away two seconds)
```

The `ok` prompt will reappear on the screen of the target system.

FCode programs loaded with `dlbin` must be in the format described in “Tokenizing FCode Source”. `dlbin` loads the files at the entry point indicated in the file header. It is suggested that this address be `0x4000`.

Using `boot` to Load From Hard Disk, Diskette, or Ethernet

You can also load an FCode program with `boot`, the command normally used to boot the operating system. Use the following format:

```
ok boot [device-specifier] [filename] -h
```

device-specifier is either a full device path or a device alias. See the *OpenBoot 3.x Command Reference* for information on device path and aliases.

For a hard disk or diskette partition, *filename* is relative to the resident file system. See the *OpenBoot 3.x Command Reference* for information on creating a bootable diskette. For a network, *filename* is relative to the system's root partition on its root server. In both cases, the leading `/` must be omitted from the file path.

The `-h` flag specifies that the program should be loaded, but not executed. This flag must be included since otherwise `boot` will attempt to automatically execute the file assuming it is executable binary.

`boot` uses intermediate booters to accomplish its task. When loading from a hard disk or diskette, the OpenBoot firmware first loads the disk's boot block, which in turn loads a second-level booter. When loading over a network, the firmware uses TFTP to load the second-level booter. In both cases, *filename* and `-h` are passed to these intermediate booters.

The output file produced by a tokenizer may need to be converted to the format required by the secondary boot program. For example, Solaris 2.x intermediate booters require ELF format. `fakeboot`, a program available from SunExpress, may be useful in this process.

The location in memory where the FCode program is loaded depends on the secondary boot program and the `fakeboot` program.

Using `d1` to Load Forth Over Serial Port A

Forth programs loaded with `d1` must be ASCII files.

To load the file over the serial line, connect the system's serial port A to a machine that is able to transfer a file on request. One method is to set up a TIP window on another Sun system. (See *OpenBoot 3.x Command Reference* for information on this procedure.) The following example assumes a TIP window setup.

1. At the `ok` prompt, type:

```
ok d1
```

2. In the TIP window of the other system, send the file, and follow it with a Control-D to signal the end of the file.

```
~C (local command) cat filename  
(Away two seconds)  
^D
```

The `ok` prompt appears on the screen of the system to which the file is loaded.

d1 normally loads the file at 4000 (hex). The file is automatically interpreted after it is loaded.

Using the Forth Monitor to Interpret an FCode Program

FCode program interpretation involves creating a device node on the device tree. Device nodes are also known as *packages*. Creating a device node from downloaded FCode involves the following steps:

1. Set up the environment with `begin-package`.

For example, a `begin-package` call for creating a device node for a SBus card installed in SBus Slot 3 of a SPARCstation 2 looks like:

```
ok 0 0 " 3,0" " /sbus" begin-package
```

In the example, the string, `/sbus`, indicates that the device node which will be created by the FCode program is to be a child node of the `/sbus` node in the device tree.

In general, parent nodes, which support child nodes, can be used as this argument to `begin-package`. The device node defined by the FCode program will be created as a child of that node. Give the full device pathname from the root node. Other types of parent nodes define different address spaces. Another example of an SBus parent node is on a SPARCstation 10 where its device pathname is `/iommu/sbus`.

In the example, the string, `" 3,0"` indicates the SBus slot number, 3 and byte-offset 0 in the slot's address space where the device node is to be based.

In general, this string is a pair of values separated by a comma which identify the physical address associated with the expansion slot. The form of this physical address depends on the physical address space defined by the parent node. For children of an SBus node, the form is `slot-number, byte-offset`. Other parent nodes will define different address spaces.

The physical address pair value is retrieved in the FCode program with both the `my-address` and `my-space` FCodes. The slot ID string is converted to a binary form consisting of three values. Those values can be retrieved with the FCode program by using `my-address` for the *phys.lo* and *phys.mid* components and `my-space` for the *phys.hi* component.

In the preceding example, the initial 0 0 represents a null argument string passed to the FCode program.

This argument string is retrieved in the FCode program with the `my-args` FCode. Generally, FCode programs do not take arguments at interpretation time so this will usually be the null string.

`begin-package` is defined as:

```
: begin-package ( arg-addr arg-len reg-addr reg-len dev-addr dev-len -- )
  select-dev new-device set-args
;
```

`select-dev (parent-dev-addr parent-dev-len --)` opens the input device node (the parent node) and makes it the *current instance*.

`new-device (--)` initializes a new device node as a child of the currently active node and makes it the current instance.

`set-args (arg-addr arg-len reg-addr reg-len --)` sets the values returned by `my-args`, `my-space`, and `my-address` for the current instance.

2. Interpret the loaded FCode with `byte-load`

`byte-load` is the Forth Monitor command that invokes the FCode evaluator to compile the FCode program into the current instance.

For FCode programs downloaded with `byte-load` use:

```
ok <fcode-stat-address> ' c@ byte-load
```

`load-base` is the system default load address. The argument, `' c@`, tells `byte-load` to use `c@` as the access routine for reading the FCode.

3. Close the environment with `end-package`

`end-package` finishes up the creation of the device tree node.

```
ok end-package
```

It is defined as:

```
: end-package ( -- ) finish-device unselect-dev ;
```

`finish-device (--)` Completes the device tree node initialized by `new-device` and changes the current instance to the parent node.

`unselect-dev (--)` Closes the parent device tree node and returns to the normal Forth Monitor environment. That is, there is no longer a current instance or active package.

Using the Forth Monitor to Browse a Device Node

The Forth Monitor has many built-in commands to navigate the device tree. Table 4-4 lists the Forth Monitor commands supporting device node browsing:

Table 4-4 Commands for Browsing the Device Tree

Command	Description
<code>.properties</code>	Display the names and values of the current node's properties.
<code>dev device-path</code>	Choose the indicated device node, making it the current node.
<code>dev node-name</code>	Search for a node with the given name in the subtree below the current node, and choose the first such node found.
<code>dev ..</code>	Choose the device node that is the parent of the current node.
<code>dev /</code>	Choose the root machine node.
<code>device-end</code>	De-select the current device node, leaving no node selected.
<code>" device-path" find-device</code>	Choose device node, similar to <code>dev</code> .
<code>get-inherited-property</code>	(<i>name-addr name-len -- true value-addr value-len false</i>) Return property value of current instance or its parents
<code>get-my-property</code>	(<i>name-addr name-len -- true value-addr value-len false</i>) Return property value of current instance.
<code>ls</code>	Display the names of the current node's children.
<code>pwd</code>	Display the device path that names the current node.
<code>see wordname</code>	Decompile the specified word.
<code>show-devs [device-path]</code>	Display all the devices known to the system directly beneath a given level in the device hierarchy. <code>show-devs</code> used by itself shows the entire device tree.
<code>words</code>	Display the names of the current node's methods.

Once a device node has been created, you can use the Forth Monitor to browse the node. See the *OpenBoot 3.x Command Reference* for a more complete discussion. Here is a brief synopsis of the available commands:

- `show-devs` displays all known devices in the device tree.
- `dev` sets the active package to a named node so its contents can be viewed. For example, to make the ACME company's SBus device named "ACME,widget" the active package:

```
ok dev /sbus/ACME,widget
```

- `find-device` is essentially identical to `dev` differing only in the way the input path is passed.

```
ok " /sbus/ACME,widget" find-device
```

- `.properties` displays the names and values of all the properties created for the active package.
- `get-my-property` returns the value of the specified property from the active package.
- `get-inherited-property` returns the location and length of the property value array of the specified property from the active package or its parents. `dump` can then be used to display the property value array.
- `ls` displays the names of all child nodes, if any, of the active package.
- `words` shows the names of the device node methods, if any, created by the FCode program. It shows all words which were defined with `external` and, if `fcode-debug?` was true when the FCode was interpreted, the words defined with `headers`.
- `see wordname` displays the source code (without comments) for `wordname`.
- `device-end` reverses the effects of the `dev` or `find-device` command.
- `pwd` displays the device path of the active package.

Using the Forth Monitor to Test a Device Driver

The Forth Monitor provides the capability to test the methods of an FCode program by allowing you to execute individual methods from the Forth Monitor prompt.

Device Node Methods

Using `select-dev`

`select-dev` initializes an execution environment for the methods of the package specified by its stack arguments. It allows the user to subsequently execute the device node's methods directly by name. For example:

```
ok " /sbus/ACME,widget" select-dev
```

`select-dev` performs the following steps:

1. Effectively calls "`dev /sbus/ACME,widget`" to make the named device the active package. This enables the recognition of the device methods by the Forth Monitor.
2. Establishes a chained set of package instances for each node in the path. In particular, this makes the package's instance-specific data items available to its methods.
3. Opens all device nodes in the path by calling the open method of each. `select-dev` assumes open (and close) methods in each node in the path, so the device node under test must have one.

Once these steps are performed, you can execute the methods of the current device node by typing their names at the prompt. For example:

```
ok clear-widget-register  
ok fetch-widget-register .  
0
```


As is generally true of the Forth language, if execution of a method exposes an error in the code, the error can be isolated by executing the component words of the method step-by-step. Use `see` to decompile the method, then type the component words individually until the error is apparent. For example:

```
ok see clear-widget-register
: clear-widget-register
  enable-register-write
  0 widget-register r1!
  disable-register-write
;
ok enable-register-write
ok 0 widget-register r1!
ok disable-register-write
```

This process can be performed recursively by decompiling the component words and then individually executing their component words. This is much easier if most of the words were defined with the `headers` directive, since `see` can then display the names of the component words instead of hexadecimal codes.

This process is also enhanced by executing `showstack`. `showstack` causes the stack's contents to be displayed prior to every `ok` prompt. For example:

```
ok 1 2
ok showstack
1 2 ok . clear 3 4
2
3 4 ok
```

Device nodes can also be modified as needed with any of the following techniques:

- Entering new methods definitions. These methods are compiled into the device node like the methods in the FCode program that created the node.

- Redefining a method to include a function neglected in the first definition. (Previously defined words using the original definition of the method are unaffected.) For example:

```
ok : open open initialize-widget-register-2 ;
```

In general, such redefinitions affect only external uses of the named method (for instance, calls from other packages via `$call-method` and the like) and interactive use via the Forth Monitor. Previously compiled calls to the method in the same package are unaffected unless the method is called by name (for example, with `$call-self`).

- Use `patch` to edit word definitions. Such patches affect all uses of the method, both internal and external. (See the *OpenBoot 3.x Command Reference Manual* for information about using `patch`.)
- Resetting the machine causes all such corrections to be lost. Consequently, once your words are debugged you'll probably want to include any modifications in the FCode program source.

`unselect-dev` reverses the effects of `select-dev` by calling the `close` method of each device in the path of the current active node, destroying the package instance of each node, and returning to the normal Forth Monitor environment. Execute `unselect-dev` as follows:

```
ok unselect-dev
```

Using `begin-select-dev`

Sometimes, `select-dev` will not work because the `open` method of a newly-written package does not work correctly. In this case, `begin-select-dev` can be used since it does everything that `select-dev` does *except* for opening the last child node. For example:

```
ok " /sbus/ACME,widget" begin-select-dev
```

Using `execute-device-method`

`execute-device-method` executes a method directly from the normal Forth Monitor environment. That is, it is not necessary to manually make the device node the current instance before executing the method. For example:

```
ok " /sbus/ACME,widget" " test-it" execute-device-method
```

`execute-device-method` returns `false` if the method could not be executed; otherwise it returns `true` on top of whatever results were placed on the stack by the successful execution of the method.

`execute-device-method` performs the following steps:

1. Establishes a chained set of package instances for each node in the path. In particular, this makes an instance of all data items of the device node available to its methods.
2. Opens all device nodes in the named device path *except* the last device node in the pathname.
3. Invokes the named method.
4. Closes all the device nodes in the path (except the last one) destroying their package instances.
5. Restores the current instance to the one that was current prior to beginning this process.
6. Restores the active package to the one that was active prior to beginning this process.
7. Returns the results.

Note that, in contrast to `select-dev`, `execute-device-method` does not call the `open` method of the last device node in the path. Consequently, any method invoked in this manner must not require any pre-established state which normally is created by `open`.

In summary, `execute-device-method` is provided to allow execution of device node methods designed to provide their own state initialization, and therefore to execute without previous execution of the `open` method. A typical example is a `selftest` method.

Using apply

`apply` provides an alternative manner of invoking `execute-device-method` in that it takes its arguments from the input stream instead of from the stack. The above example would be invoked with `apply` as follows:

```
ok apply test-it /sbus/ACME,widget
```

Since `apply` invokes `execute-device-method`, all of the restrictions listed above for `execute-device-method` must be followed.

Testing FCode Programs in Source Form

The Forth Monitor enables you to skip the tokenizer and download FCode program source directly. This practice is not recommended since the only advantage is to save a small amount of time tokenizing the program. There are also some disadvantages:

- It may cause problems in the long run since generally the Forth Monitor recognizes a larger number of words than the FCode evaluator does. So the FCode program developer who tests with FCode source may develop and test a program only to find that some of the words used are not FCode words, and will not be accepted by the tokenizer and the FCode evaluator.
- To load source you should comment out `fcode-version1` and `end0`.
- Since the download commands accept only one file, replace `fload` with the actual file.

To load an ASCII Forth source file over serial line A, you use the command `d1`. In addition to loading the file over the serial line, `d1` compiles the Forth source while it is loading, without requiring an extra command. Therefore, you must execute `begin-package` before downloading. See “Using `d1` to Load Forth Over Serial Port A” for details.

Producing an FCode PROM

The output of the tokenizer program is used to make an actual FCode PROM. If your PROM burning tools do not accept the raw binary format of the tokenizer, you may need to develop a format conversion utility.

Exercising an Installed FCode PROM

You can either let OpenBoot automatically evaluate the FCode program from the PROM or you can remove the device from the OpenBoot probing as discussed earlier in “Configuring the Target Machine”.

The same process discussed for testing FCode programs loaded to system memory can be used to test FCode programs already loaded into PROM on the device.

If you take the device out of the probing sequence, a device node can be built manually as in the following example for a device installed in SBus slot 1:

```
ok 10000 constant rom-size
ok " /sbus" select-dev
ok " 1" decode-unit      ( phys.lo phys.mid phys.hi )
ok rom-size map-in      ( virt )
ok new-device           ( virt )
ok " " " 1,0" set-args  ( virt )
ok dup 1 byte-load     ( virt )
ok finish-device       ( virt )
ok rom-size map-out
ok unselect-dev
```

This is essentially the same sequence as outlined for evaluating FCode loaded into system memory, except that you must map in and map out the FCode PROM by using the `decode-unit`, `map-in`, and `map-out` methods of the parent device node. For more information about these methods, see Chapter 10, “Memory-Mapped Buses”.

You can browse the device node and exercise the device methods in the same way as described earlier. You can also define new methods and patch existing ones. Of course, these modifications will only remain until a system reset.

select-dev-Generated Errors

To debug your FCode/device in the case of errors during the use of `select-dev` on the device, do the following.

Add a dummy `open` method to your device node's FCode if you want to be able to select (open) the device to map the device in at the `ok` prompt and look at the device registers, etc.:

```
ok dev /pci..../<device-node>
ok : open true ;
( This may generate a message about open not being unique)
ok device-end
```

Now you can use "`select-dev`" to open/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 `r1@` and other register access words return the data in the way you expect.

Helpful Testing and Debugging Hints

5 

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

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 in the configuration space are set by the CPU PROM.

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

System Cache Line Size

To determine the system's cache line size from the FCode to write into the cache-line-size configuration space register of your PCI device, look in the cache-line-size register in the configuration space; it refers to the cache line size supported by the PCI device.

Sun Ultra-30 UPA/PCI-Related Nodes

The PCI-related nodes on the Sun Ultra-30 UPA/PCI system are `/pci@1f,4000` and `/pci@1f,2000`. `pcia` and `pcib` as needed for the NVRAM variables `pcia-probe-list` and `pcib-probe-list` are determined in the following manner.

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

To get a human-readable value for that property, do the following:

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

For example, for a PCI bus at `/pci@1f,2000`:

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

will display something like:

```
6
pcia slot 1pcia slot 2
```

This is an indication that devices under `/pci@1f,2000` relate to `pcia`.

In a Sun Ultra-30 UPA/PCI with 4 plug-in PCI slots, only slot 1 is physically present for `pci@1f,2000`. It can also support 66 Mhz., 64 bit PCI devices.

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

This is an indication that devices under `/pci@1f,4000` relate to

pcib,

In a Sun Ultra-30 UPA/PCI with 4 plug-in PCI slots:

- Slots 2, 4 and 5 under `/pci@1f,4000` support 33 Mhz., 32 bit PCI devices.
- Slot 3 under `/pci@1f,4000` is for an on-board SCSI device.

Note that the value of the "slot-names" property differs for different systems. Some systems may not indicate which PCI bus is which by the value of the "slot-names" property.

Also in different releases of the PROM for the same system, the value of the "slot-names" property may change. You may need to refer to the system documentation for details about 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 and/or by plugging PCI card(s) in different slot(s).

Finding and Using Physical Addresses

To find and use physical addresses to access, for instance, configuration space registers on a Sun Ultra-30 UPA/PCI system, do MMU bypassing with the choice of 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-30 UPA/PCI systems, a PCI device's 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:

`0000.0000.bbbb.bbbb.dddd.dfff.rrrr.rrrr`

where

where `bbbb.bbbb` is an eight 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

So, 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.

So you'll access the 0th configuration register at 1fe.0181.0100 (physical address). On Sun Ultra-30 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, for any system, 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 the `phys.lo`, `phys.mid`, `phys.hi`, and `length` arguments. `Phys.lo`, `phys.mid`, and `phys.hi` numbers can be taken 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.

Here is an example of getting the physical address of the configuration space registers, using onboard ethernet on a Sun Ultra-30 UPA/PCI system:

```
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
```

Hence the physical address for the base of configuration registers is 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, using the "reg" entry for memory or I/O space, you can find a physical address for those spaces.

Controlling PCI Slot Probing on an Ultra-30 UPA/PCI System

You can control probing of PCI slots on your Sun Ultra-30 UPA/PCI system as follows. On Sun Ultra-30 UPA/PCI system, during normal system initialization, there are NVRAM variables which indicate to the CPU PROM what slots to probe and in what order. On the Sun Ultra-30 UPA/PCI system they are: `pcia-probe-list` and `pcib-probe-list`. The default value for `pcia-probe-list` is 1,2; for `pcib-probe-list`, it is 3,2,4,5. To disable slot 4 probing on `pcib`, during normal initialization after a reset, change `pcib-probe-list` to:

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

After a reset, to probe slot 4 on `pcib` manually do:

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

Note that not all CPU PROMs have the `probe-pci-slot` command. Also, in future PROMs, behavior of this command may change, including the possibility of its deletion.

Using 3.x Tokenizer and 3.x CPU PROMs

Here are some points to consider while using the 3.x tokenizer: While you are testing FCode under CPU OpenBoot PROM 3.x versions, make sure that you have OpenBoot PROMs version 3.1 or later. Pre-3.1 PROMs will need the following NVRAM patch:

```
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
```

Also, 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:

```
8000.0000 constant xxx
```

will in reality be giving a value as: `ffff.fff.8000.0000`. 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.

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. 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). Using this or any other method, obtain the values of `bbbb.bbbb`, `dddd` 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

Three types of software involved during a boot: the kernel, FCode, and the OS driver. This section describes the normal Solaris operating environment boot scenario, including the functions of each 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, thus interpreting the FCodes on all FCode PROMs. In the FCode probing process, FCode PROMs generate properties for devices. Some FCode PROMs execute commands to reset the device and perform other initialization.

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. Then, 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.

So, the order is:

- CPU-PROM
- FCode-PROM
- `bootblk`
- Kernel
- OS driver.

Enabling Access to a PCI Device's Memory Space Locations

If a developer is loading FCode and can't access memory space locations, how should they go about enabling access to memory space locations for their PCI device?

Look in the format for the physical address of "reg" property. Using that (or any other method), obtain the values of `bbbb.bbbb`, `dddd` and `fff` for your device. Then use

```
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 needed by your application. If the bus number is 1000.0001 (0x81), the device number is 0.0000, and the function no. 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. Then it can set various bits in the command register to enable the desired access. The `close` routine should disable that access.

Expansion FCode PROM

If a developer is unable to access his expansion FCode PROM, how can access to it be enabled?

To enable access, look in the format for the physical address of the `"reg"` property. Using that (or any other method), obtain the values of `bbbb.bbbb`, `dddd` and `fff` for your device. Then use

```
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. In other words, if the bus number is `1000.0001` (`0x81`), the device number is `00000`, and the function number is `001` (`0x01`), then use:

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

If for any reason, (for example, to access Vital Product Data stored in the PROM) the FCode needs to access PROM data, 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* the value with 1, and write the result back to that address.

Also, since the FCode would have been copied in memory, devices' memory and I/O spaces may not be enabled. So, the FCode must enable them, using 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 example above, 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

In trying to load the FCode from Ethernet, the code seems to load without errors; however, when the developer tries to build the package, she gets an error:

```
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
```

The error may be due to either of these causes:

1. The PCI header is attached to the PROM image, or
2. my-address is 2 32-bit numbers for PCI and only 1 32-bit number for SBUS.

If the cause is 1, dump the download image beginning at 4000, for example

```
4000 60 dump
```

and see where `f1` or `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
```

`f1` or `fd` is the beginning of the FCode header, 8 bytes long, as:

```
f1, <reserved byte>, <2 reserved bytes>, <4 bytes 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` or `fcode-version3`, the first FCode data is `f1`.

If the cause is 2, change your FCode to handle two numbers returned from `my-address`.

One way to do this is:

```
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` in the creation of the "reg" property.

A *package* is the set of methods and properties that resides in a device node. A *support package* is a group of functions or methods that implements a specific interface. A package implements a library of functions that may then be called by FCode programs.

For many devices, this is not particularly useful, but it will be useful for FCode programs that:

- Implement bootable devices
- Call functions or properties from other packages
- Implement functions intended to be called from other packages

A *plug-in package* is a package that is not permanently resident in the main OpenBoot PROM. Plug-in packages are written in FCode. Since FCode is represented with a machine-independent binary format, it lets the same plug-in packages be used on machines with different CPU instruction sets.

During the linking process, a package's references to OpenBoot PROM system functions are resolved and the functions defined by the package made available to other parts of OpenBoot . This occurs at run-time, when OpenBoot interprets (probes) the package. Thus, plug-in packages do not need to be pre-linked with a particular OpenBoot implementation.

OpenBoot only needs the beginning address of the package in order to probe it. Once probed, the package becomes a working part of OpenBoot, until the system is reset or turned off. A package exports its interface to OpenBoot, and to other packages, as a vocabulary of Forth words.

Many packages implement a specific interface; a standard set of functions. Different packages may implement the same interface. For example, there may be two display device driver packages, each implementing the standard display device interface, but for two different display devices.

There may also be multiple instances of a single package. For example, a plug-in disk driver may have as many instances as there are disks of that type.

Package Instances

A package consists of:

- methods (software procedures)
- properties (externally-visible information describing the package), and
- data (information used internally by the package).

The active package is the package whose methods are currently visible. `dev` and `find-device` can be used to change the active package. However, they only make a package's methods visible; they do not enable the execution of those methods.

Before a package's methods may be executed, create an instance of the package. Think of an instance as a working copy of the package. An instance contains a working copy of all of the package's private data.

An instance is created from a package by opening that package. The act of opening a package allocates memory for the instance's data and sets the contents of that memory to the initial values stored in the package. The instance exists until it is terminated by closing it. When it is closed, the memory used to hold that instance's private data is freed. Multiple instances may be created from the same package, and exist simultaneously.

The current instance is the instance whose private data and methods are available for direct use (i.e. directly by name without having to use `$call-method`).

When a package method accesses a data item, it refers to the copy of that data item associated with the current instance. The private data of the current instance is accessible; the private data of all other instances is inaccessible. Furthermore, to use the methods of a package, an instance of that package must be (at least temporarily) the current instance.

A package to be opened is described by a device path or device alias. The process of opening the package includes opening each of the nodes in the device path from the root to the specified device (i.e. from the top of the chain to the bottom). As each of these nodes is opened, an instance is created for the node and all of these instances are linked together in an *instance chain* as shown in Figure 6-1. When a method is accessed using the *ihandle* of the chain, each node in the chain is able to access the methods of its parent with `$call-parent` using the links provided by the instance chain.

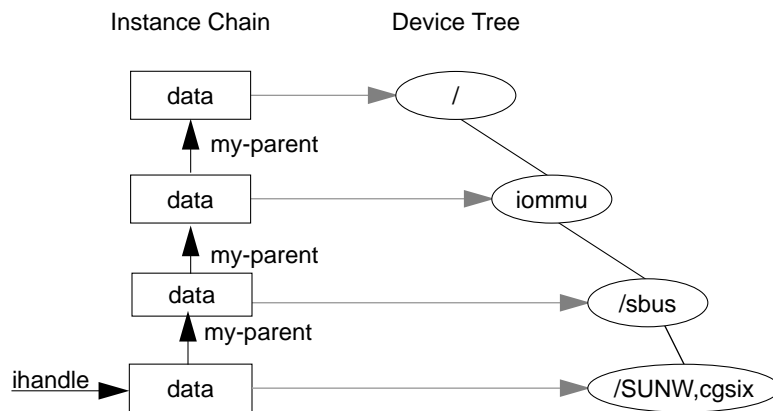


Figure 6-1 An Instance Chain for `/iommu/sbus/SUNW,cgsix`

When the chain is no longer needed, the individual instances of the chain may be closed or the entire chain may be closed. When closing the entire chain, the chain is closed from bottom to top to enable a given node's `close` method to use parental methods.

The current instance is a dynamic entity. It is changed in several different ways under several different circumstances. Specifically:

- When a package is first created, `new-device`:
 - Creates a new device node that is a child of the active package.
 - Makes that new node the active package.
 - Makes that new node's instance the current instance.

This causes any instance data/methods that are subsequently created (prior to the execution of `finish-device`) to be added to this node, and enables their later execution when an instance of this node is made current.

- When `open-dev` creates an instance chain, the current instance is repeatedly changed as each node of the instance chain is added to the instance chain (i.e. the root of the chain is first made current while it is being added to the instance chain, then the first child node is made current while it is added to the chain, and so on down to the leaf node). Immediately before terminating, `open-dev` restores the value in `my-self` to the value that `my-self` contained prior to the execution of `open-dev`. `open-dev` returns the `ihandle` of the leaf node of the newly-created instance chain. By manipulating the current instance in this way, `open-dev` is able to use instance-specific data as required.
- To execute a method not contained in the current instance, `$call-method` (or one of its derivatives) is used. `$call-method`:
 - Saves the current value of `my-self`.
 - Stores its *ihandle* argument in `my-self` (thus changing the current instance).
 - Executes the specified method.
 - Restores the saved value of `my-self`.
- From the user interface, you can change the current instance by setting the value of `my-self` directly. This is most useful in a debugging scenario when testing the methods of an opened package. (The `select-dev` method discussed in Chapter 4, “Debugging and Testing FCode Programs” resets `my-self` for just this purpose.)

If a package is in the node `/packages`, `$open-package` can be used to create an instance of the package. Unlike packages opened with `open-dev`, packages opened with `$open-package` are opened without opening their ancestors. Each time a package instance is created by `$open-package`, that instance is attached to the one that called `$open-package`. Figure 6-2 shows the modified instance chain that results when the `/iommu/sbus/ledma/le` instance opens the `obp-tftp` support package using `$open-package`.

Notice that the only additional instance created is one for the `obp-tftp` package, and that this instance is linked to the `/iommu/sbus/ledma/le` instance. If another instance of `obp-tftp` were opened by an instance in another instance chain, the resulting instance of `obp-tftp` would have no association with the instance shown in Figure 6-2.

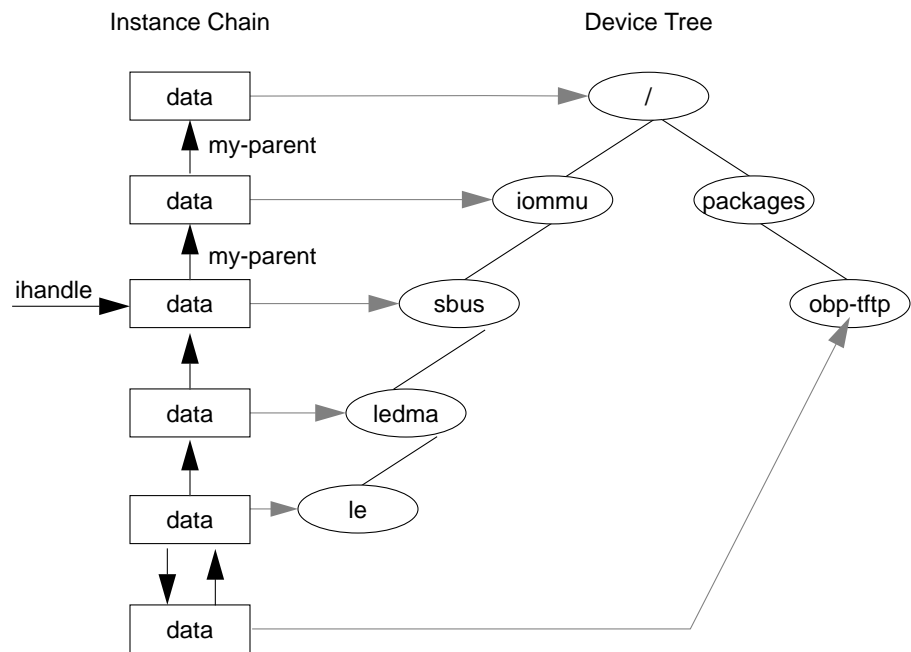


Figure 6-2 An Instance Chain for `/iommu/sbus/ledma/le` with `obp-tftp` Support

Package Data

Package data is named, read/write RAM storage used by package methods. Individual data items can be either initialized or zero-filled and either static or instance-specific.

- Static data can be accessed at any time, regardless of whether or not the package has been opened. There is only one copy of each static data item, regardless of the number of currently-open instances of that package. The process of opening a package does not in itself alter the values of static data items (although you can, of course, write code to do so explicitly).
- Instance-specific data can only be accessed when a previously-opened instance of its package is the current instance. The process of opening a package creates copies of its instance-specific data items and establishes their initial values.
- Zero-filled data items are set to zero when a package is opened.
- Initialized data items are set to possibly-non-zero initial values when a package is opened. The initial values are established during the creation of the package.

Initialized data items are created by the Forth defining words `defer`, `value` and `variable`. Uninitialized data items are created by `buffer:`. Preceding the defining word with the Forth word `instance` causes the defining word to create an instance-specific item; otherwise it creates a static data item.

Static data items are used for information that applies equally to all instances of the associated package. For example, virtual addresses of shared hardware resources, reference counts and hardware dependent configuration data are often stored as static data.

Instance-specific data items are used for information that differs between instances of the same package. For example, a package that provides a driver for a SCSI host adapter might have several simultaneous instances on behalf of several different target devices; each instance might need to maintain individual state information (e.g. the negotiated synchronous transfer rate) for its target.

Static and Instance-specific Methods

There are several different kinds of package methods, depending on the environment in which they are called and their use of static and instance-specific data.

Static methods do not:

- Access instance-specific data either directly or by calling other instance-specific methods.
- Attempt to call methods of their parent.

Static methods can be called when there is no open instance of their package. When there is no instance, there is also no parent instance (which is the reason for the prohibition about calling parent methods).

The most important example of static methods is the `decode-unit` method which is called by the system during the process of searching the device tree without opening all of the nodes that are encountered.

Instance-specific methods are permitted to:

- Use instance-specific data
- Call the methods of their parent.

There is no structural difference between static and instance-specific methods. The concept of static methods is just a terse way of saying that some methods have to obey the restrictions outlined above. Instance-specific methods are the usual case; the static methods restrictions apply only to a very small set of special-purpose methods.

Execution Tokens

A method is identified by its execution token, `xt`. For words in the package being defined, the Forth word `[']` returns an execution token. The execution token is returned by `find-method` for other packages. (See the following sections for more details.)

The execution token is used to execute a method in another package, and also to schedule a method for automatic, repeated execution by the system clock interrupt. See the `alarm` FCode.

Intra-package Calling Methods

A package can call its own methods directly simply by naming the target method in a Forth colon definition. Such calls require neither a call-time name search nor a change of the current instance. The binding of name to execution behavior occurs at compile time, so subsequent redefinitions of a name do not affect previously-compiled references to old versions of that named method.

Infrequently, it may be desirable to call a method in the same package so that the name search happens at run-time. To do so, use either `$call-method` or `find-method/call-package` with `my-self` as the `ihandle` argument. (See the next section for details.)

Accessing Other Packages

Packages often use methods of other previously-defined packages. There are two types of packages whose methods can be used directly:

- The parent of the package being defined.
- Support packages in the `/packages` node of the device tree.

`phandle` **and** `ihandle`

A package definition is identified by its `phandle`. `find-package` returns the `phandle` of a package in the `/packages` node. The `phandle` can then be used to open that support package or to examine its properties. For example:

```
" deblocker" find-package
```

returns either `false` (package not found), or `phandle true`.

Opening a support package with `open-package` returns an `ihandle`. This `ihandle` is used primarily to call the methods of the support package, and to close the support package when it is no longer needed.

The `ihandle` of the current instance is returned by `my-self`. An instance argument string must be supplied when opening any package (it may be null). The instance argument string can then be accessed from in the opened package with the `my-args` FCode (see below for details). For example (assume that `phandle` has already been found):

```
" 5,3,0" phandle open-package ( ihandle )
```

If the package cannot be opened, an `ihandle` of 0 is returned.

`$open-package` includes the functions of `find-package` and `open-package`. In most cases, it can be used in their place. The primitive functions `find-package` and `open-package` are rarely used directly, although `find-package` is sometimes used when it's necessary to examine a support package's properties without opening it.

The following FCode functions are used to find and open packages (in the `/packages` node):

Table 6-1 Package Access FCodes

Name	Stack Diagram	Description
<code>find-package</code>	<code>(name-str name-len -- false phandle true)</code>	Finds the package specified by the string <i>name-str name-len</i> in <code>/packages</code> . Returns the <i>phandle</i> of the package, or <i>false</i> if not found.
<code>open-package</code>	<code>(arg-str arg-len phandle -- ihandle false)</code>	Opens an instance of the package <i>phandle</i> . Returns <i>ihandle</i> for the opened package, or <i>false</i> if unsuccessful. The package is opened with an instance argument string specified by <i>arg-str arg-len</i> .
<code>\$open-package</code>	<code>(arg-str arg-len name-addr name-len -- ihandle false)</code>	Shortcut word to find and open the package named <i>name-str name-len</i> in <code>/packages</code> in one operation. Returns <i>ihandle</i> for the opened package, or <i>false</i> if unsuccessful.

Here is an example of using `$open-package`:

```
" 5,3,0" " deblocker" $open-package ( ihandle | 0 )
```

Table 6-2 Manipulating phandles and ihandles

Name	Stack Diagram	Description
<code>my-self</code>	<code>(-- ihandle)</code>	Return the instance handle of the currently-executing package instance.
<code>my-parent</code>	<code>(-- ihandle)</code>	Return the instance handle of the parent of the currently-executing package instance.
<code>ihandle>phandle</code>	<code>(ihandle -- phandle)</code>	Convert an instance handle to a package handle
<code>close-package</code>	<code>(ihandle --)</code>	Close a package instance.

Don't confuse `phandle` with `ihandle`. Here's how to use them:

1. **Open the package with `$open-package` which returns an `ihandle`.**
2. **Use the `ihandle` to call the methods of the package.**
3. **When done calling the methods of the package, use the `ihandle` to close the instance of the package with `close-package`.**

A package's `phandle` is primarily used to access the package's properties which are never instance-specific. Use `ihandle>phandle` to find the `phandle` of an open package. `my-self` and `my-parent` return `ihandles`, which can be converted into `phandles` with `ihandle>phandle`.

Inter-package Calling Methods

The following functions enable the calling of methods of other packages:

Table 6-3 Functions Enabling Calling Other Packages' Methods

Name	Stack Diagram	Description
<code>\$call-method</code>	<code>(... method-str method-len ihandle -- ???)</code>	Shortcut word that finds and executes the method <i>method-str method-len</i> in the package instance <i>ihandle</i> .
<code>call-package</code>	<code>(... xt ihandle -- ???)</code>	Executes the method <i>xt</i> in the instance <i>ihandle</i> .

Table 6-3 Functions Enabling Calling Other Packages' Methods

Name	Stack Diagram	Description
<code>\$call-parent</code>	<code>(... method-str method-len -- ???)</code>	Executes the method <i>method-str method-len</i> in the parent's package instance. Identical to calling <code>my-parent \$call-method</code> .
<code>execute-device-method</code>	<code>(... dev-str dev-len method-str method-len -- ... false ??? true)</code>	Executes the method <i>method-str method-len</i> in the package named <i>dev-str dev-len</i> . Returns <i>false</i> if the method could not be executed.
<code>find-method</code>	<code>(method-str method-len phandle -- false xt true)</code>	Finds the method named <i>method-str method-len</i> in the package <i>phandle</i> . Returns <i>false</i> if not found.

`$call-parent` is used most-often, but is the least flexible of the above methods; it is exactly equivalent to the sequence "my-parent \$call-method". Most inter-package method calling involves calling the methods of one's parent; `$call-parent` conveniently encapsulates the process of doing so.

`$call-method` can call methods of non-parent packages. It is most commonly used for calling methods of support packages. The *ihandle* argument of `$call-method` identifies the package instance whose method is to be called.

For example:

```
$call-parent
$open-package $call-method
```

Both `$call-parent` and `$call-method` identify their target method by name. The *method-str method-len* arguments denote a text string that `$call-parent` or `$call-method` uses to search for a method of the same name in the target instance's list of methods. Obviously, this run-time name search is not as fast as directly executing a method whose address is already known. However:

1. Most packages have a relatively small number of methods,
2. Systems typically implement a reasonably-efficient name search mechanism, and
3. Inter-package calls tend to occur relatively infrequently.

Consequently, the length of time spent searching is usually not a limiting factor.

A more complete example demonstrates the use of `$open-package` and `$call-method`:

```

: add-offset ( x.byte# -- x.byte#' )
  my-args " disk-label" $open-package( ihandle )
  " offset" rot          ( name-addr name-len ihandle )
  $call-method
;

```

When method name search time is a limiting factor, use `find-method` to perform the name search once. Then use `call-package` repetitively thereafter. `find-method` returns, and `call-package` expects, an *execution token* by which a method can be called quickly.

A more complex example that is somewhat faster if called repeatedly:

```

0 value label-ihandle      \ place to save the other package's ihandle
0 value offset-method      \ place to save found method's xt

: init ( -- )
  my-args " disk-label" $open-package ( ihandle ) to label-ihandle
  " offset" label-ihandle ihandle>phandle ( name-addr name-len phandle)
  find-method if
    ( xt ) to offset-method
  else ." Error: can't find method"
  then
;
: add-offset ( d.byte# -- d.byte#' )
  offset-method label-ihandle call-package
;

```

Because device access time often dominates I/O operations, the benefit of this extra code probably won't be noticed. It is only justified if the particular method will be called often.

Another use of `find-method` is to determine whether or not a package has a method with a particular name. This allows you to add new methods to an existing package interface definition without requiring version numbers to denote which new or optional methods a package implements.

With `$call-method` and `$call-parent`, the method name search is performed on every call. Consequently, if a new method (either one with a new name or with the same name as a previously-existing name) is created, any subsequent uses of `$call-method` or `$call-parent` naming that method will find the new one. On the other hand, `find-method` binds a name to an execution token and subsequent redefinitions of that name do not affect the previous execution token, so subsequent uses of `$call-method` continue to call the previous definition. In practice, this difference is rarely important, since it is quite unusual for new methods to be created when a package is already open. The one case where methods are routinely redefined under these circumstances is when a programmer does it explicitly during a debugging session; making such redefinitions is a powerful debugging technique.

All of the method calling functions described above change the current instance to the instance of the callee for the duration of the call, restoring it to the instance of the caller on return.

`execute-device-method` *and* `apply`

In addition to the inter- and intra-package method calling techniques just described, there is another way of calling methods.

`execute-device-method` and its variant `apply` allow a user to invoke a method of a particular package as a self-contained operation without explicitly opening and closing the package as separate operations.

`execute-device-method` first opens all the package's parents, then calls the named method, and then closes all the parents. `apply` performs the same functions as `execute-device-method`, but it takes its arguments from the command line instead of from the Forth stack.

`execute-device-method` and `apply` are most often used for methods like `selftest`. `selftest` methods are usually called with the test user interface command, which is usually implemented with `execute-device-method`.

Methods that are intended to be called with `execute-device-method` or its equivalent must not assume that the package's `open` method has been called, because `execute-device-method` does not call the `open` method of the

package containing the target method although it opens all of the package's parents. Consequently, the target method must explicitly perform whatever initialization actions it requires, perhaps by calling the `open` method in the same package, or by executing some sub-sequence thereof. Before exiting, the target method must perform the corresponding `close` actions to undo its initialization actions.

`execute-device-method` was intentionally designed *not* to call the target's `open` and `close` methods automatically since the complete initialization sequence of `open` is not always appropriate for methods intended for use with `execute-device-method`. In particular, an `open` method usually puts its device in a fully operational state, while methods like `selftest` often need to perform a partial initialization of selected device functions.

Plug-in Device Drivers

Plug-in device drivers are plug-in packages implementing simple device drivers. The interfaces to these drivers are designed to provide basic I/O capability.

Plug-in drivers are used for such functions as booting the operating system from a device or displaying text on a device before the operating system has activated its own drivers. Plug-in drivers are added to the device tree during the probing phase of the OpenBoot PROM start-up sequence.

Plug-in drivers must be programmed to handle portability issues, such as hardware alignment restrictions and byte ordering of external devices. With care, you can write a driver so that it is portable to all of the systems in which the device could be used.

Plug-in drivers are usually stored in PROM located on the device itself, so that the act of installing the device automatically makes its plug-in driver available to the OpenBoot PROM.

For devices with no provision for such a plug-in driver PROM, the plug-in driver can be located elsewhere, perhaps in PROM located on a different device or in an otherwise unused portion of the main OpenBoot PROM. However, use of such a strategy limits such a device to certain systems and/or system configurations.

Common Package Methods

Different packages have different collections of methods depending on the job(s) that the packages have to do. The following four methods are found in many device drivers. None of them can be considered to be required, however, since the nature of a given driver governs the methods that the driver needs.

`open` and `close` are found in many drivers, but even they are not universally required. `open` and `close` are needed only if the device will be used with `open-dev` or another method that calls `open-dev`. Any device that has `read` and/or `write` methods needs `open` and `close`, as does any parent device whose children could possibly be opened.

Another way of looking at this is that `open` and `close` are needed for devices that are used to perform a series of related operations distributed over a period of time, relative to some other calling package. `open` initializes the device state that is maintained during the series of later operations, and `close` destroys that state after the series is complete.

To illustrate, a series of `write` calls generated by another package is such a series. Conversely, `selftest` is not such a series; `selftest` happens “atomically” as an indivisible self-contained operation.

Basic Methods

open

```
( -- ok? )
```

Prepares a package for subsequent use. `open` typically allocates resources, maps, initializes devices, and performs a brief sanity check (making no check at all may be acceptable). `true` is returned if successful, `false` if not. When `open` is called, the parent instance chain has already been opened, so this method may call its parent’s methods.

close

```
( -- )
```

Restores a package to its “not in use” state. `close` typically turns off devices, unmaps, and de-allocates resources. `close` is executed before the package’s parent is closed, so the parent’s methods are available to `close`. It is an error to close a package which is not open.

Recommended Methods

The following methods are highly recommended.

reset

```
( -- )
```

Put the package into a “quiet” state. `reset` is primarily for packages that do not automatically assume a quiet state after a hardware reset, such as devices that turn on with interrupt requests asserted.

selftest

```
( -- error# )
```

Test the package. `selftest` is invoked by the OpenBoot `test` word. It returns 0 if no error is found or a package-specific error number if a failure is noticed.

`test` does not open the package before executing `selftest`, so `selftest` is responsible for establishing any state necessary to perform its function prior to starting the tests, and for releasing any resources allocated after completing the tests. There should be no user interaction with `selftest`, as the word may be called from a program with no user present.

If the device was already open when `selftest` is called, a new instance will still be created and destroyed. A well-written `selftest` should handle this possibility correctly, if appropriate.

If the device is already open, but it is not possible to perform a complete `selftest` without destroying the state of the device, the integrity of the open device should take precedence, and the `selftest` process should test only those aspects of the device that can be tested without destroying device state. The inability to fully test the device should not be reported as an error result; an error result should occur only if `selftest` actually finds a device fault.

The "device already open" case happens most commonly for display devices, which are often used as the console output device, and thus remain open for long periods of time. When testing a display device that is already open, it is not necessary to preserve text that may already be on the screen, but the device state should be preserved to the extent that further text output can occur and be visible after `selftest` exits. Any error messages that are displayed by the `selftest` method will be sent to the console output device, so when testing an already-open display device, such error messages should be avoided during times when `selftest` has the device in a state where it is unable to display text.

`selftest` is *not* executed in an open/close pair. When `selftest` executes, a new instance is created (and destroyed). It will have its own set of variables, values, and so forth. These quantities are not normally shared with an instance opened with the normal `open` routine for the package.

Note - `selftest` should be written to do its own mapping and unmapping.

Package Data Definitions

The following examples show how to create static data items:

```
variable bar
5 value grinch
defer stub
create ival x , y , z ,
7 buffer: foo
ival foo 7 move           \ One way to initialize a buffer
```

The data areas defined above are shared among all open instances of the package. If a value is changed, for instance, the new value will persist until it is changed again, independent of the creation and destruction of package instances.

Any open instance of a package can access and change the value of a static data item, which changes it for all other instances.

The following examples show how to create instance-specific data items, whose values are not shared among open instances:

```
instance variable bar
5 instance value grinch
instance defer stub
7 instance buffer: foo
```

Instance-specific data areas are re-initialized when a package instance is created (usually by opening the package), so each instance gets its own copy of the data area. For example, changes to *bar* in one instance will not affect the contents of it in another instance. (Note that `create` operates across all the instances, and cannot be made instance-specific.)

The total amount of data space needed for a package's instance-specific data items is remembered as part of the package definition when `finish-device` finishes the package definition. Also, the contents of all the `variables`, `values`, and `defers` at the time `finish-device` executes are stored as part of the package definition.

An instance of the package is created when that package is opened. Data space is allocated for that instance (the amount of which was remembered in the package definition). The portion of that data space created with `variable`, `value`, or `defer` is initialized from the values stored in the package definition. Data space created with `buffer:` is set to zero.

You can add new methods and new properties to a package definition at any time, even after `finish-device` has been executed for that package. To do so, select the package and create definitions or properties.

However, you cannot add new data items to a package definition after `finish-device` has been executed for that package. `finish-device` sets the size of the data space for that package, and subsequently the size is fixed.

Instance Arguments and Parameters

An instance argument (`my-args`) is a string that is passed to a package when it is opened. The string may contain parameters of any sort, based on the requirements of the package, or may simply be a null-string if no parameters are needed. A null string can be generated with either " " or 0 0.

The instance argument passed can be accessed from inside the package with the `my-args` FCode.

Note – A package is not required to inspect the passed arguments.

If the argument string contains several parameters separated by delimiter characters, you can extract the subsections from the package with `left-parse-string`. You can use any character as the delimiter; a comma is commonly used for this.

Note – Avoid using blanks or the `/` character, since these will confuse the parsing of pathnames.

A new value for `my-args` is passed when a package is opened. This can happen under a number of circumstances:

1. The `my-args` string will generally be null when FCode on a Plug-in card is interpreted automatically by the OpenBoot system at power-on.
2. The `my-args` string is set by a parameter to `begin-package`, which is used to set up the device tree when Forth source code is downloaded and interpreted interactively.
3. The `my-args` string can be set with `set-args` before a particular slot is probed, if probing is being controlled from `nvrामrc`.

The above three instances happen only once, when the package FCode is interpreted for the first time. If you want to preserve the initial value for `my-args`, the FCode program should copy it into a static buffer to preserve the information.

Whenever a package is re-opened, a new value for `my-args` is supplied. The method for supplying this new value depends on the method used to open the package, as described below.

- The instance argument (`my-args`) is supplied as a string parameter to the commands `open-package` or `$open-package`.

- User Interface commands, such as `open-dev`, `execute-device-method` and `test`, supply the entire pathname to the device being opened. This approach lets an instance argument be included in the pathname. For example, to open the SBus device `SUNW,bwtwo` with the argument string `5,3,0`, enter:

```
ok " /sbus/SUNW,bwtwo:5,3,0" open-dev
ok
```

Here is a more complicated (and fictitious) example:

```
ok " /sbus/SUNW,fremly:test/grumpin@7,32:print/SUNW,fht:1034,5"
ok open-dev
ok
```

Here the string `test` is passed to the `SUNW,fremly` package as it is opened, the string `print` is passed to the `grumpin` package as it is opened, and the string `1034,5` is passed to the `SUNW,fht` package as it is opened.

Package Addresses

A package's address relative to its parent package is another piece of information available to a package. Again, there are two main ways to pass this address to the package:

- Part of the pathname of the package
- A string parameter given to the probe words

As an example of the first method, suppose the following package is being opened:

```
ok " /sbus/esp/sd@3,0:b" open-dev
```

Then the address of the `/sd` package relative to the `/esp` package is `3,0`.

The package can find its relative address with `my-unit`, which returns the address as a pair of numbers. The first number (*high*) is the number before the comma in the example above, and the second number (*low*) is the number after the comma. Note that these are numbers, not strings.

As an example of the second method, suppose a test version of an FCode package is being interpreted:

```
ok 0 0 " 3,0" " /sbus" begin-package
```

Here the `my-args` parameters for the new FCode are null, the initial address is `3,0` and it will be placed under the `/sbus` node.

The initial address can be obtained through `my-address` and `my-space`. Typically, you use `my-space` and `my-address` (plus an offset) to create the package's "reg" property, and also to map in needed regions of the device.

Package Mappings

Mappings set up by a package persist across instances unless they are explicitly unmapped. It is usually best for each new instance to do its own mappings, being sure to unmap resources as they are no longer needed.

nvrामrc

Machines that support packages will generally also support the `nvrामrc` facility. `nvrामrc` is a special area in the NVRAM that can contain user interface commands to be executed by OpenBoot as the machine powers on. These commands can be used to specify behavior during start up or to define changes for later execution.

For example: assume a card in SBus slot #2 (named `XYZ,me`) needs custom attributes set by the user. `nvrामrc` contents would include:

```
probe-all
dev /sbus/XYZ,me
" type5" encode-string " xyzmode" property
device-end
install-console
banner
```

After editing `nvrामrc`, turn on the NVRAM parameter `use-nvrामrc?` and reset the machine to activate the contents of `nvrामrc`. See `nvedit` in Chapter 14, “FCODE Dictionary” for more about editing `nvrामrc` contents.

Modifying Package Properties

To modify the properties of a package, first probe the package to get it into memory, then create or modify properties by executing `property` or one of its short-hand forms. Normally, probing is done automatically after the `nvrामrc` commands are executed.

See Chapter 7, “Properties“, for more information about properties.

Standard Support Packages

The `/packages` node of the device tree is unique. It has children, but instead of describing a physical bus, `/packages` serves as a parent node for support packages. The children of `/packages` are general-purpose software packages not attached to any particular hardware device. The “physical address space” defined by `/packages` is a trivial one: there are no addresses. Its children are distinguished by name alone.

The children of `/packages` are used by other packages to perform commonly used functions. They may be opened with the FCODEs `open-package` or `$open-package`, and closed with `close-package`. *IEEE Standard 1275-1994* defines three support packages that are children of `/packages`.

Sun Disk-Label Support Package

Disk (block) devices are random-access, block-oriented storage devices with fixed-length blocks. Disks may be subdivided into several logical partitions, as defined by a *disk label*—a special disk block, usually the first one, containing information about the disk. The disk driver is responsible for appropriately interpreting a disk label. The driver may use the standard support package `/disk-label` if it does not implement a specialized label.

`/disk-label` interprets a standard Sun disk label, reading any partitioning information contained in it. It includes a first-stage disk boot protocol for the standard label. `load` is the most important method defined by this package.

This package uses the `read` and `seek` methods of its parent (in practice, the package which opens this one to use the support routines). `/disk-label` defines the following methods:

Table 6-4 Sun Disk Label Package Methods

Name	Stack Diagram	Description
open	(-- flag)	Reads and verifies the disk label accessed by the <code>read</code> and <code>seek</code> methods of its parent instance. Selects a disk partition based on the text string returned by <code>my-args</code> . For the standard Sun disk label format, the argument is interpreted as follows:
.	.	Argument
.	.	Partition
.	.	<none>
.	.	0
.	.	a or A
.	.	0
.	.	b or B
.	.	1
.
.
.	.	h or H
.	.	7
.	.	Returns -1 if the operation succeeds. As a special case, if the argument is the string "nolabel", <code>open</code> returns -1 (success) without attempting to read or verify the label.
close	(--)	Frees all resources that were allocated by <code>open</code> .
load	(adr -- size)	Reads a stand-alone program from the standard disk boot block location for the partition specified when the package was opened. Puts the program at memory address <code>adr</code> , returning its length <code>size</code> . For the standard Sun disk format, the stand-alone program is 7.5K bytes beginning 512 bytes from the start of the partition.
offset	(x.rel-- x.abs)	Returns the 64-bit absolute byte offset <code>x.abs</code> corresponding to the 64-bit partition-relative byte offset <code>x.rel</code> . In other words, adds the byte location of the beginning of the selected partition to the number on the stack.

TFTP Booting Support Package

The `/obp-tftp` package implements the Internet Trivial File Transfer Protocol (TFTP) for use in network booting. It is typically used by a `network` device driver for its first stage network boot protocol. Again, `load` is the most important method defined by this package.

This package uses the `read` and `write` methods of its parent, and defines the following methods:

Table 6-5 TFTP Package Methods

Name	Stack Diagram	Description
<code>open</code>	(-- flag)	Prepares the package for subsequent use, returning -1 if the operation succeeds and 0 otherwise.
<code>close</code>	(--)	Frees all resources that were allocated by <code>open</code> .
<code>load</code>	(<code>adr</code> -- <code>size</code>)	Reads the default stand-alone program from the default TFTP server, putting the program at memory address <code>adr</code> and returning its length <code>size</code> . For the standard Sun TFTP booting protocol, RARP (Reverse Address Resolution Protocol) is used to acquire the IP address corresponding to the system's MAC address (equivalent to its Ethernet address). From the IP address, the default file name is constructed, of the form <code><Hex-IP-Address>.<architecture></code> (for example, <code>C0092E49.SUN4C</code>). Then <code>obp-tftp</code> tries to TFTP read that file, first trying the server that responded to the RARP request, and if that fails, then broadcasting the TFTP read request.

Deblocker Support Package

The `/deblocker` package makes it easy to implement byte-oriented device methods, using the block-oriented or record-oriented methods defined by devices such as disks or tapes. It provides a layer of buffering between the high-level byte-oriented interface and the low-level block-oriented interface. `/deblocker` uses the `max-transfer`, `block-size`, `read-blocks` and `write-blocks` methods of its parent, and defines the following methods:

Table 6-6 Deblocker Package Methods

Name	Stack Diagram	Description
<code>open</code>	(-- flag)	Prepares the package for subsequent use, allocating the buffers used by the deblocking process based on the values returned by the parent instance's <code>max-transfer</code> and <code>block-size</code> methods. Returns -1 if the operation succeeds, 0 otherwise.
<code>close</code>	(--)	Frees all resources that were allocated by <code>open</code> .

≡ 6

Table 6-6 Deblocker Package Methods

Name	Stack Diagram	Description
read	(<i>adr len -- actual</i>)	Reads at most <i>len</i> bytes from the device into the memory buffer beginning at <i>adr</i> . Returns <i>actual</i> , the number of bytes actually read, or 0 if the read operation failed. Uses the parent's <i>read-blocks</i> method as necessary to satisfy the request, buffering any unused bytes for the next request.
write	(<i>adr len -- actual</i>)	Writes at most <i>len</i> bytes from the device into the memory buffer beginning at <i>adr</i> . Returns <i>actual</i> , the number of bytes actually read, or 0 if the write operation failed. Uses the parent's <i>write-blocks</i> method as necessary to satisfy the request, buffering any unused bytes for the next request.
seek	(<i>x.position -- flag</i>)	Sets the device position at which the next <i>read</i> or <i>write</i> will take place. The position is specified by the 64-bit number <i>x.position</i> . Returns 0 if the operation succeeds or -1 if it fails.

This chapter describes characteristics of hardware devices, software and user selections. Properties are associated with the device node in which they are created and are accessible both by OpenBoot routines and by client programs. Properties can be inspected and, in some cases, modified.

Each property has a *property name* and a *property value*.

- Property names are human-readable strings consisting of one to 31 printable, lower-case letters and symbols not including blanks, “/”, “\”, “:”, “[”, “]” or “@”. Property names beginning with “+” are reserved for future use by *IEEE Standard 1275-1994*.
- Property values specify the contents, or value, of a particular property. The value is an array of bytes that may be used to encode integer numbers, text strings, or other forms of information.

Properties are accessed by name. Given a property’s name, it is possible to determine whether that property has been defined and, if so, what its value is.

Property values are encoded as arrays of zero or more bytes for portability across machine architectures. The encoding and decoding procedures are defined by *IEEE Standard 1275-1994*. The encoding format is independent of hardware byte order and alignment characteristics. The encoded byte order is big-endian and the bytes are stored in successive memory locations without any padding.

The format of the property value array associated with a given property name is specific to that property name. There are five basic types of property value array formats:

- Flag

Since property value arrays may be of zero length, properties may convey “true” or “false” information by their presence or absence.

- Byte

An array of 1 or more bytes is stored in a property value array as a series of sequential bytes in the property value array.

- 32-bit integer

A 32-bit integer is stored in a property value array in four successive bytes. The most significant byte of the integer in the next available address in the property value array is followed by the high middle, low middle and least significant bytes of the integer (in other words, in big-endian format).

- Text string

A text string of n printable characters is stored in a property value array in $n+1$ successive locations by storing the string in the first n locations followed by a byte of zero value (in other words, a null terminated string).

- Composite

A composite value is made up of the concatenation of encoded bytes, encoded 32-bit integers and/or encoded strings. Each such primitive is stored immediately after the preceding primitive with no intervening space (in other words, the items are packed). Here are examples of composite values:

- a. *physical address range*. Encoded as 4 integers: *phys.lo phys.mid phys.hi size*
- b. *array*. The concatenation of n items of some type.

The standard defines a number of standard properties with specified names and value formats. If a package uses one of these standard properties then the value format of the property must be as defined by the standard. Packages may define other properties whose names do not conflict with the list of standard properties. Such newly defined properties may have any value format.

Properties may be created by FCode programs. The CPU's OpenBoot is able to use property names that tell it such things as the device type (e.g. disk, tape, network, display, and so on) to determine how to use the device (if at all) during the boot process.

Solaris recognizes other property names that give information for configuring the operating system automatically. These properties include the driver name, the addresses and sizes of the device's registers, and interrupt levels and interrupt vectors used by the device.

Other properties may be used by individual operating system device drivers. The names of such properties and the interpretation of their values is subject to agreement between the writers of the FCode programs and the operating system driver, but may otherwise be arbitrarily chosen. For example, a display device might declare width, height, and depth properties to allow a single operating system driver to automatically configure itself for one of several similar but different devices.

A package's properties identify the characteristics of the package and its associated physical device, if any. You can create a property either with the `property` FCode, or with the `name`, `reg`, `model`, and `device-type` FCodes, described below.

For example, a frame buffer package might export its register addresses, interrupt levels, and frame buffer size. Every package has an associated property list, which is arbitrarily extensible. The user interface command `.properties` displays the names and values of the current node's properties.

For example, if a property named `foo` is created in a device node which already has a property named `foo`, the new property supersedes the old one.

New properties can be added during the lifetime of a product. For backward compatibility, an FCode or device driver program that needs the value of a particular property should determine whether or not the property exists, and if not the program should supply its own default value.

Standard FCode Properties

IEEE Standard 1275-1994 defines the following standard properties. A package should never create any property using any of the following names, unless the defined meanings and structures are used.

Standard Property Names

This group of properties applies to all device nodes regardless of type. The name property is required in all packages. The remaining properties are optional.

name	Name of the package.
reg	Package's registers.
device_type	Characteristics that the device is expected to have.
model	Manufacturer's model number.
interrupts	Interrupts used by the device.
address	Virtual addresses of one or more memory-mapped regions of the device.
compatible	List of devices with which this device is compatible.
status	Operational status of the device.

Display Device Properties

Display devices include bit-mapped frame buffers, graphics displays and character-mapped displays. Display devices are typically used for console output. The following properties are specific to display devices:

character-set	Character set (e.g. ISO8859-1).
depth	Number of bits in each pixel of the display.
height	Number of pixels in the "y" direction of the display.
linebytes	Number of pixels between consecutive scan lines of the display.
width	Number of pixels in the "x" direction of the display.
big-endian-aperture	The big-endian aperture of the frame buffer.
little-endian-aperture	The little-endian aperture of the frame buffer.

Network Device Properties

Network devices are packet-oriented devices capable of sending and receiving Ethernet packets. Network devices are typically used for booting.

<code>mac-address</code>	Last used network address.
<code>address-bits</code>	Number of address bits needed to address this device on the physical layer.
<code>max-frame-size</code>	Maximum packet size that the device can transmit at one time.

Memory Device Properties

Memory devices are traditional random-access memory, suitable for temporary storage of data.

<code>reg</code>	Physical addresses installed in the system.
<code>available</code>	Regions of physical addresses that are currently unallocated by OpenBoot.

General Properties For Parent Nodes

<code>#address-cells</code>	Device node's address format.
<code>#size-cells</code>	Number of cells that are used to encode the size field of a child's <code>reg</code> property.
<code>ranges</code>	Relationship between the physical address spaces of the parent and child nodes.

Properties For PCI Parent Nodes

#address-cells	The value of this property for a PCI bus node is 3.
#size-cells	The value of this property for a PCI bus node is 2, reflecting the 64-bit address space of PCI.
device_type	The value of this property for a PCI bus node is "pci".
reg	For nodes representing PCI-to-PCI bridges, the value denotes the configuration space address of the bridges's configuration registers. The format is the same as that for PCI child nodes. For nodes representing bridges from some other bus to PCI, the format is as defined for the other bus.
bus-range	Specifies the range of bus numbers controlled by this PCI bus.
slot-names	Describes the external labeling of add-in slots.

Properties for PCI Child Nodes

The following definitions are specified by the PCI Bus Binding to IEEE Standard 1275-1994.

reg	This standard property is mandatory for PCI Child nodes.
interrupts	The presence of this property indicates that the function represented by this node is connected to a PCI expansion connector's interrupt line.
alternate-reg	Defines alternate access paths for addressable regions.
has-fcode	The presence of this property indicates that this node was created by the evaluation of an FCode program.
assigned-addresses	Defines the configuration space's base address and size.
power-consumption	Describes the device's maximum power consumption categorized by the various power rails and the device's power-management state.

Each of the following PCI child node properties is created during the probing process, after the device node has been created, and before evaluating the device's FCode (if any). The property values are those found in the standard PCI configuration registers.

Unless otherwise specified, each of the following properties has a property value created by encoding the value contained in the associated hardware register with `encode-int`.

- "vendor-id"
- "device-id"
- "revision-id"
- "class-code"
- "interrupts"

This property is present only if the interrupt pin register is non-zero.

- "min-grant"
- "max-latency"
- "devsel-speed"
- "fast-back-to-back"

This property is present only if the fast-back-to-back bit (Bit 7) of the function's status register is set.

Detailed Descriptions of Standard Properties

#address-cells

Applies only to bus nodes. It specifies the number of cells that are used to represent a physical address with a bus' address space.

The value for SBus nodes is 2, for PCI bus nodes is 3.

#size-cells

Applies only to bus nodes. It specifies the number of cells used to represent the length of a physical address range (in other words, the *size* field of a child's `reg` property).

The value for SBus nodes is 1, for PCI bus nodes is 2.

address

Declares currently-mapped device virtual addresses. It is generally used to declare large regions of existing mappings in order to enable the operating system device driver to re-use those mappings, thus conserving system resources. This property should be created after virtual addresses have been assigned by mapping operations. Should be deleted when the corresponding virtual addresses are unmapped.

The property value is an arbitrary number of virtual addresses. The correspondence between declared addresses and the set of mappable regions of a particular device is device-dependent.

```
-l value my-buffers
-l value my-dma-addr
: map-me ( -- )
  my-address my-space 1.0000 " map-in" $call-parent ( virt1 )
  to my-buffers
  2000 " dma-alloc" $call-parent ( virt2 ) to my-dma-addr
  my-buffers encode-int my-dma-addr encode-int encode+
  " address" property
;
: unmap-me ( -- )
  my-dma-addr 2000 " dma-free" $call-parent
  my-buffers 1.0000 " map-out" $call-parent
  " address" delete-property
;
```

See also: `free-virtual`, `property`.

address-bits

When declared in network devices, indicates the number of address bits needed to address this device on its network. Used as:

```
d# 48 encode-int " address-bits" property
```

See also: `property` and Chapter 11, “Network Devices”.

alternate-reg

This property describes alternative access paths for the addressable regions described by the "reg" property. Typically, an alternative access path exists when a particular part of a device can be accessed either in memory space or in I/O space, with a separate base address register for each of the two access paths. The primary access paths are described by the "reg" property and the secondary access paths, if any, are described by the `alternate-reg` property.

If no alternative paths exist, the `alternate-reg` property should not be defined. If the device has alternative access paths, each entry (in other words, each *phys-addr size* pair) of its value represents the secondary access path for the addressable region whose primary access path is in the corresponding entry of the "reg" property value. If the number of `alternate-reg` entries exceeds the number of "reg" property entries, the additional entries denote addressable regions that are not represented by "reg" property entries, and are thus not intended to be used in normal operation; such regions might, for example, be used for diagnostic functions. If the number of `alternate-reg` entries is less than the number of "reg" entries, the regions described by the extra "reg" entries do not have alternative access paths. An `alternate-reg` entry whose *phys.hi* component is zero indicates that the corresponding region does not have an alternative access path; such an entry can be used as a placeholder to preserve the positions of later entries relative to the corresponding "reg" entries. The first `alternate-reg` entry, corresponding to the "reg" entry describing the function's configuration space registers, has a *phys.hi* component of zero.

The property value is an arbitrary number of (*phys-addr, size*) pairs where:

- *phys-addr* is (*phys.lo phys.mid phys.hi*), encoded with `encode-phys`.
- *size* is a pair of integers, each encoded with `encode-int`. The first integer denotes the most-significant 32 bits of the 64-bit region size and the second integer denotes the least-significant 32 bits thereof.

assigned-addresses

This property describes the location and size of regions of physical address space that are specified in the device's configuration space base address registers.

The property value is zero to six (*phys-addr, size*) pairs where:

- *phys-addr* is (*phys.lo phys.mid phys.hi*), encoded with `encode-phys`.
- *size* is a pair of integers, each encoded with `encode-int`. The first integer denotes the most-significant 32 bits of the 64-bit region size and the second integer denotes the least-significant 32 bits thereof.

Each entry (in other words, (*phys-addr*; *size*) pair) in this property value corresponds to one (or two in the case of 64-bit-address memory space) of the function's configuration space base address registers. The entry indicates the physical address that has been assigned to that base address register, and the size in bytes of the assigned region. The size is a power of two (since the structure of PCI base address registers forces the decoding granularity to powers of two). Please see the glossary entry for this property for a complete description of the formatting details.

Note – There is no implied correspondence between the order of entries in the "reg" property value and order of entries in the `assigned-addresses` property value. The correspondence between the "reg" entries and `assigned-addresses` entries is determined by matching the fields denoting the base address register.

available

Defines the resources that are managed by this package (in other words, `/memory` or `/mmu`) that are currently available for use by a client program.

The property value is an arbitrary number of (*phys-addr*; *length*) pairs where: *phys-addr* is a *phys.lo phys.mid phys.hi* list of integers encoded with `encode-int`.

- *length* (whose format depends on the package) is one or more integers, each encoded with `encode-int`.

big-endian-aperture

This property is associated with display devices. Encoded identically to "reg" for the corresponding bus, the property value contains the address of the big-endian aperture of the frame buffer (in other words, the address range through which the frame buffer can be addressed in big-endian mode).

bus-range

This property specifies the range of bus numbers controlled by this PCI bus. The property value is two integers, each encoded with `encode-int`. The first integer represents the bus number of the PCI bus implemented by the bus controller represented by this node. The second integer represents the largest bus number of any PCI bus in the portion of the PCI domain that is subordinate to this node.

big-endian-aperture

Associated with `display` devices. Encoded identically to `reg` for the corresponding bus, the property value contains the address of the big endian “aperture” of the frame buffer (i.e. the address range through which the frame buffer can be addressed in big-endian mode).

character-set

When declared in `display` or `serial` devices, indicates the recognized character set for this device. The property value is a text string.

A typical value is “ISO8859-1”. 8859-1 is the number of the ISO specification for that particular character set, which essentially covers the full range of western European languages. To get a list of possible values, consult the X registry for which there is an address in the X11R5 documentation.

Used as:

```
" ISO8859-1" encode-string " character-set" property
```

See also: `property`, Chapter 9, “Display Devices” and Chapter 12, “Serial Devices”.

class-code

This property contains the value of the class code register from the configuration space header. This register identifies the generic function of the device and (in some cases) a specific register-level programming interface.

The property value is the register's value encoded with `encode-int`.

See also: PCI Local Bus Specification.

compatible

This property specifies a list of devices with which this device is compatible. The property is typically used by client programs to determine the correct driver to use with the device in those cases where the client program does not have a driver which matches the "name" property.

The property value is the concatenation (with `encode+`) of an arbitrary number of text strings (encoded with `encode-string`) wherein each text string follows the formatting conventions as described for the "name" property.

Note – Recommended practice document on the topic “Generic Names” is available on the Open Firmware Working Group’s homepage. Recommended practice documents can be obtained as described in “Related Books” in the Preface.

See also: `name`.

class-code

Contains the value of the “Class Code” register from the Configuration Space header. That register identifies the generic function of the device and (in some cases) a specific register-level programming interface.

The property value is the register’s value encoded with `encode-int`.

See also: *PCI Local Bus Specification*

depth

Associated with `display` devices. Encoded with `encode-int`, the property value specifies the number of bits in each pixel of the display.

device-id

This property contains the value of the device ID register from the configuration space header. That register identifies the particular device. The encoding of the register is determined by the device vendor.

The property value is the register's value encoded with `encode-int`.

See also: *PCI Local Bus Specification*.

device-id

Contains the value of the "Device ID" register from the Configuration Space header. That register identifies the particular device. The encoding of the register is determined by the device vendor.

The property value is the register's value encoded with `encode-int`.

See also: *PCI Local Bus Specification*

device_type

Declares the type of this plug-in device. The type need not be declared, unless this device is intended to be used for booting. If this property is declared, using one of the following key values, the FCode program *must* follow the required conventions for that particular type of device, by implementing a specified set of properties and procedures (methods). Used as:

```
" display" encode-string " device_type" property
```

Defined values for this property are:

Table 7-1 Standard Device Types

Device Type	Device Characteristics
block	Random-access, block-oriented device, such as a disk drive, usable as a boot file source. See Chapter 8, “Block and Byte Devices” for the requirements of this type of device.
byte	Random-access, byte-oriented device, such as a tape drive, usable as a boot file source. See Chapter 8, “Block and Byte Devices” for the requirements of this type of device.
display	Frame buffer or other similar display device, usable for message display during booting. See Chapter 9, “Display Devices” for the requirements of this type of device
memory	Random-access memory device. See <i>IEEE Standard 1275-1994</i> for the requirements of this type of device.
network	Packet-oriented network device, such as Ethernet, can be used as a boot file source. See Chapter 11, “Network Devices” for the requirements of this type of device.
pci	A PCI bus node to which PCI plug-in devices can be attached. See Chapter 10, “Memory-Mapped Buses” for the requirements of this type of device.
serial	Byte-oriented device, such as a serial port, usable for console input and/or console output. See Chapter 12, “Serial Devices” for the requirements of this type of device.

See also: `device-type`, `property`.

devsel-speed

Contains the value of the “DEVSEL timing” field (Bits 9-10) of the “Status” register from the Configuration Space header. This field describes the timing of the DEVSEL# output of the device.

The property value is the register’s value encoded with `encode-int`. A value of 0 indicates fast, 1 indicates medium and 2 indicates slow timing.

See also: *PCI Local Bus Specification*

existing

Specifies all of the regions physical addresses actually installed in the system.

fast-back-to-back

This property should be present only if the fast back-to-back capable field (Bit 7) is set in the status register from the configuration space header. That field indicates whether the device is capable of accepting fast back-to-back transactions when the transactions are not to the same agent.

See also: PCI Local Bus Specification.

has-fcode

This property should be present only if this device node creation involved FCode program evaluation, as opposed to being completely automatically created from information in configuration registers.

fast-back-to-back

Should be present only if the “Fast Back-to-Back Capable” field (Bit 7) is set in the status register from the Configuration Space header. That field indicates whether the device is capable of accepting fast back-to-back transactions when they are not to the same agent.

See also: *PCI Local Bus Specification*.

has-fcode

Should be present only if the creation of this device node involved the evaluation of an FCode program rather than automatic creation from information in configuration registers.

height

Associated with `display` devices. Encoded with `encode-int`, the property value specifies the number of displayable pixels in the “y” direction of the display.

interrupts

This optional property declares the interrupt level(s) for this plug-in device. The contents are one or more integers. Note that the bus-level interrupt (not the CPU-level interrupt) is specified.

For SBus devices, SBus interrupt levels 1-7 are allowed. The correct choice for your interrupt level will depend on your latency requirements. Typical usage is: video - SBus level 5, Ethernet - SBus level 4, SCSI and DMA - SBus level 3. SBus levels 6 and 7 should only be used with great care, otherwise significant system performance degradation may occur.

Because of previous usage of the `intr` property instead of the `interrupts` property in earlier systems, we recommend that both `intr` and `interrupts` be declared in FCode for SBus cards. However, cards which only declare `intr` should continue to work, as current systems automatically generate the `interrupts` property for you as required.

To declare a single interrupt (level 5), used as:

```
5 encode-int " interrupts" property
```

To declare two interrupts (levels 3 and 5), used as:

```
5 encode-int 3 encode-int encode+ " interrupts" property
```

See also: `interrupts`, `property`

For PCI devices, this property should be present only if the function represented by this node is connected to a PCI expansion connector's interrupt line. The value of this property is determined from the contents of the interrupt pin register from the configuration space header.

The property value is the register's value encoded with `encode-int`. The defined values are:

Table 7-2 interrupts Property Value Encoding

Value	Description
1	The device uses the INTA# interrupt line
2	The device uses the INTB# interrupt line
3	The device uses the INTC# interrupt line
4	The device uses the INTD# interrupt line

The `interrupts` property is used to report the interrupt pin that the card uses, strictly within the domain of interrupts defined by the PCI specification.

It is the responsibility of the operating system's PCI bus driver code to translate the interrupts reported by its children into the interrupt domain of its parent.

This makes it possible to write portable, system-independent FCode drivers, because the FCode driver does not need to know system-specific information about the way that the system handles interrupts. The system-specific information is known by the code that handles the system component that actually performs the hardware mapping from PCI interrupt pins to whatever interrupt facilities exist on the system.

In some cases, the mapping may even be hierarchical. For example, a UPA-to-PCI bus bridge might translate PCI interrupt pins into UPA interrupt vectors.

See also: *PCI Local Bus Specification*.

linebytes

Associated with display devices. Encoded with `encode-int`, the property value specifies the number of pixels between consecutive scan lines of the display.

little-endian-aperture

This property is associated with display devices. Encoded identically to "reg" for the corresponding bus, the property value contains the address of the little-endian aperture of the frame buffer (in other words, the address range through which the frame buffer can be addressed in little-endian mode).

local-mac-address

Used with devices whose `device_type` is `network`, this should be present only if the device has a built-in, 48-bit, IEEE 802.3-style Media Access Control (MAC) address. The system may or may not use this address in order to access this device. Encoded with `encode-bytes`.

See also: `mac-address`, "`mac-address`", `property`, and Chapter 11, "Network Devices".

mac-address

Must be created by the `open` method of network devices to indicate the Media Access Control (MAC) address that this device is currently using. This value may or may not be the same as any `local-mac-address` property.

This property is typically used by client programs that determine which network address was used by the network interface from which the client program was loaded.

The property value is the six-byte MAC address encoded with `encode-byte`. Here's how it is made up:

1. If a plug-in device has an assigned MAC address from the factory, this address is published as the value for `local-mac-address`.
2. The system (based on various factors such as presence or absence of `local-mac-address` and/or the value of the NVRAM parameter `local-mac-address?`) determines the address for the plug-in device to use. The value returned by the `mac-address` FCode is set to this address.
3. The plug-in device then reports the address it is using by publishing the `mac-address` property.

For example:

For a well-behaved plug-in “network” device (which has a factory-unique MAC address but can use another system-supplied MAC address if desired by the system), the FCode would appear as:

```
create mac-address 8 c, 0 c, 20 c, 0 c, 14 c, 5e c,  
mac-address          encode-bytes " mac-address"          property  
(plus code to "assign" the correct mac-address value into registers)
```

See also: `mac-address`, `"local-mac-address"`, `property` and Chapter 11, “Network Devices”.

max-frame-size

When declared in “network” devices, indicates the maximum packet size (in bytes) that the physical layer of the device can transmit. This property can be used by client programs to allocate buffers of the appropriate length.

Usage:

```
4000 encode-int " max-frame-size" property
```

See also: `property` and Chapter 11, “Network Devices”.

max-latency

This property contains the value of the `Max_Lat` register from the configuration space header. That register specifies how frequently the device needs to gain access to the PCI bus. The value is given in units of 250 nanoseconds. A value of zero indicates that the device has no major requirements for the setting of the Latency Timers.

The property value is the register's value encoded with `encode-int`.

See also: *PCI Local Bus Specification*.

min-grant

This property contains the value of the Min_Gnt register from the configuration space header. That register specifies how long a burst period the device needs assuming a clock frequency of 33 MHz. The value is given in units of 250 nanoseconds. A value of zero indicates that the device has no major requirements for the setting of the Latency Timers.

The property value is the register's value encoded with `encode-int`.

See also: *PCI Local Bus Specification*.

model

Identifies the model name and number (including revision) for a device, for manufacturing and field-service purposes.

The 'model' property is useful to identify the specific piece of hardware (the plug-in card), as opposed to the name property (since several different but functionally-equivalent cards would have the same "name" property, thus calling the same device driver). Although the "model" property is good to have in general, it generally does not have any other specific purpose.

The property value format is arbitrary, but conventional usage is to begin the string with the manufacturer's name (as with the "name" property) and to end it with the revision level.

Usage:

```
" SUNW,501-1415-1" encode-string " model" property
```

See also: "name", model, property.

name

Specifies the manufacturer's name and device name. All device nodes *must* publish this property. The "name" property can be used to match a particular operating system device driver with the device.

The property value is an arbitrary string. Any combination of one to 31 printable characters is allowed, except for “@”, “.” or “/”. The string may contain one comma, at most. Embedded spaces are not allowed.

IEEE Standard 1275-1994 specifies three different formats for the manufacturer’s name portion of the property value where two of those formats are strongly preferred.

For United States companies that have publicly listed stock, the most practical form of name is to use the company’s stock symbol (e.g. SUNW for Sun Microsystems, Inc.). This option is also available to any company in the world provided that there is no duplication of the company’s stock symbol on either the New York Stock Exchange or the NASDAQ exchange. If a non-U.S. company’s stock is traded as an American Depository Receipt (ADR), the ADR symbol satisfies this requirement. A prime advantage of this form of manufacturer’s name is that such stock symbols are very human-readable.

Alternatively, a company may obtain an *organizationally unique identifier* (OUI) from the IEEE Registration Authority Committee. This is a 24-bit number that is guaranteed to be unique world-wide. Companies that have obtained an OUI would use a sequence of hexadecimal digits of the form “0NNNNNN” for the manufacturer’s name portion of the property. This form of name has the disadvantage that the manufacturer is not easily recognizable.

Each manufacturer may devise its own format for the device name portion of the property value.

Here is an example usage:

```
" SUNW,bison-printer" encode-string " name" property
```

You may also use the `name` command to create this property.

See also: `name`, `property`, `device-name`.

page-size

This property specifies the number of bytes in the smallest mappable region of virtual address space managed by the `/mmu` package.

power-consumption

This property describes the device's maximum power consumption (in microwatts) categorized by the various power rails and the device's power-management state (standby or fully-on).

The property value is a list of up to ten integers encoded with `encode-int` in the following order:

- Unspecified standby
- Unspecified full-on
- +5V standby
- +5V full-on
- +3.3V standby
- +3.3V full-on
- I/O power standby
- I/O power full-on
- Reserved standby
- Reserved full-on

The unspecified entries indicate that it is unknown how the power is divided among the various rails. The unspecified entries must be zero if any of the other entries are non-zero. The unspecified entries are provided so that the "power-consumption" property can be created for devices without FCode, from the information on the PRSNT1# and PRSNT2# connector pins.

If the number of integers in the encoded property value is less than ten, the power consumption is zero for the cases corresponding to the missing entries. For example, if there are four integers, they correspond to the two unspecified and the two "+5" quantities, and the others are zero.

The following code would create a "power-consumption" property for a device with +5V standby consumption of 100 mA and +5V full-on consumption of 2.5A:

```
0 encode-int 0 encode-int encode+ \ Set unspecified values to zero
500000 encode-int encode+          \ 100 mA@5V = 500,000 uW standby
12500000 encode-int encode+        \ 2.5A@5V = 12,500,000 uW full-on
" power-consumption" property
```

ranges

The `ranges` property is a list of child-to-parent physical address correspondences required for most hierarchical devices.

`ranges` is a property for bus devices, particularly those buses whose children can be accessed with CPU load and store operations (as opposed to buses like SCSI, whose children are accessed with a command protocol).

The "ranges" property value describes the correspondence between the part of the physical address space of the bus node's parent available for use by the bus node (the parent address space), and the physical address space defined by the bus node for its children (the child address space).

The "ranges" property value is a sequence of specifications.

```
child-phys parent-phys size
```

- *child-phys* is a starting address in the child physical address space defined by the bus node
- *parent-phys* is a starting address in the physical address space of the parent of the bus node
- *size* is the length in bytes of the address range.

The specification means that there is a one-to-one correspondence between the child addresses and the parent addresses in that range. The parent addresses given are always relative to the parent's address space.

Each starting address is represented using the physical address representation as two 32-bit numbers (one for *space* and one for *offset*). *size* is encoded as an unsigned integer.

The total size of each such specification is five 32-bit numbers (two for each of the two addresses, plus one for the size). Successive specifications are encoded sequentially. A space with length $2^{**}(\text{number of bits in a machine word})$ is represented with a size of 0.

The specifications should be sorted in ascending order of the child address. The address ranges thus described need not be contiguous in either the child space or the parent space. Also, the entire child space must be described in terms of parent addresses, but not all of the parent address space available to the bus device need be used for child addresses (the bus device might reserve some addresses for its own purposes, for instance).

For example, suppose that a 4-slot 25-bit SBus is attached to a machine whose physical address space consists of a 32-bit memory space (*space*=0) and a 32-bit I/O space (*space*=1). The SBus slots appear in I/O space at address 0xf800.0000, 0xfa00.0000, 0xfc00.0000, and 0xfe00.0000. In terms of the SBus's parent address space, the SBus device has available for its purposes the offsets from 0xf800.0000 through 0xffff.ffff in space 1 of its parent.

The SBus device defines for its children the spaces 0, 1, 2, and 3, all starting at offset 0 and running for 0x200.0000 bytes. In this case the SBus device uses all the address space given to it by its parent for the SBus children, and reserves none of the addresses for itself. The "ranges" property for the SBus device would contain the encoded form of the following sequence of numbers:

Table 7-3 Child-Parent Address Relationships

Child Address		Parent Address		
Space	Offset	Space	Offset	Size
0	0	1	f800.0000	200.0000
1	0	1	fa00.0000	200.0000
2	0	1	fc00.0000	200.0000
3	0	1	fe00.0000	200.0000

Here the high components of the child address represent the SBus slot numbers, and the high component of the parent address represents I/O space.

If `ranges` exists but its value is of 0 length, the bus's child address space is identical to its parent address space.

If the `ranges` property for a particular bus device node is nonexistent, code using that device should use an appropriate default interpretation. Some examples include the following:

- SBus node: Missing `ranges` means that the version of OpenBoot was created before the `ranges` property came into existence. Code should supply the correct ranges based on the machine type, from the finite set of machines that existed before `ranges` came into existence.
- Machine node: The machine node has no parent. Therefore the correspondence between its child and parent address spaces is meaningless, and there is no need for `ranges`.
- SCSI host adapter node: The child address space is not directly addressable, thus `ranges` would be meaningless.

The distinction between `reg` and `ranges` is as follows:

- `reg` is supposed to represent the actual device registers in the parent address space. For a bus adapter, this would be such as configuration/mode/initialization registers.
- `ranges` represents the correspondence between a bus adapter's child and parent address spaces.

Most packages do not need to be concerned with `ranges`. These properties are mainly to communicate with stand-alone programs. One exception could be a bus extender or adaptor.

See also: Chapter 10, "Memory-Mapped Buses".

For a PCI node in a PowerPC Reference Platform (PPCRP) compliant machine, the total size of each such specification is five 32-bit numbers (one for the parent address space, three for the child address space, and one for the size). Successive specifications are encoded sequentially. A space with length $2^{**}(\text{number of bits in a machine word})$ is represented with a size of 0.

Sort the specifications in ascending order of *child-phys*, in accordance with recommendations. The address ranges thus described need not be contiguous in either the child space or the parent space. Also, the entire child space must be described in terms of parent addresses, but not all of the parent address space available to the bus device need be used for child addresses (the bus device might reserve some addresses for its own purposes, for instance).

In the PPCRP machine example, consider a 4-slot 32-bit PCI bus attached to a machine whose physical address space consists of a 32-bit “memory” space (Bit 31 = 0) and a 32-bit “I/O” space (Bit 31 = 1).

- ISA I/O space appears in the parent’s “I/O” space at 0x8000.0000; the size is 0x1.0000.
- A reserved block of addresses begins at 0x8001.0000; the size is 0x7f.0000.
- PCI configuration space begins at 0x8080.0000; the size is 0x80.0000. The configuration registers of the individual PCI slots appear at addresses 0x8080.1000, 0x8080.2000, 0x8080.4000, and 0x8080.8000.
- PCI I/O space begins at 0x8100.0000; the size is 3e80.0000.
- Parity/interrupt vectors begin at 0xbf80.0000; their size is 0x80.0000.
- PCI memory space begins at 0xc000.0000; the size is 3e00.0000.

The PCI device defines:

Configuration spaces for Devices 1 through 4 that each begin at 0x0000.0000; their size is 0x100 bytes.

ISA I/O space that begins at 0x0000.0000; the size is 0x1.0000.

PCI I/O space that begins at 0x0001.0000; the size is 0x3e80.0000.

A 32-bit, PCI memory space that begins at 0x0000.0000; the size is 0x3e00.0000.

The `ranges` property for the PCI device would contain the encoded form of the following sequence of numbers:

Table 7-4 Child-Parent Address Relationships for a PCI Node in a PPCRP Machine

Child Address				Parent Address	Size
Function	phys.hi	phys.mid	phys.lo		
SCSI	0000.0800	0000.0000	0000.0000	8080.1000	0000.0800
Slot A	0000.1000	0000.0000	0000.0000	8080.2000	0000.0800
Slot B	0000.1800	0000.0000	0000.0000	8080.4000	0000.0800
Slot C	0000.2000	0000.0000	0000.0000	8080.8000	0000.0800
ISA I/O space	0100.0000	0000.0000	0000.0000	8000.0000	0001.0000
PCI I/O space	0100.0000	0000.0000	0001.0000	8100.0000	3e80.0000
PCI Memory space	0200.0000	0000.0000	0000.0000	c000.0000	3e00.0000

Here the *phys.hi* component of the child address represents the type of address space and the PCI device numbers; Bit 31 of the parent address represents “I/O space.” (Please see the *PCI Bus Binding to IEEE Standard 1275-1994* for a detailed description of the encoding of the *phys.hi* field.)

The code to create this ranges property is:

```

\ SCSI Configuration Space
0000.0800 encode-int encode+ 0 encode-int encode+ 0 encode-int encode+
8080.1000 encode-int encode+
800 encode-int encode+

\ Slot A Configuration Space
0000.1000 encode-int encode+ 0 encode-int encode+ 0 encode-int encode+
8080.2000 encode-int encode+
800 encode-int encode+

\ Slot B Configuration Space
0000.1800 encode-int encode+ 0 encode-int encode+ 0 encode-int encode+
8080.4000 encode-int encode+
800 encode-int encode+

\ Slot C Configuration Space
0000.2000 encode-int encode+ 0 encode-int encode+ 0 encode-int encode+
8080.8000 encode-int encode+
800 encode-int encode+

\ ISA I/O space
0100.0000 encode-int encode+ 0 encode-int encode+ 0 encode-int encode+
8000.0000 encode-int encode+
1.0000 encode-int encode+

\ PCI I/O space
0100.0000 encode-int encode+ 0 encode-int encode+ 1.0000 encode-int encode+
8100.0000 encode-int encode+
3e80.0000 encode-int encode+

\ PCI Memory space
0200.0000 encode-int encode+ 0 encode-int encode+ 0 encode-int encode+
c000.0000 encode-int encode+
3e00.0000 encode-int encode+
" ranges" property

```

If `ranges` exists but its value is of 0 length, the bus's child address space is identical to its parent address space.

If the `ranges` property for a particular bus device node is nonexistent, code using that device should use an appropriate default interpretation. Here are some examples:

- Root node: The root node has no parent. Therefore the correspondence between its child and parent address spaces is meaningless, and there is no need for `ranges`.
- SCSI host adapter node: The child address space is not directly addressable, thus `ranges` would be meaningless.
- For memory-mapped bus devices where a `ranges` property would be meaningful, the absence of a `ranges` property is conventionally interpreted to mean that the parent and child address spaces are identical.

`reg` and `ranges` are differentiated thus:

- `reg` represents the device registers in the parent address space. For a bus adapter, this would be such as configuration/mode/initialization registers.
- `ranges` represents the correspondence between a bus adapter's child and parent address spaces.

Most packages do not need to consider `ranges`. This property is mainly used for bus bridges. The firmware system does not itself use this property. `ranges` is mainly used by operating systems that wish to auto-configure themselves.

See also: Chapter 10, "Memory-Mapped Buses".

reg

This property declares the location and size of onboard registers for its device. The FCode program for every plug-in SBus and PCI device *must* declare this property.

For SBus, the contents are one or more (*phys*, *size*) pairs. Each pair specifies an addressable region of the device. An FCode PROM at location 0 of the device is generally *not* declared, except in the case where there are no other regions to declare.

For an SBus device, to declare two register fields at 10.0000-10.00ff and 20.0000-20.037f, use the following:

```
my-address 10.0000 + my-space encode-phys \ Offset#1
100 encode-int encode+ \ Merge size#1

my-address 20.0000 + my-space encode-phys encode+ \ Merge
offset#2
380 encode-int encode+ \ Merge size#2

" reg" property
```

In the first (*phys,size*) pair for a PCI device, the *phys* component is the configuration space address of the beginning of the function's set of configuration registers; the size component is zero. Each additional (*phys,size*) pair specifies the address and characteristics of an addressable region of memory space or I/O space associated with the function including the PCI Expansion ROM.

For a PCI device, the order of the pairs should be:

- An entry describing the configuration space for the device.
- An entry for each active base address register (BAR), in configuration space order, describing the entire space mapped by that BAR.
- An entry describing the Expansion ROM BAR, if the device has an Expansion ROM.
- An entry for each non-relocatable addressable resource.

In the event that a function has an addressable region that can be accessed relative to more than one base address register (for example, in memory space relative to one base register, and in I/O space relative to another), only the primary access path (typically, the one in memory space) is listed in the "reg" property, and the secondary access path is listed in the `alternate-reg` property.

For PCI, *phys* is (*phys.lo phys.mid phys.hi*), encoded with `encode-phys`, and *size* is a pair of integers, each encoded with `encode-int`. The first integer denotes the most-significant 32 bits of the 64-bit region size, and the second integer denotes the least-significant 32 bits thereof.

For example, to declare a PCI device with:

- A register field of size 0x100 in 32-bit memory space that is controlled by the first 32-bit base address register.
- A register field of size 0x440 in I/O space that is controlled by the second 32-bit base address register. The register field of interest is offset from the base address register by 0x20.0000.
- A 128Kbyte PCI Expansion ROM.
- A non-relocatable field at 0-fff in I/O space.

use the following:

```
hex
my-address my-space encode-phys \ Config space regs
0 encode-int encode+ 0 encode-int encode+
0 0 my-space 0200.0010 or encode-phys encode+ \ Memory space
0 encode-int encode+ 100 encode-int encode+ \ BAR at 0x10
20.0000 0 my-space 0100.0014 or encode-phys \ I/O space
encode+ \ BAR at 0x14
0 encode-int encode+ 440 encode-int encode+

0 0 my-space 0200.0030 or encode-phys encode+ \ PCI Expansion ROM
0 encode-int encode+ 2.0000 encode-int encode+ \ memory space
0 0 my-space 8100.0000 or encode-phys encode+ \ Non-relocatable
0 encode-int encode+ 1000 encode-int encode+ \ memory space
" reg" property
```

In some non-PCI cases, the `reg` command may also be used to create this property. However, `reg` may only be used on buses for which `#size-cells` is one and only when a single “`reg`” property component is required. Consequently, `reg` is never used with PCI devices which require at least three “`reg`” property component—in other words

- One component for the card's configuration space registers
- At least one for the device's functional registers and

- One for the PCI Expansion ROM).

Note – The contents of the “`reg`” property are used by OpenBoot firmware to determine how large a portion of the system's virtual address space to reserve for use by the card. It is important that the *size* arguments be as large as the actual available addressable resource. If the *size* argument for a region were to be declared smaller than that actually available, and if the driver or a user were to later add a legitimate offset that was larger than *size* to the base address of the region, the resulting virtual address might be within the virtual address space of another card.

See the *PCI Bus Binding to IEEE Standard 1275-1994* for the encoding details.

See also: `reg`, `property`

revision-id

This property contains the value of the revision ID register from the configuration space header. That register specifies a device-specific revision identifier that is chosen by the vendor. Zero is an acceptable value.

The property value is the register's value encoded with `encode-int`.

See also: PCI Local Bus Specification.

slot-names

This property describes the external labeling of plug-in slots.

The property value is an integer, encoded with `encode-int`, followed by a list of strings, each encoded with `encode-string`.

The integer portion of the property value is a bit mask of available slots; for each add-in slot on the bus, the bit corresponding to that slot's Device Number is set. The least-significant bit corresponds to Device Number 0, the next bit corresponds to Device Number 1, and so on. The number of following strings is the same as the number of slots; the first string gives the label that is printed on the chassis for the slot with the smallest Device Number, and so on.

status

This optional property indicates the operational status of the device.

Absence of this property means that the operational status of the device is unknown or okay.

If this property is present, the value is a string indicating the status of the device, as follows:

Table 7-5 status Property values

Status Value	Meaning
<i>okay</i>	The device is believed to be operational.
<i>disabled</i>	The device represented by this node is not operational, but it might become operational in the future (e.g. an external switch is turned off, or something isn't plugged in).
<i>fail</i>	The device represented by this node is not operational because a fault has been detected, and it is unlikely that the device will become operational without repair. No additional failure details are available.
<i>fail-xxx</i>	The device represented by this node is not operational because a fault has been detected, and it is unlikely that the device will become operational without repair. "xxx" is additional human-readable information about the particular fault condition that was detected.

Used as:

```
" disabled" encode-string " status" property
```

See also: [property](#).

translations

This property contains an array of (*phys-addr*, *virt-addr*, *size*) entries describing the address translations currently in use by OpenBoot. Those operating systems calling on OpenBoot services while taking over the memory management function must create all translations described by this property's value.

vendor-id

This property contains the value of the vendor ID register from the configuration space header. That register identifies the manufacturer of the device. Vendor identifiers are assigned by the PCI SIG to ensure uniqueness. 0xffff is an invalid value for vendor-id.

The property value is the register's value encoded with `encode-int`.

See also: *PCI Local Bus Specification*.

width

Associated with `display` devices. Encoded with `encode-int`, the property value specifies the number of displayable pixels in the "x" direction of the display.

Manipulating Properties

Property Creation and Modification

Use the `FCode` `Function` property to create new properties or modify values of existing properties.

There are some special property-publishing `FCode`s, designed for use in common situations:

- `reg` is a quick way to create a "reg" property that describes the location of the package's physical resources.
- `model` is a quick way to create the "model" property.
- `device-name` is a quick way to create the "name" property.
- `delete-property` completely removes a property.

Property Values

Various kinds of information can be stored in a property value byte array by using property encoding and decoding methods. The encoding format is machine-independent. Property value representation is independent of the byte organization and word alignment characteristics of a processor.

A property's data type must be recognized by any software that uses it. In other words, property value data types are not self-identifying. Furthermore, the presence or absence of a property with a particular name can encode a true/false flag; such a property may have a zero-length property value.

Property Encoding

There are three FCodes for encoding a basic piece of data into a property value and one for concatenating the basic pieces of data for a property with multiple values.

<code>encode-int</code>	encodes a number
<code>encode-string</code>	encodes a string
<code>encode-bytes</code>	encodes a sequence of bytes
<code>encode+</code>	is used to concatenate two previously encoded, basic pieces of data
<code>encode-phys</code>	is an FCode that encodes a physical address (hiding all the relative addressing information)

Property Retrieval

There are three property value retrieving words, `get-my-property`, `get-inherited-property`, and `get-package-property`.

- Use `get-my-property` if the property desired exists for the package being defined.
- Use `get-package-property` if the property exists in some other package. In this case, you must first find the phandle of the other package, perhaps by using `find-package`.
- Use `get-inherited-property` if the property exists somewhere in the chain of parent instances between the package being defined and the root node of the machine.

Note – Using `get-inherited-property` can be a bad idea because you don't know who supplied the data.

FCode programs often do not retrieve property values. Such programs usually know the values of their own properties implicitly, and often interact with their parents by calling well-known parent methods.

For example, suppose a particular SBus FCode package calls DVMA to transfer data between a device and memory.

It could use `my-parent ihandle>phandle get-package-property` to find the value of a property named `slave-only`. `slave-only` will be a property of the parent node of the package being defined, if it exists.

The value of the property is a bit mask of the SBus slots that do *not* support DVMA. Then the package would look at `my-unit` or `my-space` to get its slot number. The two pieces of information will tell the package whether or not it can use DVMA.

Property Decoding

Once a package has found a property's value, it must decode the value to forms it can recognize. If the value is the representation of an integer, use `decode-int` to generate the number as a binary number on the stack. If the value is the representation of a string, use `decode-string`. Both of these FCodes act as parsers — they will also return the unused portion of the value for further decoding.

Other kinds of values can be decoded by `left-parse-string` or package-specific decoders. Note that the package must be able to decode the value of a property.

There is no `decode-bytes` function, but it is easy to synthesize if you need it.

```
: decode-bytes ( addr1 len1 #bytes -- addr len2 addr1 #bytes )
  tuck -      ( addr1 #bytes len2 )
  >r 2dup +   ( addr1 #bytes addr2 ) ( R: len2 )
  r> 2swap
;
```

Property-Specific FCodes

Following is a summary of property-specific FCodes. See the individual dictionary entries in Chapter 14, “FCODE Dictionary”, for more information.

Table 7-6 Property-specific FCodes

Name	Stack Diagram	Description
Property Creation		
property	(prop-addr prop-len name-addr name-len --)	Create a property named <i>name-addr name-len</i> with the value <i>prop-addr prop-len</i> .
device-type	(addr len --)	Shorthand word to create the <code>device_type</code> property with the value <i>addr len</i> .
model	(addr len --)	Shorthand word to create the <code>model</code> property with the value <i>addr len</i> .
name	(addr len --)	Shorthand macro to create the <code>name</code> property with the value <i>addr len</i> .
reg	(phys.lo ... phys.hi size --)	Shorthand word to create the <code>reg</code> property.
device-name	(addr len --)	Shorthand word to create the <code>name</code> property with the value <i>addr len</i> . Similar to <code>name</code> , but uses only one FCode instead of creating a macro.
delete-property	(name-addr name-len --)	Delete the desired property.
Property Encoding		
encode-int	(n -- prop-addr prop-len)	Converts an integer to a prop-encoded-array.
encode-phys	(phys.lo ... phys.hi -- prop-addr prop-len)	Converts a physical unit pair to a prop-encoded-array.
encode-string	(addr len -- prop-addr prop-len)	Converts a text string to a prop-encoded-array.
encode+	(prop-addr1 prop-len1 prop-addr2 prop-len2 -- prop-addr prop-len1+2)	Concatenate two prop-encoded-array structures. They must have been created sequentially.
encode-bytes	(addr len -- prop-addr prop-len)	Converts a byte array to a prop-encoded-array. Similar to <code>encode-string</code> , except no trailing null is appended.
Property Decoding		
decode-int	(prop-addr prop-len -- prop-addr2 prop-len2 n)	Converts a prop-encoded-array string to an integer.

Table 7-6 Property-specific FCodes (Continued)

Name	Stack Diagram	Description
decode-string	(prop-addr prop-len -- prop-addr2 prop-len2 addr len)	Converts a prop-encoded-array string to a normal string.
Property Retrieval		
get-my-property	(name-addr name-len -- true prop-addr prop-len false)	Returns the prop-encoded-array contents for the property <i>addr len</i> in the current instance, or <i>true</i> if not found.
get-package-property	(addr len phandle -- true prop-addr prop-len false)	Returns the prop-encoded-array contents for the property <i>addr len</i> in the package <i>phandle</i> , or <i>true</i> if not found.
get-inherited-property	(addr len -- true prop-addr prop-len false)	Returns the prop-encoded-array contents for the property <i>addr len</i> , or <i>true</i> if not found. The current package instance is searched first. If not found, the parent is searched next, then the parent's parent, and so on.

Block Devices

Block devices are nonvolatile mass storage devices whose information can be accessed in any order. Hard disks, floppy disks, and CD-ROMs are examples of block devices. OpenBoot typically uses block devices for booting.

This device type generally applies to disk devices, but as far as OpenBoot is concerned, it simply means that the device “looks like a disk” at the OpenBoot software interface level.

The block device’s FCode must declare the `block` device type, and implement the methods `open` and `close`, as well as the methods described in “Required Methods”.

Although packages of the `block` device type present a byte-oriented interface to the rest of the system, the associated hardware devices are usually block-oriented i.e. the device reads and writes data in blocks (groups of, for example, 512 or 2048 bytes). The standard `/deblocator` support package assists in the presentation of a byte-oriented interface overlaying a block-oriented interface, implementing a buffering layer that hides the underlying block length.

Block devices are often subdivided into several logical partitions, as defined by a disk label - a special block, usually the first one on the device, which contains information about the device. The driver is responsible for appropriately interpreting a disk label. The driver may use the standard `/disk-label` support package if the device does not implement a specialized label. The `/disk-label` support package interprets a system-dependent label format.

Since the disk booting protocol usually depends on the label format, the standard `/disk-label` support package also implements a `load` method for the corresponding boot protocol.

Byte Devices

Byte devices are sequential-access mass storage devices, for example tape devices. OpenBoot typically uses byte devices for booting.

The byte device's FCode program must declare the `byte` device type, and implement the `open` and `close` methods in addition to those described in "Required Methods".

Although packages of the `byte` device type present a byte-oriented interface to the rest of the system, the associated hardware devices are usually record-oriented; the device reads and writes data in records containing more than one byte. The records may be fixed or variable length. The standard `/deblocator` support package assists in presenting a byte-oriented interface overlaying a record-oriented interface, implementing a buffering layer that hides the underlying record structure.

Required Methods

`block-size (-- block-len)`

Returns the record size *block-len* (in bytes) of all data transfers to or from the device. The most common value for *block-len* is 512.

This method is only required if the `/deblocator` support package is used.

`load (addr -- size)`

`load` works differently for block and byte devices:

With block devices, it loads a stand-alone program from the device into memory at *addr*; *size* is the size in bytes of the program loaded. If the device can contain several such programs, the instance arguments returned by `my-args` can be used to select the program desired. `open` is executed before `load` is invoked.

With byte devices, `load` reads a stand-alone program from the tape file specified by the value of the argument string given by `my-args`. That value is the string representation of a decimal integer. If the argument string is null, tape file 0 is used. `load` places the program in memory at `addr`, returning the size of the read-in program in bytes.

`max-transfer (-- max-len)`

Returns the size in bytes of the largest single transfer that the device can perform. `max-transfer` is expected to be a multiple of `block-size`.

This method is only required if the `/deblocker` support package is used.

`read (addr len -- actual)`

Read at most `len` bytes from the device into memory at `addr`. `actual` is the number of bytes read. If the number of bytes read is 0 or negative, the read failed. Note that `len` need not be a multiple of the device's normal block size.

`read-blocks (addr block# #blocks -- #read)`

Read `#blocks` records of length `block-size` bytes each from the device, starting at device block `block#`, into memory at address `addr`. `#read` is the number of blocks actually read.

This method is only required if the `/deblocker` support package is used.

`seek (poslow poshigh -- status) for block; (offset file# -- error?) for byte`

`seek` works differently depending on whether it's being used with a block or byte device.

For block devices, `seek` sets the device position for the next read or write. The position is the byte offset from the beginning of the device specified by the 64-bit number which is the concatenation of `poshigh` and `poslow`. `status` is -1 if the seek fails, and 0 or 1 if it succeeds.

For byte devices, it seeks to the byte `offset` in file `file#`. If `offset` and `file#` are both 0, rewind the tape. `error?` is -1 if seek fails, and 0 if seek succeeds.

write (*addr len -- actual*)

Write *len* bytes from memory at *addr* to the device. *actual* is the number of bytes written. If *actual* is less than *len*, the write did not succeed. *len* need not be a multiple of the device's normal block size.

write-blocks (*addr block# #blocks -- #written*)

Write *#blocks* records of length `block-size` bytes each to the device, starting at block *block#*, from memory at *addr*. *#written* is the number of blocks actually written.

If the device is not capable of random access (e.g. a sequential access tape device), *block#* is ignored.

This method is only required if the `/deblocator` support package is used.

Required Properties

Table 8-1 Required Properties of Block and Byte Devices

Property Name	Sample Value
name	" SUNW,my-scsi"
reg	list of registers (device-dependent)
device_type	block or byte

Device Driver Examples

The structure of the device tree for the sample card supported by the sample device drivers in this chapter is as follows:

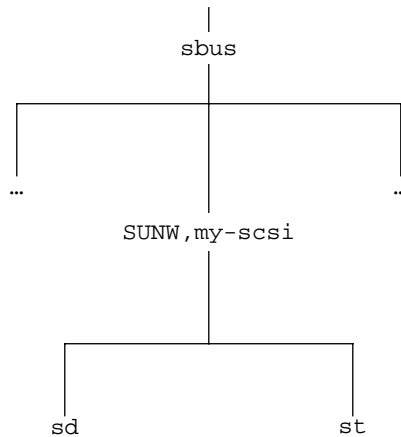


Figure 8-1 Sample Device Tree

Simple Block Device Driver

Code Example 8-1 Simple Block Device Driver

```

\ This is at a stage where each leaf node can be used only as a non-bootable device.
\ It only creates nodes and publishes necessary properties to identify the device.
fcode-version3
hex
  " SUNW,my-scsi"  encode-string " name" property

  h# 20.0000    constant scsi-offset
  h# 40         constant /scsi
  my-address scsi-offset + my-space /scsi  reg

  new-device   \ missing "reg" indicates a SCSI "wild-card" node
    " sd"      encode-string " name" property
  finish-device

  new-device   \ missing "reg" indicates a SCSI "wild-card" node
    " st"      encode-string " name" property
  finish-device
end0

```

Extended Block Device Driver

Code Example 8-2 Sample Driver for my-scsi Device

```

\ It is still a non-bootable device. The purpose is to show how an intermediate stage
\ of driver can be used to debug board during development. In addition to publishing
\ the properties, this sample driver shows methods to access, test and control
\ "SUNW,my-scsi" device.

\ The following main methods are provided for "SUNW,my-scsi" device.
\ open ( -- okay? )
\ close ( -- )
\ reset ( -- )
\ selftest ( -- error? )
fcode-version3
hex
  headers

  h# 20.0000    constant scsi-offset
  h# 40         constant /scsi
  d# 25.000.000 constant clock-frequency

```


Code Example 8-2 Sample Driver for my-scsi Device (Continued)

```

: identify-me ( -- )
  " SUNW,my-scsi" encode-string " name" property
  " scsi" device-type

  my-address scsi-offset + my-space /scsi reg
;
identify-me

h# 10.0000 constant dma-offset
h# 10      constant /dma
-1 instance value dma-chip

\ methods to access/control dma registers
: dmaaddress ( -- addr ) dma-chip 4 + ;
: dmacount   ( -- addr ) dma-chip 8 + ;
: dmaaddr@   ( -- n )     dmaaddress rl@ ;
: dmaaddr!   ( n -- )     dmaaddress rl! ;
: dmacount@  ( -- n )     dmacount rl@ ;
: dmacount!  ( n -- )     dmacount rl! ;
: dma-chip@  ( -- n )     dma-chip rl@ ;
: dma-chip!  ( n -- )     dma-chip rl! ;
: dma-btest  ( mask -- flag ) dma-chip@ and ;
: dma-bset   ( mask -- )     dma-chip@ or dma-chip! ;
: dma-breset ( mask -- ) not dma-btest dma-chip! ;

external

\ methods to allocate, map, unmap, free dma buffers
: decode-unit ( addr len -- low high ) decode-2int ;
: dma-alloc   ( size -- vaddr )         " dma-alloc" $call-parent ;
: dma-free    ( vaddr size -- )         " dma-free" $call-parent ;
: dma-map-in  ( vaddr size cache? -- devaddr ) " dma-map-in" $call-parent ;
: dma-map-out ( vaddr devaddr size -- )     " dma-map-out" $call-parent ;

\ Dma-sync could be a dummy routine if the parent device doesn't support.
: dma-sync ( virt-addr dev-addr size -- )
  " dma-sync" my-parent ['] $call-method catch if
    2drop 2drop 2drop
  then
;

: map-in ( addr space size -- virt ) " map-in" $call-parent ;

```

Code Example 8-2 Sample Driver for my-scsi Device (Continued)

```

: map-out ( virt size -- )          " map-out" $call-parent ;

headers
: dma-open ( -- )
    my-address dma-offset + my-space /dma map-in to dma-chip
;
: dma-close ( -- ) dma-chip /dma map-out -1 to dma-chip ;

-1 instance value scsi-init-id
-1 instance value scsi-chip
h# 20 constant /mbuf
-1 instance value mbuf
-1 instance value mbuf-dma
d# 6 constant /sense
-1 instance value sense-command
-1 instance value sense-cmd-dma
d# 8 constant #sense-bytes
-1 instance value sense-buf
-1 instance value sense-buf-dma
-1 instance value mbuf0
d# 12 constant /cndbuf
-1 instance value cndbuf
-1 instance value cndbuf-dma
-1 instance value scsi-statbuf

\ mapping and allocation routines for scsi
: map-scsi-chip ( -- )
    my-address scsi-offset + my-space /scsi map-in to scsi-chip
;
: unmap-scsi-chip ( -- ) scsi-chip /scsi map-out -1 to scsi-chip ;

\ After any changes to sense-command by CPU or any changes to sense-cmd-dma by
\ device, synchronize changes by issuing " sense-command sense-cmd-dma /sense
\ dma-sync " Similarly after any changes to sense-buf, sense-buf-dma, mbuf,
\ mbuf-dma, cndbuf or cndbuf-dma, synchronize changes by appropriately issuing
\ dma-sync map scsi chip and allocate buffers for "sense" command and status
: map-scsi ( -- )
    map-scsi-chip
    /sense dma-alloc to sense-command
    sense-command /sense false dma-map-in to sense-cmd-dma
    #sense-bytes dma-alloc to sense-buf
    sense-buf #sense-bytes false dma-map-in to sense-buf-dma

```

Code Example 8-2 Sample Driver for my-scsi Device (Continued)

```

    2 alloc-mem to scsi-statbuf
;

\ free buffers for "sense" command and status and unmap scsi chip
: unmap-scsi ( -- )
  scsi-statbuf 2 free-mem
  sense-buf sense-buf-dma #sense-bytes dma-sync \ redundant
  sense-buf sense-buf-dma #sense-bytes dma-map-out
  sense-buf #sense-bytes dma-free
  sense-command sense-cmd-dma /sense dma-sync \ redundant
  sense-command sense-cmd-dma /sense dma-map-out
  sense-command /sense dma-free
  -1 to sense-command
  -1 to sense-cmd-dma
  -1 to sense-buf
  -1 to scsi-statbuf
  -1 to sense-buf-dma
  unmap-scsi-chip
;

\ constants related to scsi commands
h# 0 constant nop
h# 1 constant flush-fifo
h# 2 constant reset-chip
h# 3 constant reset-scsi
h# 80 constant dma-nop

\ words to get scsi register addresses.
\ Each chip register is one byte, aligned on a 4-byte boundary.
: scsi+ ( offset -- addr ) scsi-chip + ;
: transfer-count-lo ( -- addr ) h# 0 scsi+ ;
: transfer-count-hi ( -- addr ) h# 4 scsi+ ;
: fifo ( -- addr ) h# 8 scsi+ ;
: command ( -- addr ) h# c scsi+ ;
: configuration ( -- addr ) h# 20 scsi+ ;
: scsi-test-reg ( -- addr ) h# 28 scsi+ ;

\ Read only registers:
: scsi-status ( -- addr ) h# 10 scsi+ ;
: interrupt-status ( -- addr ) h# 14 scsi+ ;
: sequence-step ( -- addr ) h# 18 scsi+ ;

```

Code Example 8-2 Sample Driver for my-scsi Device (Continued)

```

: fifo-flags          ( -- addr ) h# 1c scsi+ ;

\ Write only registers:
: select/reconnect-bus-id ( -- addr ) h# 10 scsi+ ;
: select/reconnect-timeout ( -- addr ) h# 14 scsi+ ;
: sync-period            ( -- addr ) h# 18 scsi+ ;
: sync-offset           ( -- addr ) h# 1c scsi+ ;
: clock-conversion-factor ( -- addr ) h# 24 scsi+ ;

\ words to read from/store to scsi registers.
: cnt@      ( -- w ) transfer-count-lo rb@ transfer-count-hi rb@ bwjoin ;
: fifo@     ( -- c ) fifo rb@ ;
: cmd@      ( -- c ) command rb@ ;
: stat@     ( -- c ) scsi-status rb@ ;
: istat@    ( -- c ) interrupt-status rb@ ;
: fifo-cnt  ( -- c ) fifo-flags rb@ h# 1f and ;
: data@     ( -- c ) begin fifo-cnt until fifo@ ;
: seq@      ( -- c ) sequence-step rb@ h# 7 and ;

: fifo! ( c -- ) fifo rb! ;
: cmd!  ( c -- ) command rb! ;
: cnt!  ( w -- ) wbsplit transfer-count-hi rb! transfer-count-lo rb! ;
: targ! ( c -- ) select/reconnect-bus-id rb! ;
: data! ( c -- ) begin fifo-cnt d# 16 <> until fifo! ;

\ scsi chip noop and initialization
: scsi-nop ( -- ) nop cmd! ;
: init-scsi ( -- ) reset-chip cmd! scsi-nop ;

: wait-istat-clear ( -- error? )
  d# 1000
  begin
    1 ms 1- ( count )
    dup 0= ( count expired? )
    istat@ ( count expired? istat )
    0= or ( count clear? )
  until ( count )
  0= if
    istat@ 0<> if
      cr ." Can't clear ESP interrupts: "
      ." Check SCSI Term. Power Fuse." cr
    true exit

```

Code Example 8-2 Sample Driver for my-scsi Device (Continued)

```

        then
        then
        false
;

: clk-conv-factor ( -- n ) clock-frequency d# 5.000.000 / 7 and ;

\ initialize scsi chip, tune time amount, set async operation mode, and set scsi
\ bus id
: reset-my-scsi ( -- error? )
    init-scsi
    h# 93 select/reconnect-timeout rb!
    0 sync-offset rb!
    clk-conv-factor clock-conversion-factor rb!
    h# 4 scsi-init-id 7 and or configuration rb!
    wait-istat-clear
;

: reset-bus ( -- error? )
    reset-scsi cmd! wait-istat-clear
;

: init-n-test ( -- ok? ) reset-my-scsi 0= ;

: get-buffers ( -- )
    h# 8000 dma-alloc to mbuf0
    /cmdbuf dma-alloc to cmdbuf
    cmdbuf /cmdbuf false dma-map-in to cmdbuf-dma
;

: give-buffers ( -- )
    mbuf0 h# 8000 dma-free -1 to mbuf0
    cmdbuf cmdbuf-dma /cmdbuf dma-sync \ redundant
    cmdbuf cmdbuf-dma /cmdbuf dma-map-out
    cmdbuf /cmdbuf dma-free
    -1 to cmdbuf -1 to cmdbuf-dma
;

: scsi-selftest ( -- fail? ) reset-my-scsi ;

\ dma-alloc and dma-map-in mbuf-dma

```

Code Example 8-2 Sample Driver for my-scsi Device (Continued)

```

: mbuf-alloc ( -- )
  /mbuf dma-alloc to mbuf
  mbuf /mbuf false dma-map-in to mbuf-dma
;

\ dma-map-out and dma-free mbuf-dma
: mbuf-free ( -- )
  mbuf mbuf-dma /mbuf dma-sync          \ redundant
  mbuf mbuf-dma /mbuf dma-map-out
  mbuf /mbuf dma-free
  -1 to mbuf
  -1 to mbuf-dma
;

external
\ If any routine was using buffers allocated by dma-alloc, and was using dma mapped
\ by dma-map-in, it would have to dma-sync those buffers after making any changes to
\ them.
: open ( -- success? )
  dma-open
  " scsi-initiator-id" get-inherited-property 0= if
    decode-int to scsi-init-id
    2drop
    map-scsi
    init-n-test          ( ok? )
    dup if               ( true )
      get-buffers        ( true )
    else
      unmap-scsi dma-close ( false )
    then                 ( success? )
  else
    ." Missing initiator id" cr false
    dma-close
  then                   ( success? )
;

: close ( -- )
  give-buffers unmap-scsi dma-close
;

: reset ( -- )
  dma-open map-scsi
  h# 80 dma-breset

```

Code Example 8-2 Sample Driver for my-scsi Device (Continued)

```
    reset-my-scsi drop reset-bus drop
    unmap-scsi dma-close
;

\ if scsi-selftest was actually using buffers allocated by mbuf-alloc, it would
\ have to do dma-sync after any changes to mbuf or mbuf-dma.
: selftest ( -- fail? )
    map-scsi
    mbuf-alloc
    scsi-selftest
    mbuf-free
    unmap-scsi
;

new-device \ missing "reg" indicates a SCSI "wild-card" node
    " sd"      encode-string " name" property
finish-device

new-device \ missing "reg" indicates a SCSI "wild-card" node
    " st"      encode-string " name" property
finish-device
end0
```

Complete Block and Byte Device Driver

Code Example 8-3 Sample Driver for Bootable Devices

```

\ This driver supports "block" and "byte" type bootable devices, by using standard
\ "deblocker" and "disk-label" packages.
fcode-version3
hex
headers

: copyright ( -- )
    ." Copyright 1992 - 1995 Sun Microsystems. All Rights Reserved" cr
;
h# 20.0000    constant scsi-offset
h# 40        constant /scsi
d# 25.000.000 constant clock-frequency

: identify-me ( -- )
    " SUNW,my-scsi" encode-string " name" property
    " scsi" device-type
    my-address scsi-offset + my-space /scsi reg

;
identify-me

h# 10.0000 constant dma-offset
h# 10      constant /dma
-1 instance value dma-chip

external
: decode-unit ( addr len -- low high )    decode-2int ;
: dma-alloc   ( size -- vaddr )           " dma-alloc" $call-parent ;
: dma-free    ( vaddr size -- )           " dma-free" $call-parent ;
: dma-map-in  ( vaddr size cache? -- devaddr ) " dma-map-in" $call-parent ;
: dma-map-out ( vaddr devaddr size -- )     " dma-map-out" $call-parent ;

\ Dma-sync could be dummy routine if parent device doesn't support.
: dma-sync ( virt-addr dev-addr size -- )
    " dma-sync" my-parent [' ] $call-method catch if
        2drop 2drop 2drop
    then
;

: map-in ( addr space size -- virt ) " map-in" $call-parent ;
: map-out ( virt size -- )           " map-out" $call-parent ;

```


Code Example 8-3 Sample Driver for Bootable Devices (Continued)

```

headers
\ variables/values for sending commands, mapping etc.
-1 instance value scsi-init-id
-1 instance value scsi-chip
-1 instance value mbuf
-1 instance value mbuf-dma
h# 20 constant /mbuf
...

\ mapping and allocation routines for scsi
: map-scsi-chip ( -- )
  my-address scsi-offset + my-space /scsi map-in to scsi-chip
;

: unmap-scsi-chip ( -- ) scsi-chip /scsi map-out -1 to scsi-chip ;

: map-scsi ( -- )
  map-scsi-chip
  \ allocate buffers etc. for "sense" command and status
  ...
;

: unmap-scsi ( -- )
  \ free buffers etc. for "sense" command and status
  ...
  unmap-scsi-chip
;

\ words related to scsi commands and register access.
...

: reset-my-scsi ( -- error? ) ... ;
: reset-bus ( -- error? ) ... ;

: init-n-test ( -- ok? ) ... ;
: get-buffers ( -- ) ... ;
: give-buffers ( -- ) ... ;
: scsi-selftest ( -- fail? ) ... ;

```

Code Example 8-3 Sample Driver for Bootable Devices (Continued)

```

d# 512 constant ublock
0 instance value /block
0 instance value /tapeblock
instance variable fixed-len?
...

external
: set-timeout ( n -- ) ... ;
: send-diagnostic ( -- error? )
    \ run diagnostics and return any error.
    ...
;

: device-present? ( lun target -- present? ) ... ;
: mode-sense ( -- true | block-size false ) ... ;
: read-capacity ( -- true | block-size false ) ... ;

\ Spin up a SCSI disk, coping with a possible wedged SCSI bus
: timed-spin ( target lun -- ) ... ;

: disk-r/w-blocks ( addr block# #blocks direction? -- #xfered )
    ... ( #xfered )
;

\ Execute "mode-sense" command. If failed, execute read-capacity command.
\ If this also failed, return d# 512 as the block size.
: disk-block-size ( -- n )
    mode-sense if read-capacity if d# 512 then then
    dup to /block
;

: tape-block-size ( -- n ) ... ;
: fixed-or-variable ( -- max-block fixed? ) ... ;
: tape-r/w-some ( addr block# #blks read? -- actual# error? ) ... ;

headers

: dma-open ( -- ) my-address dma-offset + my-space /dma map-in to dma-chip ;

: dma-close ( -- ) dma-chip /dma map-out -1 to dma-chip ;

```

Code Example 8-3 Sample Driver for Bootable Devices (Continued)

```

\ After any changes to mbuf by cpu or any changes to mbuf-dma by device, synchronize
\ changes by issuing " mbuf mbuf-dma /mbuf dma-sync "
: mbuf-alloc ( -- )
  /mbuf dma-alloc to mbuf
  mbuf /mbuf false dma-map-in to mbuf-dma
;

\ dma-map-out and dma-free mbuf-dma
: mbuf-free ( -- )
  mbuf mbuf-dma /mbuf dma-sync          \ redundant
  mbuf mbuf-dma /mbuf dma-map-out
  mbuf /mbuf dma-free
  -1 to mbuf
  -1 to mbuf-dma
;

external

\ external methods for scsi bus ( "SUNW,my-scsi" node)
: open ( -- okay? )
  dma-open
  " scsi-initiator-id" get-inherited-property 0= if
    decode-int to scsi-init-id
    2drop
    map-scsi
    init-n-test          ( ok? )
    dup if              ( true )
      get-buffers      ( true )
    else
      unmap-scsi dma-close ( false )
    then              ( success? )
  else
    ." Missing initiator id" cr false
    dma-close
  then              ( success? )
;

: close ( -- ) give-buffers unmap-scsi dma-close ;

: reset ( -- )
  dma-open map-scsi
  ...
  reset-my-scsi drop reset-bus drop

```

Code Example 8-3 Sample Driver for Bootable Devices (Continued)

```

    unmap-scsi dma-close
;

: selftest ( -- fail? )
    map-scsi
    mbuf-alloc
    scsi-selftest
    mbuf-free
    unmap-scsi
;

headers

\ start of child block device

new-device \ missing "reg" indicates SCSI "wild-card" node

    " sd" encode-string " name" property
    " block" device-type

    0 instance value offset-low
    0 instance value offset-high
    0 instance value label-package

\ The "disk-label" package interprets any partition information contained in
\ the disk label. The "load" method of the "block" device uses the load method
\ provided by "disk-label"
: init-label-package ( -- okay? )
    0 to offset-high 0 to offset-low
    my-args " disk-label" $open-package to label-package
    label-package if
        0 0 " offset" label-package $call-method
        to offset-high to offset-low
        true
    else
        ." Can't open disk label package" cr false
    then
;

    0 instance value deblocker
: init-deblocker ( -- okay? )
    " " " deblocker" $open-package to deblocker
    deblocker if

```

Code Example 8-3 Sample Driver for Bootable Devices (Continued)

```

        true
    else
        ." Can't open deblocker package"  cr  false
    then
;

: device-present? ( lun target -- present? )
    " device-present?" $call-parent
;
\ The following methods are needed for "block" device:
\ open, close, selftest, reset, read, write, load, seek, block-size,
\ max-transfer,read-blocks, write-blocks.
\ Carefully notice the relationship between the methods for the "block" device
\ and the methods pre-defined for "disk-label" and "deblocker"

external
\ external methods for "block" device ( "sd" node)

: spin-up ( -- ) my-unit " timed-spin" $call-parent ;

: open ( -- ok? )
my-unit device-present? 0= if false exit then
    spin-up          \ Start the disk if necessary

    init-deblocker 0= if false exit then
    init-label-package 0= if
        deblocker close-package false exit
    then
    true
;

: close ( -- )
    label-package close-package 0 to label-package
    deblocker close-package 0 to deblocker
;

: selftest ( -- fail? )
    my-unit device-present? if
        " send-diagnostic" $call-parent ( fail? )
    else
        true ( error )
    then
;

```

Code Example 8-3 Sample Driver for Bootable Devices (Continued)

```

: reset ( -- ) ... ;

\ The "deblocker" package assists in the implementation of byte-oriented read and
\ write methods for disks and tapes. The deblocker provides a layer of buffering
\ to implement a high level byte-oriented interface "on top of" a low-level
\ block-oriented interface.

\ The "seek", "read" and "write" methods of this block device use corresponding
\ methods provided by "deblocker"

\ In order to be able to use the "deblocker" package this device has to define the
\ following four methods, which the deblocker uses as its low-level interface
\ to the device:
\ 1) block-size, 2) max-transfer, 3) read-blocks and 4) write-blocks

: block-size ( -- n ) " disk-block-size" $call-parent ;
: max-transfer ( -- n ) block-size h# 40 * ;

: read-blocks ( addr block# #blocks -- #read )
  true " disk-r/w-blocks" $call-parent
;
: write-blocks ( addr block# #blocks -- #written )
  false " disk-r/w-blocks" $call-parent
;

: dma-alloc ( #bytes -- vadr ) " dma-alloc" $call-parent ;
: dma-free ( vadr #bytes -- ) " dma-free" $call-parent ;
: seek ( offset.low offset.high -- okay? )
  offset-low offset-high x+ " seek" deblocker $call-method
;
: read ( addr len -- actual-len ) " read" deblocker $call-method ;
: write ( addr len -- actual-len ) " write" deblocker $call-method ;
: load ( addr -- size ) " load" label-package $call-method ;

finish-device \ finishing "block" device "sd"

headers

\ start of child byte device

new-device \ missing "reg" indicates "wild-card" node
  " st" encode-string " name" property

```

Code Example 8-3 Sample Driver for Bootable Devices (Continued)

```

" byte" device-type

false instance value write-eof-mark?
instance variable file-mark?
true instance value scsi-tape-first-install

: scsi-tape-rewind      ( -- [[xstatbuf] f-hw] error? ) ... ;

: write-eof ( -- [[xstatbuf] f-hw] error? ) ... ;

0 instance value deblocker
: init-deblocker ( -- okay? )
  " " " deblocker" $open-package to deblocker
  deblocker if
    true
  else
    ." Can't open deblocker package" cr false
  then
;

: flush-deblocker ( -- )
  deblocker close-package init-deblocker drop
;

: fixed-or-variable ( -- max-block fixed? )
  " fixed-or-variable" $call-parent
;

: device-present? ( lun target -- present? )
  " device-present?" $call-parent
;

\ The following methods are needed for "byte" devices:
\ open, close, selftest, reset, read, write, load, seek, block-size,
\ max-transfer, read-blocks, write-blocks. Carefully notice the relationship
\ between the methods for "byte" devices and the methods pre-defined for the
\ standard deblocker package.

external
\ external methods for "byte" device ( "st" node)

```

Code Example 8-3 Sample Driver for Bootable Devices (Continued)

```

\ The "deblocator" package assists in the implementation of byte-oriented read
\ and write methods for disks and tapes. The deblocator provides a layer of
\ buffering to implement a high level byte-oriented interface "on top of" a
\ low-level block-oriented interface.

\ The "read" and "write" methods of this "byte" device use the corresponding
\ methods provided by the "deblocator"

\ In order to be able to use the "deblocator" package this device has to define the
\ following four methods which the deblocator uses as its low-level interface to
\ the device:
\ 1) block-size, 2) max-transfer, 3) read-blocks and 4) write-blocks
: block-size ( -- n ) " tape-block-size" $call-parent ;

: max-transfer ( -- n )
  fixed-or-variable ( max-block fixed? )
  if
    \ Use the largest multiple of /tapeblock that to <= h# fe00
    h# fe00 over / *
  then
;

: read-blocks ( addr block# #blocks -- #read )
  file-mark? @ 0= if
    true " tape-r/w-some" $call-parent file-mark? ! ( #read )
  else
    3drop 0
  then
;

: write-blocks ( addr block# #blocks -- #written )
  false " tape-r/w-some" $call-parent file-mark? !
;

: dma-alloc ( #bytes -- vaddr ) " dma-alloc" $call-parent ;

: dma-free ( vaddr #bytes -- ) " dma-free" $call-parent ;

: open ( -- okay? ) \ open for tape
  my-unit device-present? 0= if false exit then
  scsi-tape-first-install if
    scsi-tape-rewind if

```


Code Example 8-3 Sample Driver for Bootable Devices (Continued)

```

        ." Can't rewind tape" cr
        0= if drop then
            false exit
        then
            false to scsi-tape-first-install
        then
            \ Set fixed-len? and /tapeblock
            fixed-or-variable 2drop
            init-deblocker 0= if false exit then
            true
    ;

: close ( -- )
    deblocker close-package 0 to deblocker
    write-eof-mark? if
        write-eof if
            ." Can't write EOF Marker."
            0= if drop then
                then
            then
    ;

: reset ( -- ) ... ;
: selftest ( -- fail? )
    my-unit device-present? if
        " send-diagnostic" $call-parent ( fail? )
    else
        true ( error )
    then
;

: read ( addr len -- actual-len ) " read" deblocker $call-method ;
: write ( addr len -- actual-len )
    true to write-eof-mark?
    " write" deblocker $call-method
;

: load ( addr -- size )
    \ use my-args to get tape file-no
    ... ( addr file# )

    \ position at requested file
    ...

```

Code Example 8-3 Sample Driver for Bootable Devices (Continued)

```

dup begin
    dup max-transfer read      ( start-addr next-addr )
    dup 0>                    ( start-addr next-addr #read )
    dup 0>                    ( start-addr next-addr #read got-some? )
while                          ( start-addr next-addr #read )
    +                          ( start-addr next-addr' )
repeat                          ( start-addr end-addr 0 )
drop swap -                    ( size )
;


: seek ( byte# file# -- error? )
  \ position at requested file
  ...                          ( byte# )

  flush-deblocker              ( byte# )
begin dup 0> while              ( #remaining )
  " mbuf0" $call-parent
  over ublock min read         ( #remaining #read )
  dup 0= if                    ( #remaining 0 )
    2drop true
    exit                       ( error )
  then                          ( #remaining #read )
  -                              ( #remaining' )
repeat                          ( 0 )
drop false                      ( no-error )
;

  finish-device \ finishing "byte" device "st"
end0
\ finishing "SUNW,my-scsi"

```

Display Devices

9 

This chapter discusses writing FCode programs for display devices. The display device type applies to frame buffers and other devices appearing as memory to the processor, with associated hardware to convert the memory image to a visual display. Display devices can be used as console output devices.

Required Methods

The display device's FCode must declare the `display` device type, and must implement the methods `open` and `close`.

System defer words are loaded by appropriate routines. `is-install`, `is-remove` and `is-selftest` are used to create the `open`, `close` and `selftest` routines. `set-font` initializes the values of `frame-buffer-adr`, `char-height` and `char-width`, all of which are built into the system ROM and can be used by any display device driver.

For display devices, created methods interact with OpenBoot commands in a way that is different from that of other device types. Other device types provide methods that are found by dictionary searches looking for specific names.

Some FCodes are specifically designed for display devices. See Table A-36 through Table A-42 in Appendix , "FCode Reference".

Required Properties

Table 9-1 Required Display Device Properties

Property Name	Typical Value
name	SUNW,cgsix
reg	list of registers {device dependent}
device_type	" display" {required for display devices}

Device Driver Examples

Simple Display Device Driver

This is a sample FCode program for a display device that does not need to be usable as a console display device during system power-up.

Code Example 9-1 Basic Display Device Driver Example

```

\ cg6 (Lego) frame buffer driver
fcode-version3
hex

" SUNW,cgsix" name
" SUNW,501-xxxx" model

h# 20.0000 constant dac-offset h# 10 constant /dac
h# 30.0000 constant fhc-offset h# 10 constant /fhc
h# 30.1800 constant thc-offset h# 20 constant /thc
h# 70.0000 constant fbc-offset h# 10 constant /fbc
h# 70.1000 constant tec-offset h# 10 constant /tec
h# 80.0000 constant fb-offset h# 10.0000 constant /frame

: >reg-spec ( offset size -- encoded-reg )
  >r 0 my-address d+ my-space encode-phys 0 encode-int encode+ r> encode-int encode+
;

0 0 >reg-spec \ Configuration space registers
dac-offset /dac >reg-spec encode+
fhc-offset /fhc >reg-spec encode+
thc-offset /thc >reg-spec encode+

```

Code Example 9-1 Basic Display Device Driver Example (Continued)

```

fbc-offset /fbc >reg-spec    encode+
tec-offset /tec >reg-spec    encode+
fb-offset  /frame >reg-spec  encode+
" reg" property

end0

```

Extended Display Device Driver

This sample FCode program has additional functionality to initialize and test the device, but still is not usable as a console display device during system power-up.

Code Example 9-2 Extended Display Device Driver Example

```

\ cg6 (Lego) frame buffer driver
fcode-version3
hex

" SUNW,cgsix" name
" SUNW,501-xxxx" model

h# 20.0000 constant dac-offset h#      10 constant /dac
h# 30.0000 constant fhc-offset h#      10 constant /fhc
h# 30.1800 constant thc-offset h#      20 constant /thc
h# 70.0000 constant fbc-offset h#      10 constant /fbc
h# 70.1000 constant tec-offset h#      10 constant /tec
h# 80.0000 constant fb-offset  h# 10.0000 constant /frame

: >reg-spec ( offset size -- encoded-reg )
  >r 0 my-address d+ my-space encode-phys 0 encode-int encode+ r> encode-int encode+
;

0 0 >reg-spec                                \ Configuration space registers
dac-offset /dac >reg-spec    encode+
fhc-offset /fhc >reg-spec    encode+
thc-offset /thc >reg-spec    encode+
fbc-offset /fbc >reg-spec    encode+
tec-offset /tec >reg-spec    encode+
fb-offset  /frame >reg-spec  encode+
" reg" property

```

Code Example 9-2 Extended Display Device Driver Example (Continued)

```

-1 value dac-addr
-1 value fhc-addr
-1 value thc-addr
-1 value fbc-addr
-1 value tec-addr
-1 value fb-addr

: copyright ( -- addr len ) "Copyright (c) 1992 - 1996 Sun Microsystems, Inc. " ;
: do-map-in ( offset size -- )
  >r      ( offset )      ( R: size ) \ Move size to return stack
  0      ( offset 0 )    ( R: size ) \ Convert offset to double number
  my-address ( offset 0 phys.lo phys.mid ) ( R: size )
  d+      ( phys.lo' phys.mid )      ( R: size )
  my-space r> ( phys.lo' phys.mid phys.hi size ) ( R: )
  " map-in" $call-parent
;
: do-map-out ( vaddr size -- ) " map-out" $call-parent ;

: dac-map ( -- ) dac-offset /dac do-map-in to dac-addr ;
: dac-unmap ( -- ) dac-addr /dac do-map-out -1 to dac-addr ;

: fhc-map ( -- ) fhc-offset /fhc do-map-in to fhc-addr ;
: fhc-unmap ( -- ) fhc-addr /fhc do-map-out -1 to fhc-addr ;

: thc-map ( -- ) thc-offset /thc do-map-in to thc-addr ;
: thc-unmap ( -- ) thc-addr /thc do-map-out -1 to thc-addr ;

: fbc-map ( -- ) fbc-offset /fbc do-map-in to fbc-addr ;
: fbc-unmap ( -- ) fbc-addr /fbc do-map-out -1 to fbc-addr ;

: tec-map ( -- ) tec-offset /tec do-map-in to tec-addr ;
: tec-unmap ( -- ) tec-addr /tec do-map-out -1 to tec-addr ;

: fb-map ( -- ) fb-offset /frame do-map-in to fb-addr ;
: fb-unmap ( -- ) fb-addr /frame do-map-out -1 to fb-addr ;

: map-regs ( -- ) dac-map fhc-map thc-map fbc-map tec-map ;
: unmap-regs ( -- ) tec-unmap fbc-unmap thc-unmap fhc-unmap dac-unmap ;

\ Brooktree DAC interface section

\ The Brooktree DAC has an internal address register which helps to select the
\ internal register which is to be accessed. First, the address is written to

```

Code Example 9-2 Extended Display Device Driver Example (Continued)

```

\ register 0, then the data is written to one of the other registers. Ibis has 3
\ separate DAC chips which appear as the three least-significant bytes of a
\ longword. All three chips may be simultaneously updated with a single longword
\ write.

: dac! ( data reg# -- ) >r dup 2dup bljoin r> dac-addr + 1! ;
: dac-ctl! ( data int.addr reg# -- ) swap 0 dac! dac! ;

\ color! sets an overlay color register.
\ In order to be able to use either the Brooktree 457 or 458 dacs, we set the address
\ once, then store the color 3 times. The chip internally cycles each time the color
\ register is written, selecting in turn the red color, the green color, and the blue
\ color. The chip is used in "RGB mode".

: color! ( r g b c# -- )
  0 dac! ( r g b )
  swap rot ( b g r )
  4 dac! ( b g )
  4 dac! ( b )
  4 dac! ( )
;

: lego-init-dac ( -- )
  40 06 8 dac-ctl! \ Control reg: enable off, overlay off, RGB on
  0 05 8 dac-ctl! \ Blinking off
  ff 04 8 dac-ctl! \ Read mask set to all ones
  ff ff ff 0 color! \ White in overlay background color register
  0 0 0 ff color! \ Black in overlay foreground color register
  64 41 b4 1 color! \ SUN-blue for logo
;

\ End of Brooktree DAC code

\ Lego Selftest section

: fbc! ( value offset -- ) fbc-addr + 1! ;
: fbc@ ( offset -- value ) fbc-addr + 1@ ;
: tec! ( value offset -- ) tec-addr + 1! ;

: lego-selftest ( -- failed? ) false ;

\ Hardware configuration register section

```

Code Example 9-2 Extended Display Device Driver Example (Continued)

```

: fhc! ( value offset -- ) fhc-addr + 1! ;
: thc! ( value offset -- ) thc-addr + 1! ;

: set-res-params ( hcvd hcvs hchd hchsdrv hchs fhc-conf -- )
  0 fhc! 0 thc! 4 thc! 8 thc! c thc! 10 thc!
;

\ Resolution params:          hcvd      hcvs      hchd  hchsdrv  hchs  fhc-conf
: r1024x768 ( -- params ) 2c032c 32c0005 110051 490000 510007 3bb ;
: r1152x900 ( -- params ) 2403a8 10005 15005d 570000 10009 bbb ;
: r1024x1024 ( -- params ) 200426 10005 180054 520000 10009 3bb ;
: r1152x870 ( -- params ) 2c0392 20005 120054 540000 10009 bbb ;
: r1600x1280 ( -- params ) 340534 534009 130045 3d0000 450007 1bbb ;

0 value lego-rez-width
0 value lego-rez-height

0 value sense-code

: set-resolution ( sense-code -- )
  case
    0 of d# 1152 d# 900 endof
    12 of d# 1024 d# 1024 endof
    13 of d# 1600 d# 1280 endof
    drop d# 1152 d# 900 0
  endcase
  to lego-rez-height to lego-rez-width
;

8f value thc-misc
: lego-video-on ( -- ) thc-misc 400 or 18 thc! ;
: lego-video-off ( -- ) thc-misc 18 thc! ;

: lego-init-hc ( -- )
  sense-code case
    0 of r1152x900 endof
    12 of r1024x1024 endof
    13 of r1600x1280 endof
    drop r1152x900 0
  endcase
  ( resolution-params )
  set-res-params

```


Code Example 9-2 Extended Display Device Driver Example (Continued)

```

016b 14 thc!          \ THC_HCREFRESH
148f 18 thc!          \ THC_HCMISC
\ 48f 18 thc!        \ THC_HCMISC
lego-video-off       \ Turn video on at install time
;

\ End of hardware configuration register section

end0

```

Complete Display Device Drive

This sample FCode program is for a device that would be usable as a system console device.

Code Example 9-3 Complete Display Device Driver Example

```

\ cg6 (Lego) frame buffer driver
fcode-version3
hex

" SUNW,cgsix" name
" SUNW,501-xxxx" model
" display" device-type

h# 20.0000 constant dac-offset h# 10 constant /dac
h# 30.0000 constant fhc-offset h# 10 constant /fhc
h# 30.1800 constant thc-offset h# 20 constant /thc
h# 70.0000 constant fbc-offset h# 10 constant /fbc
h# 70.1000 constant tec-offset h# 10 constant /tec
h# 80.0000 constant fb-offset h# 10.0000 constant /frame

: >reg-spec ( offset size -- encoded-reg )
  >r 0 my-address d+ my-space encode-phys 0 encode-int encode+ r> encode-int encode+
;

0 0 >reg-spec \ Configuration space registers
dac-offset /dac >reg-spec encode+
fhc-offset /fhc >reg-spec encode+
thc-offset /thc >reg-spec encode+
fbc-offset /fbc >reg-spec encode+
tec-offset /tec >reg-spec encode+

```

Code Example 9-3 Complete Display Device Driver Example (Continued)

```

fb-offset /frame >reg-spec encode+
" reg" property

-1 value dac-addr
-1 value fhc-addr
-1 value thc-addr
-1 value fbc-addr
-1 value tec-addr
-1 value fb-addr

: copyright ( -- addr len ) "Copyright (c) 1992 - 1996 Sun Microsystems, Inc. " ;

: do-map-in ( offset size -- )
>r      ( offset )      ( R: size ) \ Move size to return stack
0       ( offset 0 )    ( R: size ) \ Convert offset to double number
my-address ( offset 0 phys.lo phys.mid ) ( R: size )
d+       ( phys.lo' phys.mid )      ( R: size )
my-space r> ( phys.lo' phys.mid phys.hi size ) ( R: )
" map-in" $call-parent
;

: do-map-out ( vaddr size -- ) " map-out" $call-parent ;
: dac-map ( -- ) dac-offset /dac do-map-in to dac-addr ;
: dac-unmap ( -- ) dac-addr /dac do-map-out -1 to dac-addr ;
: fhc-map ( -- ) fhc-offset /fhc do-map-in to fhc-addr ;
: fhc-unmap ( -- ) fhc-addr /fhc do-map-out -1 to fhc-addr ;

: thc-map ( -- ) thc-offset /thc do-map-in to thc-addr ;
: thc-unmap ( -- ) thc-addr /thc do-map-out -1 to thc-addr ;

: fbc-map ( -- ) fbc-offset /fbc do-map-in to fbc-addr ;
: fbc-unmap ( -- ) fbc-addr /fbc do-map-out -1 to fbc-addr ;

: tec-map ( -- ) tec-offset /tec do-map-in to tec-addr ;
: tec-unmap ( -- ) tec-addr /tec do-map-out -1 to tec-addr ;

: fb-map ( -- ) fb-offset /frame do-map-in to fb-addr ;
: fb-unmap ( -- ) fb-addr /frame do-map-out -1 to fb-addr ;

: map-regs ( -- ) dac-map fhc-map thc-map fbc-map tec-map ;
: unmap-regs ( -- ) tec-unmap fbc-unmap thc-unmap fhc-unmap dac-unmap ;

\ Brooktree DAC interface section

```

Code Example 9-3 Complete Display Device Driver Example (Continued)

```

\ The Brooktree DAC has an internal address register which helps to
\ select the internal register which is to be accessed.
\ First, the address is written to register 0, then the data is written
\ to one of the other registers.
\ Ibis has 3 separate DAC chips which appear as the three least-significant
\ bytes of a longword. All three chips may be simultaneously updated
\ with a single longword write.

: dac! ( data reg# -- ) >r dup 2dup bljoin r> dac-addr + 1! ;
: dac-ctl! ( data int.addr reg# -- ) swap 0 dac! dac! ;

\ color! sets an overlay color register.
\ In order to be able to use either the Brooktree 457 or 458 dacs, we
\ set the address once, then store the color 3 times. The chip internally
\ cycles each time the color register is written, selecting in turn the
\ red color, the green color, and the blue color.
\ The chip is used in "RGB mode".

: color! ( r g b c# -- )
  0 dac! ( r g b )
  swap rot ( b g r )
  4 dac! ( b g )
  4 dac! ( b )
  4 dac! ( )
;

: lego-init-dac ( -- )
  40 06 8 dac-ctl! \ Control reg: enable off, overlay off, RGB on
  0 05 8 dac-ctl! \ Blinking off
  ff 04 8 dac-ctl! \ Read mask set to all ones
  ff ff ff 0 color! \ White in overlay background color register
  0 0 0 ff color! \ Black in overlay foreground color register
  64 41 b4 1 color! \ SUN-blue for logo
;

\ End of Brooktree DAC code
\ Lego Selftest section

: fbc! ( value offset -- ) fbc-addr + 1! ;
: fbc@ ( offset -- value ) fbc-addr + 1@ ;
: tec! ( value offset -- ) tec-addr + 1! ;

```

Code Example 9-3 Complete Display Device Driver Example (Continued)

```

: lego-selftest ( -- failed? ) false ;

\ Hardware configuration register section

: fhc! ( value offset -- ) fhc-addr + 1! ;
: thc! ( value offset -- ) thc-addr + 1! ;

: set-res-params ( hcvd hcvs hchd hchsdrv hchs fhc-conf -- )
  0 fhc! 0 thc! 4 thc! 8 thc! c thc! 10 thc!
;

\ Resolution params:          hcvd      hcvs      hchd  hchsdrv  hchs  fhc-conf
: r1024x768 ( -- params ) 2c032c 32c0005 110051 490000 510007 3bb ;
: r1152x900 ( -- params ) 2403a8 10005 15005d 570000 10009 bbb ;
: r1024x1024 ( -- params ) 200426 10005 180054 520000 10009 3bb ;
: r1152x870 ( -- params ) 2c0392 20005 120054 540000 10009 bbb ;
: r1600x1280 ( -- params ) 340534 534009 130045 3d0000 450007 1bbb ;

0 value lego-rez-width
0 value lego-rez-height

0 value sense-code

: set-resolution ( sense-code -- )
  case
    0 of d# 1152 d# 900 endof
    12 of d# 1024 d# 1024 endof
    13 of d# 1600 d# 1280 endof
  drop d# 1152 d# 900 0
  endcase
  to lego-rez-height to lego-rez-width
;

8f value thc-misc
: lego-video-on ( -- ) thc-misc 400 or 18 thc! ;
: lego-video-off ( -- ) thc-misc 18 thc! ;
: lego-blink ( -- ) lego-video-off 20 ms lego-video-on ;
: lego-init-hc ( -- )
  sense-code case
    0 of r1152x900 endof
    12 of r1024x1024 endof
    13 of r1600x1280 endof

```

Code Example 9-3 Complete Display Device Driver Example (Continued)

```

        drop    r1152x900    0
    endcase          ( resolution-params )
    set-res-params
    016b 14 thc!      \ THC_HCREFRESH
    148f 18 thc!      \ THC_HCMISC
    lego-video-off    \ Turn video on at install time
;

\ End of hardware configuration register section

\ Lego graphics section
: lego-install ( -- )
    map-regs fb-map fb-addr to frame-buffer-adr

    default-font ( param ... ) set-font

    frame-buffer-adr encode-int " address" property

    lego-rez-width lego-rez-height over char-width / over char-height /
    fb8-install
    ['] lego-blink to blink-screen
    lego-video-on
;

: lego-remove ( -- )
    lego-video-off
    unmap-regs
    fb-unmap -1 to frame-buffer-adr
;

\ End of Lego graphics section

: lego-probe ( -- )

    map-regs

    sense-code set-resolution

    lego-init-dac
    lego-init-hc

    unmap-regs

```

≡ 9

Code Example 9-3 Complete Display Device Driver Example (Continued)

```
lego-rez-width  encode-int  " width"  property
lego-rez-height encode-int  " height" property
d# 8           encode-int  " depth"  property
lego-rez-width  encode-int  " linebytes" property

['] lego-install  is-install
['] lego-remove   is-remove
['] lego-selftest is-selftest
;

lego-probe

end0
```

Memory-Mapped Buses

10 

This chapter discusses addressing and required properties for memory-mapped buses.

A memory-mapped bus logically extends the processor's memory address space to include the devices on that bus. This enables the children of the bus device to be mapped into the CPU address space and accessed like memory using processor load and store cycles to address those children directly.

SBus and VMEbus are examples of memory-mapped buses.

Not all bus devices fall into this category. For example, SCSI is not a memory-mapped bus; SCSI targets are not accessed with load or store instructions.

Required Methods

A memory-mapped bus package code must implement the `open`, `close`, `reset`, and `selftest` methods, as well as the following:

```
decode-unit (addr len -- phys.lo ... phys.hi)
```

Convert `addr len`, a text string representation, to `phys.lo ... phys.hi`, a numerical representation of a physical address in the address space defined by this package. The format of `phys.lo ... phys.hi` varies from bus to bus.

dma-alloc (*size* -- *virt*)

Allocate a virtual address range of length *size* bytes that is suitable for direct memory access by a bus master device. The memory is allocated according to the most stringent alignment requirements for the bus. *virt* is a 32-bit address that the OpenBoot-based system can use to access the memory.

Note that `dma-map-in` must also be called to generate a suitable DMA address.

A child of a memory-mapped device calls `dma-alloc` using

```
" dma-alloc" $call-parent
```

For example:

```
-1 value my-vaddr
: my-dma-alloc ( size -- )
  " dma-alloc" $call-parent to my-vaddr
;
```

dma-free (*virt* *size* --)

Free *size* bytes of memory previously allocated by `dma-alloc` at the virtual address *virt*.

A child of a memory-mapped device calls `dma-free` by using

```
" dma-free" $call-parent
```

For example:

```
2000 value my-size
: my-dma-free ( -- )
  my-vaddr my-size " dma-free" $call-parent
  -1 to my-vaddr
;
```


`dma-map-in` (*virt size cacheable? -- devaddr*)

Convert the virtual address range `virt size`, previously allocated by `dma-alloc`, into an address `devaddr` suitable for DMA on the bus. `dma-map-in` can also be used to map application-supplied data buffers for DMA use if the bus allows. If `cacheable?` is true, the calling child desires to use any available fast caches for the DMA buffer. If access to the buffer is required before the buffer is mapped out, the child must call `dma-sync` or `dma-map-out` to ensure cache coherency with memory.

A child of a memory-mapped device calls `dma-map-in` using

```
" dma-map-in" $call-parent
```

For example:

```
: my-vaddr-dma-map ( -- )
  my-vaddr my-size false " dma-map-in" $call-parent ( devaddr )
  to my-vaddr-dma
;
```

`dma-map-out` (*virt devaddr size --*)

Remove the DMA mapping previously created with `dma-map-in`. Flush all caches associated with the mapping.

A child of a memory-mapped device calls `dma-map-out` by using

```
" dma-map-out" $call-parent
```

For example:

```
: my-vaddr-dma-free ( -- )
  my-vaddr my-vaddr-dma my-size " dma-map-out" $call-parent
  -1 to my-vaddr-dma
;
```

dma-sync (*virt devaddr size --*)

Synchronize (flush) any memory caches associated with the DMA mapping previously established by `dma-map-in`. You must interleave calls to this method (or `dma-map-out`) between DMA and CPU accesses to the memory region, or you may not obtain the most recent data written into the cache.

For example, a child of a hierarchical device calls `dma-sync` by using `$call-parent`. This method is valid for FCode version 2.1 or later. Some early version 2 systems do not define this method in the `/sbus` node. Those systems automatically synchronize DMA and CPU access. The following example will give correct results in all cases.

```
: my-dma-sync ( virt devaddr size -- )
  " dma-sync" $call-parent
;
```

probe-self (*arg-addr arg-len reg-addr reg-len fcode-addr fcode-len --*)

Probe for a child of this node. `fcode-addr fcode-len` is a unit-address text string that identifies the location of the FCode program for the child. `reg-addr reg-len` is a probe-address text string that identifies the address of the child itself. `arg-addr arg-len` is an instance-arguments text string for any device arguments for the child (which can be retrieved in the child's FCode program with the `my-args` FCode). `probe-self` checks whether there is indeed FCode at the indicated location, perhaps by mapping the device and using `cpeek` to ensure that the device is present and that the first byte is a valid FCode start byte.

If the FCode exists, `probe-self` creates a new child device node and interprets the FCode. If the interpretation of the FCode fails in some way, the new device node may be empty, containing no properties or methods.

For example, to probe FCode for SBus slot #1:

```
" /sbus" open-dev
0 0 " 1,0" 2dup probe-self
device-end
```

map-in (*phys.lo ... phys.hi size -- virt*)

Create a mapping associating the range of physical addresses beginning at `phys.lo ... phys.hi` extending for *size* bytes in the package's physical address space with a processor virtual address `virt`.

The number of cells in the list `phys.lo ... phys.hi` is determined by the value of the "#address-cells" property of the node containing `map-in`.

For example, a child of a memory-mapped device calls `map-in` with "`map-in`" `$call-parent`. (The following example assumes that the value of the parent's "#address-cells" property is 3):

```
: map-reg ( -- )
  my-address xx-offset 0 d+ my-space ( phys.lo phys.mid phys.hi )
  xx-size " map-in" $call-parent ( virt )
  to xx-vaddr ( )
;
```

map-out (*virt size --*)

Destroy the mapping set by `map-in` at virtual address `virt` of length *size* bytes. For example, a child of a memory-mapped device calls `map-out` with "`map-out`" `$call-parent`:

```
: unmap-reg ( -- )
  xx-vaddr xx-size ( virt size )
  " map-out" $call-parent ( )
  -1 to xx-vaddr
;
```

SBus Addressing

The SBus uses geographical addressing with numbered slots.

An SBus physical address is represented numerically by the SBus slot number as the *high* number and the offset from the base of that slot as the *low* number. The text string representation is *slot#*, *offset* where both *slot#* and *offset* are the ASCII representations of hexadecimal numbers.

SBus Required Properties

Table 10-1 SBus Required Properties

Property Name	Sample Value
name	" SUNW,fas"
burst-sizes	.
device_type	" sbus"
ranges	.
slot-address-bits	.

Device Driver Examples

The following examples of a hierarchical FCode driver are based on Sun's SBus expansion hardware, XBox. XBox increases the number of SBus slots available in a system by providing a bus-bridge between the platform's onboard SBus and an SBus in the XBox hardware. XBox includes an SBus card called the XAdaptor card which plugs into the host platform's SBus and includes an expansion chassis called the XBox Expansion Box. Therefore XBox is an example of a hierarchical device which implements an SBus interface to child plug-in devices.

The example is divided into three parts: the basic device driver, the extended device driver, and the complete device driver. In the case of a hierarchical device, in practice, one would only want to develop and ship a driver with the complete functionality. Otherwise, plug-in cards which rely on a full set of parent services generally would not be able to function. The three stage presentation of the driver simply shows how a driver might grow through the development cycle.

Basic Hierarchical Device Driver

The basic driver simply declares most of the important properties of the device, particularly the addresses of the various registers. A driver in this state might be used to support the development of the OS driver which would attach to the device name and configure itself based on the device properties published by the FCode driver.

Code Example 10-1 Basic Hierarchical Device Driver

```

hex
fcode-version3

" SUNW,xbox" name
" 501-1840" model

\ Xbox Registers
\ XAdaptor card registers
h# 0 constant write0-offset h# 4 constant /write0
h# 2.0000 constant xac-err-offset h# c constant /xac-err
h# 10.0000 constant xac-ctl0-offset h# 4 constant /xac-ctl0
h# 11.0000 constant xac-ctl1-offset h# 4 constant /xac-ctl1
h# 12.0000 constant xac-elua-offset h# 4 constant /xac-elua
h# 13.0000 constant xac-ella-offset h# 4 constant /xac-ella
h# 14.0000 constant xac-ele-offset h# 4 constant /xac-ele

\ Xbox Exapnsion box registers
h# 42.0000 constant xbc-err-offset h# c constant /xbc-err
h# 50.0000 constant xbc-ctl0-offset h# 4 constant /xbc-ctl0
h# 51.0000 constant xbc-ctl1-offset h# 4 constant /xbc-ctl1
h# 52.0000 constant xbc-elua-offset h# 4 constant /xbc-elua
h# 53.0000 constant xbc-ella-offset h# 4 constant /xbc-ella
h# 54.0000 constant xbc-ele-offset h# 4 constant /xbc-ele

: >reg-spec ( offset size -- xdrreg )
  >r my-address + my-space encode-phys r> encode-int encode+
;

write0-offset /write0 >reg-spec
xac-err-offset /xac-err >reg-spec encode+
xac-ctl0-offset /xac-ctl0 >reg-spec encode+
xac-ctl1-offset /xac-ctl1 >reg-spec encode+
xac-elua-offset /xac-elua >reg-spec encode+
xac-ella-offset /xac-ella >reg-spec encode+

```

≡ 10

Code Example 10-1 Basic Hierarchical Device Driver (Continued)

```
xac-ele-offset    /xac-ele    >reg-spec  encode+
xbc-err-offset   /xbc-err    >reg-spec  encode+
xbc-ctl0-offset  /xbc-ctl0   >reg-spec  encode+
xbc-ctl1-offset  /xbc-ctl1   >reg-spec  encode+
xbc-elua-offset  /xbc-elua   >reg-spec  encode+
xbc-ella-offset  /xbc-ella   >reg-spec  encode+
xbc-ele-offset   /xbc-ele    >reg-spec  encode+
" reg" property

\ Xbox can interrupt on any SBus level

1 encode-int      2 encode-int encode+ 3 encode-int encode+ 4 encode-int encode+
5 encode-int encode+ 6 encode-int encode+ 7 encode-int encode+
" interrupts" property

1 sbus-intr>cpu encode-int      0 encode-int encode+
2 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
3 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
4 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
5 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
6 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
7 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
" intr" property

\ Xbox bus clock speed
d# 25.000.000 encode-int " clock-frequency" property

\ Burst sizes 64,32,16,8,4,2,1 bursts.
h# 7f encode-int " burst-sizes" property

\ Xbox has no slave-only slots
0 encode-int " slave-only" property

\ Get the number of address bits for this SBus slot from the parent SBus
\ node without inheritance . OpenBoot 2.5 doesn't publish slot-address-bits.
\ However 2.5 is only on 4m machines, which are all 28 bits per slot.

: $= ( addr1 len1 addr2 len2 -- equal? )      \ string compare
  rot over - if
    drop 2drop false                          \ different lengths
  else comp 0=
  then
;
;
```

Code Example 10-1 Basic Hierarchical Device Driver (Continued)

```
: 4mhack ( -- n )
  " compatible" get-inherited-property if
    d# 25                                     \ no "compatible" prop; assume 4c
  else decodestring " sun4m" $= if
    d# 28
    else
      d# 25                                     \ not sun4m
    then
      nip nip
    then
;
: #bits ( -- n )
  " slot-address-bits" my-parent ihandle>phandle
  get-package-property if
    4mhack
  else
    decode-int nip nip
  then
;
#bits constant host-slot-size
host-slot-size encode-int " slot-address-bits" property
end0
```

Extended Hierarchical Device Driver

The extended driver adds methods allowing access to various device registers in addition to the functions of the basic driver. It provides methods to:

- Map in the registers
- Fetch from and store to the registers
- Program one of the registers which control the allocation of address space across the various SBus slots.

≡ 10

Such an extended driver provides methods that a developer can use to read and write registers and verify correct hardware responses. Note that the complete driver does not use all of the device registers; read/write access methods were included for all of them to allow easy testing during development.

Code Example 10-2 Extended Hierarchical Device Driver

```
hex
fcode-version3

" SUNW,xbox" name
" 501-1840" model

\ Xbox Registers

h#      0 constant write0-offset  h# 4 constant /write0
h#  2.0000 constant xac-err-offset  h# c constant /xac-err
h# 10.0000 constant xac-ctl0-offset  h# 4 constant /xac-ctl0
h# 11.0000 constant xac-ctl1-offset  h# 4 constant /xac-ctl1
h# 12.0000 constant xac-elua-offset  h# 4 constant /xac-elua
h# 13.0000 constant xac-ella-offset  h# 4 constant /xac-ella
h# 14.0000 constant xac-ele-offset  h# 4 constant /xac-ele

h# 42.0000 constant xbc-err-offset  h# c constant /xbc-err
h# 50.0000 constant xbc-ctl0-offset  h# 4 constant /xbc-ctl0
h# 51.0000 constant xbc-ctl1-offset  h# 4 constant /xbc-ctl1
h# 52.0000 constant xbc-elua-offset  h# 4 constant /xbc-elua
h# 53.0000 constant xbc-ella-offset  h# 4 constant /xbc-ella
h# 54.0000 constant xbc-ele-offset  h# 4 constant /xbc-ele

: >reg-spec ( offset size -- xdrreg )
  >r my-address + my-space encode-phys r> encode-int encode+
;

write0-offset    /write0    >reg-spec
xac-err-offset   /xac-err    >reg-spec  encode+
xac-ctl0-offset  /xac-ctl0   >reg-spec  encode+
xac-ctl1-offset  /xac-ctl1   >reg-spec  encode+
xac-elua-offset  /xac-elua   >reg-spec  encode+
xac-ella-offset  /xac-ella   >reg-spec  encode+
xac-ele-offset   /xac-ele    >reg-spec  encode+
xbc-err-offset   /xbc-err    >reg-spec  encode+
```


Code Example 10-2 Extended Hierarchical Device Driver (Continued)

```

xbc-ctl0-offset /xbc-ctl0 >reg-spec encode+
xbc-ctl1-offset /xbc-ctl1 >reg-spec encode+
xbc-elua-offset /xbc-elua >reg-spec encode+
xbc-ella-offset /xbc-ella >reg-spec encode+
xbc-ele-offset /xbc-ele >reg-spec encode+
" reg" property

\ Xbox can interrupt on any SBus level

1 encode-int      2 encode-int encode+ 3 encode-int encode+ 4 encode-int encode+
5 encode-int encode+ 6 encode-int encode+ 7 encode-int encode+
" interrupts" property

1 sbus-intr>cpu encode-int      0 encode-int encode+
2 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
3 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
4 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
5 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
6 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
7 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
" intr" property

\ Xbox bus clock speed
d# 25.000.000 encode-int " clock-frequency" property

\ Burst sizes 64,32,16,8,4,2,1 bursts.
h# 7f encode-int " burst-sizes" property

\ Xbox has no slave-only slots
0 encode-int " slave-only" property

\ Get the number of address bits for this SBus slot from the parent SBus
\ node without inheritance . OpenBoot 2.5 doesn't publish slot-address-bits.
\ However 2.5 is only on 4m machines, which are all 28 bits per slot.

: $= ( addr1 len1 addr2 len2 -- equal? )      \ string compare
  rot over - if
    drop 2drop false                          \ different lengths
  else comp 0=
  then
;
: 4mhack ( -- n )
  " compatible" get-inherited-property if

```

≡ 10

Code Example 10-2 Extended Hierarchical Device Driver (Continued)

```
    d# 25                                \ no "compatible" prop; assume 4c
else decodestring " sun4m" $= if
    d# 28
    else
        d# 25                            \ not sun4m
        then
            nip nip
        then
;
: #bits ( -- n )
  " slot-address-bits" my-parent ihandle>phandle
  get-package-property if
    4mhack
  else
    decode-int nip nip
  then
;
#bits constant host-slot-size
host-slot-size encode-int " slot-address-bits" property

\ Utility display string
: .me ( -- ) ." SBus " my-space .d ." Xbox " ;

\ The Xbox device has two modes opaque and transparent.

\ Upon reset the device is set to opaque mode. In this mode all
\ accesses to address space of the device are directed to the Xbox H/W
\ (ie. XAdaptor Card or the Xbox Expansion Box) itself.

\ In the transparent mode all accesses are mapped to the SBus cards
\ which are plugged into the Xbox. In transparent mode the Xbox H/W is
\ accessible only via the "write-0" register. To allow another bus
\ bridge to be plugged into the Xbox all writes to the write-0 register
\ must contain a "key" which is programmed into the Xbox H/W at boot
\ time. If the key field of a write to write-0 matches that programmed
\ at boot time the H/W intercepts the write. Otherwise the H/W passes
\ the write along.

\ The Xbox has two sets of registers. Those of the XAdaptor card and
\ and those of the Xbox Expansion Box.

\ Opaque mode host adapter registers
```

Code Example 10-2 Extended Hierarchical Device Driver (Continued)

```

-1 value xac-err-regs
-1 value xac-ctl0      -1 value xac-ctl1
-1 value xac-elua     -1 value xac-ella
-1 value xac-ele
\ Opaque mode expansion box registers
-1 value xbc-err-regs
-1 value xbc-ctl0     -1 value xbc-ctl1
-1 value xbc-elua     -1 value xbc-ella
-1 value xbc-ele
\ Transparent mode register
-1 value write0-reg

: xbox-map-in ( offset space size -- virt ) " map-in" $call-parent ;
: xbox-map-out ( virt size -- ) " map-out" $call-parent ;
: map-regs ( -- )
  write0-offset  my-address + my-space /write0  xbox-map-in  to write0-reg
  xac-err-offset  my-address + my-space /xac-err  xbox-map-in  to xac-err-regs
  xac-ctl0-offset my-address + my-space /xac-ctl0 xbox-map-in  to xac-ctl0
  xac-ctl1-offset my-address + my-space /xac-ctl1 xbox-map-in  to xac-ctl1
  xac-elua-offset my-address + my-space /xac-elua xbox-map-in  to xac-elua
  xac-ella-offset my-address + my-space /xac-ella xbox-map-in  to xac-ella
  xac-ele-offset  my-address + my-space /xac-ele  xbox-map-in  to xac-ele
  xbc-err-offset  my-address + my-space /xbc-err  xbox-map-in  to xbc-err-regs
  xbc-ctl0-offset my-address + my-space /xbc-ctl0 xbox-map-in  to xbc-ctl0
  xbc-ctl1-offset my-address + my-space /xbc-ctl1 xbox-map-in  to xbc-ctl1
  xbc-elua-offset my-address + my-space /xbc-elua xbox-map-in  to xbc-elua
  xbc-ella-offset my-address + my-space /xbc-ella xbox-map-in  to xbc-ella
  xbc-ele-offset  my-address + my-space /xbc-ele  xbox-map-in  to xbc-ele
;
: unmap-regs ( -- )
  write0-reg  /write0  xbox-map-out  -1 to write0-reg
  xac-err-regs /xac-err  xbox-map-out  -1 to xac-err-regs
  xac-ctl0     /xac-ctl0 xbox-map-out  -1 to xac-ctl0
  xac-ctl1     /xac-ctl1 xbox-map-out  -1 to xac-ctl1
  xac-elua     /xac-elua xbox-map-out  -1 to xac-elua
  xac-ella     /xac-ella xbox-map-out  -1 to xac-ella
  xac-ele      /xac-ele  xbox-map-out  -1 to xac-ele
  xbc-err-regs /xbc-err  xbox-map-out  -1 to xbc-err-regs
  xbc-ctl0     /xbc-ctl0 xbox-map-out  -1 to xbc-ctl0
  xbc-ctl1     /xbc-ctl1 xbox-map-out  -1 to xbc-ctl1
  xbc-elua     /xbc-elua xbox-map-out  -1 to xbc-elua
  xbc-ella     /xbc-ella xbox-map-out  -1 to xbc-ella
  xbc-ele      /xbc-ele  xbox-map-out  -1 to xbc-ele

```

Code Example 10-2 Extended Hierarchical Device Driver (Continued)

```

;

\ Opaque mode register access words

: xac-errd@ ( -- l ) xac-err-regs    rl@ ;
: xac-erra@ ( -- l ) xac-err-regs 4 + rl@ ;
: xac-errs@ ( -- l ) xac-err-regs 8 + rl@ ;
: xac-ctl0@ ( -- w ) xac-ctl0 rl@ ;
: xac-ctl0! ( w -- ) xac-ctl0 rl! ;
: xac-ctl1@ ( -- w ) xac-ctl1 rl@ ;
: xac-ctl1! ( w -- ) xac-ctl1 rl! ;
: xac-elua@ ( -- l ) xac-elua rl@ ;
: xac-elua! ( l -- ) xac-elua rl! ;
: xac-ella@ ( -- w ) xac-ella rl@ ;
: xac-ella! ( w -- ) xac-ella rl! ;

: xbc-errd@ ( -- l ) xbc-err-regs rl@ ;
: xbc-erra@ ( -- l ) xbc-err-regs 4 + rl@ ;
: xbc-errs@ ( -- l ) xbc-err-regs 8 + rl@ ;
: xbc-ctl0@ ( -- w ) xbc-ctl0 rl@ ;
: xbc-ctl0! ( w -- ) xbc-ctl0 rl! ;
: xbc-ctl1@ ( -- w ) xbc-ctl1 rl@ ;
: xbc-ctl1! ( w -- ) xbc-ctl1 rl! ;
: xbc-elua@ ( -- l ) xbc-elua rl@ ;
: xbc-elua! ( l -- ) xbc-elua rl! ;
: xbc-ella@ ( -- w ) xbc-ella rl@ ;
: xbc-ella! ( w -- ) xbc-ella rl! ;

\ Transparent Mode register access words

external
: unique-key ( -- n ) " unique-key" $call-parent ;
headers
unique-key constant my-key
my-key encode-int " write0-key" property

: xbox! ( w offset -- ) my-key h# 18 << or or write0-reg rl! ;

: write-xac-ctl0 ( w -- ) xac-ctl0-offset xbox! ;
: write-xac-ctl1 ( w -- ) xac-ctl1-offset xbox! ;
: write-xbc-ctl0 ( w -- ) xbc-ctl0-offset xbox! ;
: write-xbc-ctl1 ( w -- ) xbc-ctl1-offset xbox! ;

```

Code Example 10-2 Extended Hierarchical Device Driver (Continued)

```

\ Some functionally oriented words

: set-key      ( -- ) my-key 8 << xac-ctl0! ;
: transparent  ( -- )          1 xac-ctl1! ;
: opaque      ( -- )          0 write-xac-ctl1 ;
: enable-slaves ( -- )    h# 38 write-xbc-ctl1 ;

: xbox-errors ( -- xbc-err xac-err )
  opaque xbc-errd@ xac-errd@ transparent
;

: ?.errors ( xbc-err xac-err -- )
  dup h# 8000.0000 and if
    cr .me ." xac-error " .h cr
  else drop
  then
  dup h# 8000.0000 and if
    cr .me ." xbc-error " .h cr
  else drop
  then
;

\ The address space of the XBox in transparent mode may be dynamically
\ allocated across its plug-in slots. This is called the
\ upper-address-decode-map (uadm). Below is a table which relates the
\ slot configuration code which is programmed in hardware to the
\ allocation of address space for each slot. The number in each cell is
\ the number of address bits needed for the slot.

decimal
create slot-sizes-array
\ slot0 slot1 slot2 slot3    slot-config
23 c, 23 c, 23 c, 23 c,    \ 00
23 c, 23 c, 23 c, 23 c,    \ 01
23 c, 23 c, 23 c, 23 c,    \ 02
23 c, 23 c, 23 c, 23 c,    \ 03
25 c,  0 c,  0 c,  0 c,    \ 04
 0 c, 25 c,  0 c,  0 c,    \ 05
 0 c,  0 c, 25 c,  0 c,    \ 06
 0 c,  0 c,  0 c, 25 c,    \ 07
24 c, 24 c,  0 c,  0 c,    \ 08
24 c,  0 c, 24 c,  0 c,    \ 09
 0 c, 24 c, 24 c,  0 c,    \ 0a

```

Code Example 10-2 Extended Hierarchical Device Driver (Continued)

```

    0 c, 0 c, 0 c, 0 c,    \ 0b
    24 c, 23 c, 23 c, 0 c,    \ 0c
    23 c, 24 c, 23 c, 0 c,    \ 0d \ Overridden in code
    23 c, 23 c, 24 c, 0 c,    \ 0e \ Overridden in code
    25 c, 0 c, 0 c, 0 c,    \ 0f
    26 c, 26 c, 26 c, 26 c,    \ 10
    26 c, 26 c, 26 c, 26 c,    \ 11
    26 c, 26 c, 26 c, 26 c,    \ 12
    26 c, 26 c, 26 c, 26 c,    \ 13
    28 c, 0 c, 0 c, 0 c,    \ 14
    0 c, 28 c, 0 c, 0 c,    \ 15
    0 c, 0 c, 28 c, 0 c,    \ 16
    0 c, 0 c, 0 c, 28 c,    \ 17
    28 c, 28 c, 28 c, 28 c,    \ 18
    28 c, 28 c, 28 c, 28 c,    \ 19
    28 c, 28 c, 28 c, 28 c,    \ 1a
    28 c, 28 c, 28 c, 28 c,    \ 1b
    0 c, 0 c, 0 c, 0 c,    \ 1c
    0 c, 0 c, 0 c, 0 c,    \ 1d
    0 c, 0 c, 0 c, 0 c,    \ 1e
    0 c, 0 c, 0 c, 0 c,    \ 1f
hex
20 constant /slot-sizes-array
-1 value slot-config

: >slot-size ( slot# -- size )
  slot-sizes-array slot-config la+ swap ca+ c@ 1 swap <<
  1 not and    \ Could have slot size of 0.
;

\ This array is to be filled with offsets for each slot.
\ Eg. 0, 100.0000, 180.0000, 200.0000
create host-offsets 0 , 0 , 0 , 0 ,

: >host-offset ( child-slot# -- adr ) host-offsets swap na+ @ ;

create config-d-offsets h# 100.0000 , 0 , h# 180.0000 , 0 ,
create config-e-offsets h# 100.0000 , h# 180.0000 , 0 , 0 ,

: set-host-offsets ( -- )
  slot-config case
    h# d of config-d-offsets host-offsets 4 /n* move exit endof

```

Code Example 10-2 Extended Hierarchical Device Driver (Continued)

```
    h# e of config-e-offsets host-offsets 4 /n* move exit endof
endcase
0          ( initial-offset )
4 0 do    ( offset )
    dup host-offsets i na+ ! ( offset )
    i >slot-size +         ( offset' )
loop      ( final-offset )
drop
;

: set-configuration ( config-code -- )
  is slot-config
  set-host-offsets
  slot-config 3 << my-key 8 << or
  dup write-xac-ctl0 \ set XAC
    write-xbc-ctl0 \ set XBC
  slot-config encode-int " uadm" property \ publish slot configuration
;

end0
```

Complete Hierarchical Device Driver

The complete driver includes all the required device node methods. It also includes code to initialize the hardware at system reset. In particular, it configures the allocation of address space across slots. It does this by either performing an autoconfiguration or by accepting a manual override via a property in its parent. During the configuration process, the driver interprets the FCode of any SBus card plugged into the XBox. This results in devices being added to the device tree.

Code Example 10-3 Complete Hierarchical Device Driver

```
hex
fcode-version3

" SUNW,xbox" name
" 501-1840" model
" sbus" device-type
```

≡ 10

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```
\ Xbox Registers

h#      0 constant write0-offset   h# 4 constant /write0
h# 2.0000 constant xac-err-offset  h# c constant /xac-err
h# 10.0000 constant xac-ctl0-offset h# 4 constant /xac-ctl0
h# 11.0000 constant xac-ctl1-offset h# 4 constant /xac-ctl1
h# 12.0000 constant xac-elua-offset h# 4 constant /xac-elua
h# 13.0000 constant xac-ella-offset h# 4 constant /xac-ella
h# 14.0000 constant xac-ele-offset  h# 4 constant /xac-ele

h# 42.0000 constant xbc-err-offset  h# c constant /xbc-err
h# 50.0000 constant xbc-ctl0-offset  h# 4 constant /xbc-ctl0
h# 51.0000 constant xbc-ctl1-offset  h# 4 constant /xbc-ctl1
h# 52.0000 constant xbc-elua-offset  h# 4 constant /xbc-elua
h# 53.0000 constant xbc-ella-offset  h# 4 constant /xbc-ella
h# 54.0000 constant xbc-ele-offset   h# 4 constant /xbc-ele

: >reg-spec ( offset size -- xdrreg )
  >r my-address + my-space encode-phys r> encode-int encode+
;

write0-offset    /write0    >reg-spec
xac-err-offset   /xac-err    >reg-spec  encode+
xac-ctl0-offset  /xac-ctl0   >reg-spec  encode+
xac-ctl1-offset  /xac-ctl1   >reg-spec  encode+
xac-elua-offset  /xac-elua   >reg-spec  encode+
xac-ella-offset  /xac-ella   >reg-spec  encode+
xac-ele-offset   /xac-ele    >reg-spec  encode+
xbc-err-offset   /xbc-err    >reg-spec  encode+
xbc-ctl0-offset  /xbc-ctl0   >reg-spec  encode+
xbc-ctl1-offset  /xbc-ctl1   >reg-spec  encode+
xbc-elua-offset  /xbc-elua   >reg-spec  encode+
xbc-ella-offset  /xbc-ella   >reg-spec  encode+
xbc-ele-offset   /xbc-ele    >reg-spec  encode+
" reg" property

\ Xbox can interrupt on any SBus level

1 encode-int      2 encode-int encode+  3 encode-int encode+  4 encode-int encode+
5 encode-int encode+ 6 encode-int encode+ 7 encode-int encode+
" interrupts" property

1 sbus-intr>cpu encode-int      0 encode-int encode+
```


Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```
2 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
3 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
4 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
5 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
6 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
7 sbus-intr>cpu encode-int encode+ 0 encode-int encode+
" intr" property

\ Xbox bus clock speed
d# 25.000.000 encode-int " clock-frequency" property

\ Burst sizes 64,32,16,8,4,2,1 bursts.
h# 7f encode-int " burst-sizes" property

\ Xbox has no slave-only slots
0 encode-int " slave-only" property

\ Get the number of address bits for this SBus slot from the parent SBus
\ node without inheritance . OpenBoot 2.5 doesn't publish slot-address-bits.
\ However 2.5 is only on 4m machines, which are all 28 bits per slot.

: $= ( addr1 len1 addr2 len2 -- equal? )          \ string compare
  rot over - if
    drop 2drop false                             \ different lengths
  else comp 0=
  then
;
: 4mhack ( -- n )
  " compatible" get-inherited-property if
    d# 25                                         \ no "compatible" prop; assume 4c
  else decode-string " sun4m" $= if
    d# 28
  else
    d# 25                                         \ not sun4m
  then
  nip nip
then
;
: #bits ( -- n )
  " slot-address-bits" my-parent ihandle>phandle
  get-package-property if
    4mhack
  else
```

≡ 10

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```
        decode-int  nip nip
    then
;
#bits constant  host-slot-size
host-slot-size encode-int  " slot-address-bits" property

\ Utility display string
: .me ( -- ) ." SBus " my-space .d ." Xbox " ;

\ The Xbox device has two modes opaque and transparent.

\ Upon reset the device is set to opaque mode.  In this mode all
\ accesses to address space of the device are directed to the Xbox H/W
\ (ie. XAdaptor Card or the Xbox Expansion Box) itself.

\ In the transparent mode all accesses are mapped to the SBus cards
\ which are plugged into the Xbox.  In transparent mode the Xbox H/W is
\ accessible only via the "write-0" register.  To allow another bus
\ bridge to be plugged into the Xbox all writes to the write-0 register
\ must contain a "key" which is programmed into the Xbox H/W at boot
\ time.  If the key field of a write to write-0 matches that programmed
\ at boot time the H/W intercepts the write.  Otherwise the H/W passes
\ the write along.

\ The Xbox has two sets of registers.  Those of the XAdaptor card and
\ and those of the Xbox Expansion Box.

\ Opaque mode host adapter registers
-1 value xac-err-regs
-1 value xac-ctl0      -1 value xac-ctl1
-1 value xac-elua      -1 value xac-ella
-1 value xac-ele
\ Opaque mode expansion box registers
-1 value xbc-err-regs
-1 value xbc-ctl0      -1 value xbc-ctl1
-1 value xbc-elua      -1 value xbc-ella
-1 value xbc-ele
\ Transparent mode register
-1 value write0-reg

: xbox-map-in ( offset space size -- virt) " map-in" $call-parent ;
: xbox-map-out ( virt size -- ) " map-out" $call-parent ;
```

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```

: map-regs ( -- )
  write0-offset  my-address + my-space /write0  xbox-map-in  to write0-reg
  xac-err-offset my-address + my-space /xac-err  xbox-map-in  to xac-err-regs
  xac-ctl0-offset my-address + my-space /xac-ctl0 xbox-map-in  to xac-ctl0
  xac-ctl1-offset my-address + my-space /xac-ctl1 xbox-map-in  to xac-ctl1
  xac-elua-offset my-address + my-space /xac-elua xbox-map-in  to xac-elua
  xac-ella-offset my-address + my-space /xac-ella xbox-map-in  to xac-ella
  xac-ele-offset my-address + my-space /xac-ele  xbox-map-in  to xac-ele
  xbc-err-offset my-address + my-space /xbc-err  xbox-map-in  to xbc-err-regs
  xbc-ctl0-offset my-address + my-space /xbc-ctl0 xbox-map-in  to xbc-ctl0
  xbc-ctl1-offset my-address + my-space /xbc-ctl1 xbox-map-in  to xbc-ctl1
  xbc-elua-offset my-address + my-space /xbc-elua xbox-map-in  to xbc-elua
  xbc-ella-offset my-address + my-space /xbc-ella xbox-map-in  to xbc-ella
  xbc-ele-offset my-address + my-space /xbc-ele  xbox-map-in  to xbc-ele
;
: unmap-regs ( -- )
  write0-reg  /write0  xbox-map-out  -1 to write0-reg
  xac-err-regs /xac-err  xbox-map-out  -1 to xac-err-regs
  xac-ctl0     /xac-ctl0 xbox-map-out  -1 to xac-ctl0
  xac-ctl1     /xac-ctl1 xbox-map-out  -1 to xac-ctl1
  xac-elua     /xac-elua xbox-map-out  -1 to xac-elua
  xac-ella     /xac-ella xbox-map-out  -1 to xac-ella
  xac-ele      /xac-ele  xbox-map-out  -1 to xac-ele
  xbc-err-regs /xbc-err  xbox-map-out  -1 to xbc-err-regs
  xbc-ctl0     /xbc-ctl0 xbox-map-out  -1 to xbc-ctl0
  xbc-ctl1     /xbc-ctl1 xbox-map-out  -1 to xbc-ctl1
  xbc-elua     /xbc-elua xbox-map-out  -1 to xbc-elua
  xbc-ella     /xbc-ella xbox-map-out  -1 to xbc-ella
  xbc-ele      /xbc-ele  xbox-map-out  -1 to xbc-ele
;

\ Opaque mode register access words

: xac-errd@ ( -- l ) xac-err-regs  rl@ ;
: xac-erra@ ( -- l ) xac-err-regs 4 + rl@ ;
: xac-errs@ ( -- l ) xac-err-regs 8 + rl@ ;
: xac-ctl0@ ( -- w ) xac-ctl0  rl@ ;
: xac-ctl0! ( w -- ) xac-ctl0  rl! ;
: xac-ctl1@ ( -- w ) xac-ctl1  rl@ ;
: xac-ctl1! ( w -- ) xac-ctl1  rl! ;
: xac-elua@ ( -- l ) xac-elua  rl@ ;
: xac-elua! ( l -- ) xac-elua  rl! ;
: xac-ella@ ( -- w ) xac-ella  rl@ ;

```

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```

: xac-ella! ( w -- ) xac-ella rl! ;

: xbc-errd@ ( -- l ) xbc-err-regs rl@ ;
: xbc-erra@ ( -- l ) xbc-err-regs 4 + rl@ ;
: xbc-errs@ ( -- l ) xbc-err-regs 8 + rl@ ;
: xbc-ctl0@ ( -- w ) xbc-ctl0 rl@ ;
: xbc-ctl0! ( w -- ) xbc-ctl0 rl! ;
: xbc-ctl1@ ( -- w ) xbc-ctl1 rl@ ;
: xbc-ctl1! ( w -- ) xbc-ctl1 rl! ;
: xbc-elua@ ( -- l ) xbc-elua rl@ ;
: xbc-elua! ( l -- ) xbc-elua rl! ;
: xbc-ella@ ( -- w ) xbc-ella rl@ ;
: xbc-ella! ( w -- ) xbc-ella rl! ;

\ Transparent Mode register access words

external
: unique-key ( -- n ) " unique-key" $call-parent ;
headers
unique-key constant my-key
my-key encode-int " write0-key" property

: xbox! ( w offset -- ) my-key h# 18 << or or write0-reg rl! ;

: write-xac-ctl0 ( w -- ) xac-ctl0-offset xbox! ;
: write-xac-ctl1 ( w -- ) xac-ctl1-offset xbox! ;
: write-xbc-ctl0 ( w -- ) xbc-ctl0-offset xbox! ;
: write-xbc-ctl1 ( w -- ) xbc-ctl1-offset xbox! ;

\ Some functionally oriented words

: set-key ( -- ) my-key 8 << xac-ctl0! ;
: transparent ( -- ) 1 xac-ctl1! ;
: opaque ( -- ) 0 write-xac-ctl1 ;
: enable-slaves ( -- ) h# 38 write-xbc-ctl1 ;

: xbox-errors ( -- xbc-err xac-err )
opaque xbc-errd@ xac-errd@ transparent
;

: ?.errors ( xbc-err xac-err -- )
dup h# 8000.0000 and if
cr .me ." xac-error " .h cr

```

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```

else drop
then
dup h# 8000.0000 and if
  cr .me ." xbc-error " .h cr
else drop
then
;

\ The address space of the Xbox in transparent mode may be dynamically
\ allocated across its plug-in slots. This is called the
\ upper-address-decode-map (uadm). Below is a table which relates the
\ slot configuration code which is programmed in hardware to the
\ allocation of address space for each slot. The number in each cell is
\ the number of address bits needed for the slot.

decimal
create slot-sizes-array
\ slot0 slot1 slot2 slot3      slot-config
23 c, 23 c, 23 c, 23 c,      \ 00
23 c, 23 c, 23 c, 23 c,      \ 01
23 c, 23 c, 23 c, 23 c,      \ 02
23 c, 23 c, 23 c, 23 c,      \ 03
25 c, 0 c, 0 c, 0 c,         \ 04
0 c, 25 c, 0 c, 0 c,         \ 05
0 c, 0 c, 25 c, 0 c,         \ 06
0 c, 0 c, 0 c, 25 c,         \ 07
24 c, 24 c, 0 c, 0 c,         \ 08
24 c, 0 c, 24 c, 0 c,         \ 09
0 c, 24 c, 24 c, 0 c,         \ 0a
0 c, 0 c, 0 c, 0 c,         \ 0b
24 c, 23 c, 23 c, 0 c,         \ 0c
23 c, 24 c, 23 c, 0 c,         \ 0d \ Overridden in code
23 c, 23 c, 24 c, 0 c,         \ 0e \ Overridden in code
25 c, 0 c, 0 c, 0 c,         \ 0f
26 c, 26 c, 26 c, 26 c,         \ 10
26 c, 26 c, 26 c, 26 c,         \ 11
26 c, 26 c, 26 c, 26 c,         \ 12
26 c, 26 c, 26 c, 26 c,         \ 13
28 c, 0 c, 0 c, 0 c,         \ 14
0 c, 28 c, 0 c, 0 c,         \ 15
0 c, 0 c, 28 c, 0 c,         \ 16
0 c, 0 c, 0 c, 28 c,         \ 17
28 c, 28 c, 28 c, 28 c,         \ 18

```

≡ 10

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```
28 c, 28 c, 28 c, 28 c,    \ 19
28 c, 28 c, 28 c, 28 c,    \ 1a
28 c, 28 c, 28 c, 28 c,    \ 1b
 0 c,  0 c,  0 c,  0 c,    \ 1c
 0 c,  0 c,  0 c,  0 c,    \ 1d
 0 c,  0 c,  0 c,  0 c,    \ 1e
 0 c,  0 c,  0 c,  0 c,    \ 1f
hex
20 constant /slot-sizes-array
-1 value slot-config

: >slot-size ( slot# -- size )
  slot-sizes-array slot-config la+ swap ca+ c@ 1 swap <<
  1 not and          \ Could have slot size of 0.
;

\ This array is to be filled with offsets for each slot.
\ Eg. 0, 100.0000, 180.0000, 200.0000
create host-offsets 0 , 0 , 0 , 0 ,

: >host-offset ( child-slot# -- adr ) host-offsets swap na+ @ ;

create config-d-offsets h# 100.0000 , 0          , h# 180.0000 , 0 ,
create config-e-offsets h# 100.0000 , h# 180.0000 , 0          , 0 ,

: set-host-offsets ( -- )
  slot-config case
    h# d of config-d-offsets host-offsets 4 /n* move exit endof
    h# e of config-e-offsets host-offsets 4 /n* move exit endof
  endcase
  0                ( initial-offset )
  4 0 do           ( offset )
    dup host-offsets i na+ ! ( offset )
    i >slot-size + ( offset' )
  loop             ( final-offset )
  drop
;

: set-configuration ( config-code -- )
  is slot-config
  set-host-offsets
  slot-config 3 << my-key 8 << or
```

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```

dup write-xac-ctl0          \ set XAC
   write-xbc-ctl0          \ set XBC
slot-config encode-int " uadm" property \ publish slot configuration
;

\ Required package methods

external

: dma-alloc  ( #bytes -- )          " dma-alloc" $call-parent ;
: dma-free   ( #bytes -- )          " dma-free" $call-parent ;
: dma-map-in ( vaddr #bytes cache? -- devaddr ) " dma-map-in" $call-parent ;
: dma-map-out ( vaddr devaddr #bytes -- )      " dma-map-out" $call-parent ;
: dma-sync   ( virt devaddr #bytes -- )        " dma-sync" $call-parent ;

: map-in ( offset slot# size -- virtual )
  >r
  >host-offset + my-space ( offset xbox-slot# )
  r> " map-in" $call-parent ( parent-offset parent-slot# )
  ( virtual )
;

: map-out ( virt size -- ) " map-out" $call-parent ;

: decode-unit ( adr len -- address space )
  decode-2int ( offset slot# )
  dup 0 3 between 0= if
    ." Invalid XBox slot number " .d cr
    1 abort
  then
  ( offset slot# )
;

\ Hack because set-args and byte-load are not FCodes
: byte-load ( adr len -- ) " byte-load" $find drop execute ;
: set-args ( adr len adr len -- ) " set-args" $find drop execute ;

: probe-self ( arg-adr arg-len reg-adr reg-len fcode-adr fcode-len -- )

  ['] decode-unit catch if
    2drop 2drop 2drop 2drop
    exit
  then
  ( arg-str reg-str fcode-offs,space )

  h# 10000 map-in ( arg-str reg-str fcode-vaddr )

```

≡ 10

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```
dup cpeek if ( arg-str reg-str fcode-vaddr byte )
  dup h# f0 = swap h# fd = or if ( arg-str reg-str fcode-vaddr )
    new-device ( arg-str reg-str fcode-vaddr )
      >r set-args r> ( fcode-vaddr )
      dup 1 byte-load ( fcode-vaddr )
    finish-device
  else ( arg-str reg-str fcode-vaddr )
    nip nip nip nip ( fcode-vaddr )
    ." Invalid FCode start byte in " .me cr
  then ( fcode-vaddr )
else ( arg-str reg-str fcode-vaddr )
  nip nip nip nip ( fcode-vaddr )
then

h# 10000 map-out
;

: open ( -- ok? ) true ;
: close ( -- ) ;

headers

\ The Xbox slot configuration may be forced by the user. The mechanism
\ for doing this is a string which specifies megs/slot (eg. "16,8,8,0").

\ This string is processed into the config bits array. Then the
\ slot-sizes-array is searched for a configuration which matches or
\ exceeds the requested number for each slot. If the request is
\ unreasonable the default-slot-config is used.
\ Then the configuration is set in the Xbox hardware.
\ Finally each slot is probed based on the config.

: default-slot-config ( -- n )
  host-slot-size d# 25 = if
    h# c \ 1x24 bits, 2x23 bits
  else h# 10 \ 4x26 bits
  then
;

\ This array to be filled with bit sizes for each slot.
\ Eg. 24, 23, 23, 0
create config-bits 0 c, 0 c, 0 c, 0 c,
```


Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```
: config-ok? ( config -- ok? )
  true
  slot-sizes-array rot 4 * ca+      ( ok? slot-adr )
  4 0 do
    config-bits i ca+ c@
    over i ca+ c@                  ( ok? slot-adr conf-bits slot-bits )
    > if
      nip false swap leave
    then
  loop
  drop
;

: fit-config ( -- config )
  default-slot-config
  /slot-sizes-array 0 do
    i config-ok? if
      drop i leave
    then
  loop
;

: megs>bits ( megs -- bits )      \ Convert requested megs to # of address bits
  ?dup 0= if 0 exit then
  dup 9 < if drop d# 23 exit then
  dup d# 17 < if drop d# 24 exit then
  dup d# 33 < if drop d# 25 exit then
  dup d# 65 < if drop d# 26 exit then
  dup d# 129 < if drop d# 27 exit then
  dup d# 257 < if d# 28 exit then
  d# 29 \ d#29 is too many bits => error
;

: request-megs ( adr len -- )      \ Fill config-bits table
  base @ >r decimal
  4 0 do
    ascii , left-parse-string
    $number 0= if
      megs>bits config-bits i ca+ c!
    then
  loop
  2drop
```

≡ 10

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```
r> base !
;

: find-config ( adr len -- config )
  request-megs fit-config
;

create slot-string  ascii # c, ascii , c,  ascii 0 c,

: probe-slot ( slot# -- )
  dup >slot-size 0= if drop exit then ( slot# )
  ascii 0 + slot-string c!
  " " slot-string 3 ( arg-str reg-str )
  2dup ( arg-str reg-str fcode-str )
  probe-self
;

: probe-children ( -- )
  4 0 do
    config-bits i ca+ c@ if
      i probe-slot
    then
  loop
;

: forced-configuration ( adr len -- )
  find-config ( config-code )
  set-configuration
  probe-children
;

\ The Xbox slot configuration may be autoconfigured by the driver. The
\ autoconfiguration mechanism uses the following state transition table.
\ The table basically loops through each Xbox slot with a current guess
\ at the slot config. With each slot the code then probes the slot's
\ FCode and uses the reg property information of the slot's new device
\ node to determine the amount of address space required by the slot.
\ The slot config guess is updated and a state transition is made.

\ This is the state transition table. Each entry in the table consists
\ of 16 bits. The most significant 8 bits is the Xbox configuration
\ code for the next state, and the least 8 bits is the next state.
```

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```

create states
\ Empty      min      mid
\ Empty      23      24      for 25 bit host SBus slot
  0501 w, 0d04 w, 0803 w, \ 0          testing slot 0
  0602 w, 0a05 w, 0a0f w, \ 1 Slot 0 empty, testing slot 1
  0706 w, 000f w, 060e w, \ 2 Slots 0,1 empty, testing slot 2
  090f w, 0c0f w, 080e w, \ 3 Slot 0 is 24 bit, testing slot 1
  0e05 w, 0e05 w, 0d0f w, \ 4 Slot 0 23 bit, testing slot 1
  000f w, 000f w, 0e0e w, \ 5 Slot 0 empty and Slot1 23 bit,
                        \ or Slot 0,1 are 23 bit testing slot 2
  0c0e w, 070e w, 070e w, \ 6 Slots 0,1,2 empty, testing slot 3
\ Empty      notused  26      for 28 bit host SBus slot
  1508 w, 100e w, 100b w, \ 7          testing slot 0
  1609 w, 100e w, 100c w, \ 8 Slot 0 empty, testing slot 1
  170a w, 100e w, 100d w, \ 9 Slots 0,1 empty, testing slot 2
  100e w, 100e w, 170e w, \ a Slots 0,1,2 empty, testing slot 3
  100c w, 100e w, 100c w, \ b Slot 0 is 26 bit, testing slot 1
  100d w, 100e w, 100d w, \ c Slots 0,1 are 26 bit, testing slot 2
  100e w, 100e w, 100e w, \ d Slots 0,1,2 are 26 bit, testing slot 3
                        \ e
                        \ f

0          value slot#
0          value start-state      \ for auto-config state machine
4          value start-config

h# 100.0000 value max-card          \ 25 bit default
h# 080.0000 value mid-card          \ 25 bit default

: configure25 ( -- )                \ 25 bit host SBus slots
  0          is start-state
  4          is start-config
  h# 100.0000 is max-card            \ 25 bits for one Xbox slot
  h# 080.0000 is mid-card            \ 24 bits per Xbox slot
;

: configure28 ( -- )                \ 28 bit host SBus slots
  7          is start-state
  h# 14      is start-config
  h# 800.0000 is max-card            \ 28 bits for one Xbox slot
  h# 0       is mid-card             \ 26 bits per Xbox slot
;

0 value child-node

```

≡ 10

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```
\ Since child and peer do not appear until 2.3,
\ we include the following workarounds.
: next-peer ( phandle -- phandle' )
  fcode-version 2.0003 >= if
    peer
  else
    " romvec" $find drop execute 1c + @ 0 + @
    " call" $find drop execute nip
  then
;
: first-child ( phandle -- phandle' )
  fcode-version 2.0003 >= if
    child
  else
    " romvec" $find drop execute 1c + @ 4 + @
    " call" $find drop execute nip
  then
;

0 value extent          \ 1 if card exists, but no reg prop or 0 reg

: bump-extent ( n -- ) extent max is extent ;

: max-reg-extent ( adr len -- )
  begin dup while
    decode-int drop decode-int >r decode-int r> + ( adr' len' extent)
    bump-extent
  repeat
  2drop
  extent 0= if          \ reg prop is 0 -- fake it
    1 bump-extent
  then
;

: find-extent ( -- )
  0 is extent
  begin
    child-node if
      child-node next-peer
    else
      my-self ihandle>phandle first-child
    then
      ( next-child )
```

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```

?dup while
  is child-node
  " reg" child-node get-package-property 0= if ( adr len )
    max-reg-extent
  else \ card has no reg prop -- fake it
    1 bump-extent
  then
  repeat
;

: evaluate-size ( -- size-code )
  find-extent
  extent slot# >slot-size > if
    ." The card in slot " slot# .
    ." of " .me
    ." uses too much address space." cr
    abort
  then
  extent ( max-extent )
  dup max-card > if drop 3 exit then ( max-extent ) \ max-size card
  dup mid-card > if drop 2 exit then ( max-extent ) \ mid-size card?
  0 > if 1 exit then ( ) \ 25-small card?
  0 \ null for 28
;

: test-slot ( xbox-config -- size-code )
  set-configuration ( )
  slot# probe-slot ( )
  evaluate-size ( size-code )
;

: autoconfigure ( -- )
  0 is child-node
  -1 is slot#

  host-slot-size d# 25 = if configure25 else configure28 then

  start-state start-config ( state# xbox-config )
  begin ( state# xbox-config )
    slot# 1+ is slot# test-slot ( state# size-code )
    dup 3 = if 2drop exit then ( state# size-code )
    over h# f = if 2drop exit then ( state# size-code )
    states rot 3 * wa+ swap wa+ w@ wbsplit ( state#' xbox-config' )

```

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```

over h# e = until                                ( state#' xbox-config' )

2drop
;

\ Initialize the Xbox H/W.  If the XAdaptor H/W detects that Xbox
\ Expansion H/W is connected and powered-up it puts the H/W into
\ transparent mode and sets the Xbox slot configuraton based on either a
\ forced configuraton or the autoconfiguration algorithm.

: configuration ( -- )
  " xbox-slot-config" get-inherited-property 0= if
    decodestring ( adr len adr len )
    find-config forced-configuration
    2drop
  else
    2drop
    autoconfigure
  then
;

: null-xdr ( -- adr len )
  fcode-version 2.0001 >= if
    0 0 encodebytes
  else
    here 0
  then
;

: make-ranges ( -- )
  null-xdr ( adr len )
  4 0 do
    i >slot-size if ( adr len )
      0 i encode-phys encode+ ( adr len )
    i >host-offset my-space encode-phys encode+ ( adr len )
    i >slot-size encode-int encode+ ( adr len )
  then
  loop
  " ranges" property
;

\ Because we go transparent in the middle and therefore the fcode prom
\ disappears the following must be in a definition.

```

Code Example 10-3 Complete Hierarchical Device Driver (Continued)

```
: init-pkg ( -- )
  map-regs
  set-key
  xac-errs@ h# 40 and if \ opaque already
    transparent \ Child ready?
    enable-slaves \ Go transparent, then enable-slaves
  configuration
  make-ranges
  xbox-errors
  ?.errors
  " true"
else
  cr .me
  ." child not ready --" cr
  ." perhaps the cable is not plugged in" cr
  ." or the expansion box is not turned on." cr
  " false"
then
  ( adr len )
  encodestring " child-present" property
  unmap-regs
  ['] end0 execute
;

init-pkg

end0
```


This chapter describes how to implement network device drivers.

Network devices are packet-oriented devices capable of sending and receiving packets addressed according to IEEE 802.2 (Ethernet). OpenBoot firmware typically uses network devices for diskless booting. The standard `/obp-tftp` support package assists in the implementation of the `load` method for this device type.

Normally the network device driver would have a one level tree or a two level tree. The user can create a multi-level tree by applying `new-device` and `finish-device`.

A one level tree could have several nodes, depending on how many net channels the plug-in card can support, each node corresponds to one net channel.

This chapter shows three sample network device drivers for the Quad Ethernet device card. The device tree structure for the examples is as follows:

Each QED SBus card defines two levels:

- *one qec device node*
- *four qe device nodes*

sbus (or sbi on sun4d such as SS2000 and SC1000)

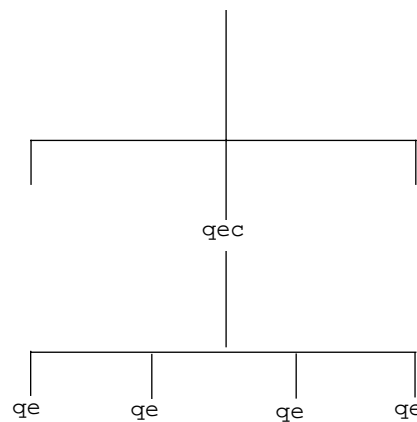


Figure 11-1 QED Device Tree

The general pathname (after sbus or sbi) for a qe node is

```
qec@S,20000/qe@C,0
```

where S is the SBus slot number and C is the network channel number.

Required Methods

The network device FCode must declare the `network device-type`, and must implement the methods `open` and `close`, as well as the following methods:

`load (addr -- len)`

Read the default stand-alone program into memory starting at *addr* using the default network booting protocol. *len* is the size in bytes of the program read in.

read (*addr len -- actual*)

Receive a network packet, placing at most the first *len* bytes in memory at *addr*. Return the *actual* number of bytes received (not the number copied), or -2 if no packet is currently available. Packets with hardware-detected errors are discarded as though they were not received. Do not wait for a packet (non-blocking).

write (*addr len -- actual*)

Transmit the network packet of size *len* bytes starting at memory address *addr*. Return the number of bytes actually transmitted. The packet must be complete with all addressing information, including source hardware address.

Required Device Properties

The required properties for a `network` device are:

Table 11-1 Required Network Device Properties

Name	Typical Value
<code>name</code>	" SUNW,my-net"
<code>reg</code>	list of registers {device-dependent}
<code>device_type</code>	" network"
<code>mac-address</code>	8 0 20 0c ea 41 {the MAC address currently being used.}

Optional Device Properties

Several other properties may be declared for `network` devices:

Table 11-2 Optional Network Device Properties

Property Name	Typical Property Value
<code>max-frame-size</code>	0x4000
<code>address-bits</code>	48
<code>slave-burst-sizes</code>	0x7f {depends on the number of entries in the <code>reg</code> property}
<code>local-mac-address</code>	8 0 20 ef 45 44 {the built-in Media Access Control address.}

Device Driver Examples

Simple Network Device Example

At minimum, a network device driver need only provide the desired tree structure and to publish all the necessary properties to identify the devices.

Code Example 11-1 QED Identification ROM Sample

```
\ qed-idrom.fth

fcode-version3

fload board.fth
headers
: copyright ( -- )
  ." Two-level QED-IDROM 1.1 " cr
  ." Copyright 1992 - 1998 Sun Microsystems, Inc.. All Rights Reserved" cr
;

: identify-qed ( -- )
  create-qec-properties
  4 0 do
    new-device
    i create-qe-properties
    finish-device
  loop
;
identify-qed

end0

\ -----
\ board.fth
\ To define required properties for QED devices.

headers
my-address    constant my-sbus-addr
my-space      constant my-sbus-space
headerless
```

Code Example 11-1 QED Identification ROM Sample (Continued)

```

\ Define the address map.
\ MED Address Map PA[18:0] (totally 512KB address space).
\ h# 00.0000    constant eprom-pa
\ h# 00.8000    constant /eprom                \ 32KB used, 64KB total
  h# 01.0000    constant mace-regs-offset
  h# 01.0000    constant mace0-base
  h# 01.4000    constant mace1-base
  h# 01.8000    constant mace2-base
  h# 01.c000    constant mace3-base
  h# 00.4000    constant /mace-regs            \ 16KB per channel, 64KB total
  h# 02.0000    constant global-regs-offset
  h# 01.0000    constant /global-regs          \ 64KB total
  h# 03.0000    constant channel-regs-offset
  h# 03.0000    constant channel0-base
  h# 03.4000    constant channel1-base
  h# 03.8000    constant channel2-base
  h# 03.c000    constant channel3-base
  h# 00.4000    constant /channel-regs         \ 16KB per channel, 64KB total
  h# 04.0000    constant locmem-pa
  h# 01.0000    constant /locmem              \ 64KB used, 256KB total

\ Real size of mace/qec-global/qec-channel registers.
  20 constant /qec-mace-regs
  14 constant /qec-global-regs
  34 constant /qec-channel-regs

\ Miscellaneous constant definitions.
  1          constant #channels
  h# 4000    constant max-frame-size          ( d# 1536 for le )
  d# 48      constant address-bits

\ Hardwired SBus interrupt level for MED.
  4 constant sbus-qe-intr

: encode-reg ( addr space size -- addr len ) >r encode-phys r> encode-int encode+ ;

: encode-ranges ( offs bustype phys offset size -- addr len )
  >r >r >r encode-phys r> r> r> encode-reg encode+
;

: offset>physical-addr ( offset -- paddr.lo paddr.hi )
  my-sbus-addr + my-sbus-space
;

headers

```

Code Example 11-1 QED Identification ROM Sample (Continued)

```

: create-qec-properties ( -- )
  " qec"      name
  " SUNW,595-3198" encode-string " model" property \ 595-3198-01
  global-regs-offset offset>physical-addr /global-regs encode-reg
  locmem-pa  offset>physical-addr /locmem encode-reg encode+
  " reg" property

  0 0 channel0-base  offset>physical-addr /channel-regs encode-ranges
  0 1 channel1-base  offset>physical-addr /channel-regs encode-ranges encode+
  0 2 channel2-base  offset>physical-addr /channel-regs encode-ranges encode+
  0 3 channel3-base  offset>physical-addr /channel-regs encode-ranges encode+
  0 h# 10 mace0-base offset>physical-addr /mace-regs  encode-ranges encode+
  0 h# 11 mace1-base offset>physical-addr /mace-regs  encode-ranges encode+
  0 h# 12 mace2-base offset>physical-addr /mace-regs  encode-ranges encode+
  0 h# 13 mace3-base offset>physical-addr /mace-regs  encode-ranges encode+
  " ranges" property

  #channels encode-int " #channels" property
  \ One interrupt per qec, not one interrupt per channel.
  sbus-qe-intr encode-int " interrupts" property
;
: create-qe-properties ( chan# -- )
  >r
  " qe" encode-string " name"  property
  r@ encode-int " channel#" property
  max-frame-size encode-int " max-frame-size" property
  address-bits encode-int " address-bits" property
  0 r@ /channel-regs encode-reg
    0 r@ h# 10 + /mace-regs encode-reg encode+
  " reg" property
  r> drop
;

```

Sample Driver With Test and Debugging Methods

This version of a network device driver is still non-bootable, but it shows how an intermediate step of driver can be used to debug and test the device during or after development.

The coding techniques shown in this and the following examples are:

- Each `qe` node has the same set of instance variables as each of the other `qe` nodes.
- All the `qe` nodes use the same `qe` driver source code defined in the first `qe` node (`qe0`).

Code Example 11-2 QED Test ROM Sample

```
\ qed-test.fth

fcode-version3
  headers
  fload board.fth
  : copyright ( -- )
    ." QED-TEST 1.1 " cr
    ." Copyright 1992 - 1998 Sun Microsystems, Inc.. All Rights Reserved" cr
  ;
: instance ( -- ) fcode-revision 20001 >= if instance then ;
\ Create qec device node.
  create-qec-properties
  fload qec-test.fth \ qec test code.

\ Create qe0 device node.
  new-device
    0 create-qe-properties
    : dma-sync ( virt-addr dev-addr size -- ) " dma-sync" $call-parent ;

    \ ***** qe0 instance variables *****
    0 instance value mace \ virtual address of Mace registers base
    0 instance value qecc \ virtual address of Qec channel registers base
    instance variable my-channel# \ qe channel#
    my-channel# off
    fload qe-test.fth \ qe test code.

    \ ***** qe0 external methods *****
    external
    : selftest ( -- fail? )
      qe0-selftest
    ;
    : open ( -- okay? )
      qe0-open
    ;
    : close ( -- )
```

Code Example 11-2 QED Test ROM Sample (Continued)

```
        qe0-close
    ;
    : reset ( -- )
        qe0-reset
    ;
    headers
finish-device

\ Create qe1 device node.
new-device
    1 create-qe-properties

    \ ***** qe1 instance variables *****
    0 instance value mace \ virtual address of Mace registers base
    0 instance value qecc \ virtual address of Qec channel registers base
    instance variable my-channel# \ qe channel#
    my-channel# off

    \ ***** qe1 external methods *****
    external
    : selftest ( -- fail? )
        qe0-selftest
    ;
    : open ( -- okay? )
        qe0-open
    ;
    : close ( -- )
        qe0-close
    ;
    : reset ( -- )
        qe0-reset
    ;
    headers
finish-device

\ Create qe2 device node.
new-device
    2 create-qe-properties

    \ ***** qe2 instance variables *****
    0 instance value mace \ virtual address of Mace registers base
    0 instance value qecc \ virtual address of Qec channel registers base
    instance variable my-channel# \ qe channel#
```


Code Example 11-2 QED Test ROM Sample (Continued)

```
my-channel# off

\ ***** qe2 external methods *****
external
: selftest ( -- fail? )
    qe0-selftest
;
: open ( -- okay? )
    qe0-open
;
: close ( -- )
    qe0-close
;
: reset ( -- )
    qe0-reset
;
headers
finish-device

\ Create qe3 device node.
new-device
    3 create-qe-properties

\ ***** qe3 instance variables *****
0 instance value mace \ virtual address of Mace registers base
0 instance value qecc \ virtual address of Qec channel registers base
instance variable my-channel# \ qe channel#
my-channel# off

\ ***** qe3 external methods *****
external
: selftest ( -- fail? )
    qe0-selftest
;
: open ( -- okay? )
    qe0-open
;
: close ( -- )
    qe0-close
;
: reset ( -- )
    qe0-reset
;
```

Code Example 11-2 QED Test ROM Sample (Continued)

```

    headers
    finish-device

end0

\ -----
\ gec-test.fth
\ Test code for the gec node.

/locmem #channels / value chmem
chmem 2/ value rxbufsize

\ ***** qed utility (from qed-util.fth) *****

: lwrt-rd-cmp ( mask data addr -- success? )
  2dup rl! rl@ rot and =
;
: cwrt-rd-cmp ( mask data addr -- success? )
  2dup rb! rb@ rot and =
;
instance defer wrt-rd-cmp
' lwrt-rd-cmp to wrt-rd-cmp
d# 32 instance value #bits

external
: wlk-test ( mask addr #bits -- success? )
  dup to #bits
  d# 32 = if ['] lwrt-rd-cmp else ['] cwrt-rd-cmp then to wrt-rd-cmp
  true -rot ( true mask addr )
  #bits 0
  do ( flag0 mask addr )
    over 1 i lshift and ?dup if ( flag0 mask addr data )
      >r 2dup r> swap wrt-rd-cmp false = ( flag0 mask addr flag )
      if rot drop false -rot leave then
    then
  loop
  2drop
;

headers
instance variable ms-timeout

external

```

Code Example 11-2 QED Test ROM Sample (Continued)

```
: set-ms-timeout ( #ms -- ) ms-timeout ! ;
: ms-timeout? ( -- flag )
  ms-timeout @ dup if
    1- ms-timeout ! 1 ms false
  else
    drop true
  then
;
headers

\ ***** qec global register (from global.h.fth) *****
\
\ QEC Global register set.
\
\ Virtual addresses of QEC global registers.
\ The actual addresses will be assigned later.
0 instance value qecg

hex
\ global control register (RW)
: qecg-control ( -- vaddr ) qecg ;
: qecg-control@ ( -- data ) qecg-control rl@ ;
: qecg-control! ( data -- ) qecg-control rl! ;

headerless
\ For Global Control Register.
f000.0000 constant gcr-mode \ Mode mask
4000.0000 constant gcr-mace \ Mace mode
1 constant gcr-reset \ Reset bit (0), 1 to enable reset.

headers

\ ***** qec map (from qecmap.fth ) *****

0 instance value locmem-base
false value dma-sync?
0 value dma-sync-addr

: find-dma-sync ( -- )
  " dma-sync" my-parent ihandle>phandle find-method if
    true to dma-sync?
    to dma-sync-addr
```

Code Example 11-2 QED Test ROM Sample (Continued)

```
    then
;
find-dma-sync

external
: decode-unit ( addr len -- address space ) decode-2int ;
: map-in ( offset slot# #bytes -- virtual ) " map-in" $call-parent ;
: map-out ( addr len -- ) " map-out" $call-parent ;
: dma-map-in ( vaddr n cache? -- devaddr ) " dma-map-in" $call-parent ;
: dma-map-out ( vaddr devaddr n -- ) " dma-map-out" $call-parent ;
: dma-alloc ( size -- addr ) " dma-alloc" $call-parent ;
: dma-free ( addr size -- ) " dma-free" $call-parent ;

\ Dma-sync could be dummy routine if parent device doesn't support.
\ sun4c ROMs may not support it.
: dma-sync ( virt-addr dev-addr size -- )
    dma-sync? if
        dma-sync-addr my-parent call-package
    else
        3drop
    then
;

headers

: map-qec-regs ( -- )
    global-regs-offset my-sbus-addr + my-sbus-space /qec-global-regs
    " map-in" $call-parent to qecg
;
: unmap-qec-regs ( -- )
    qecg /qec-global-regs " map-out" $call-parent
    0 to qecg
;

: map-locmem ( -- )
    locmem-pa my-sbus-addr + my-sbus-space /locmem
    " map-in" $call-parent to locmem-base
;
: unmap-locmem ( -- )
    locmem-base /locmem " map-out" $call-parent
    0 to locmem-base
;
```

Code Example 11-2 QED Test ROM Sample (Continued)

```
\ ***** qec test (from qectest.fth) *****
hex

headerless
\ 18 constant /qec-global-regs
\ Define the mask bits that can be tested for each global register.
create gl-reg-masks
    0000.001e , 0000.0000 , 0000.0000 , 0001.e000 ,
    0000.f000 , 0000.f000 ,

\ Test Qec global registers.
: gl-reg-test ( -- success? )
    true
    /qec-global-regs 0 do                ( flag0 )
        gl-reg-masks i + @
        qecg i + d# 32 wlk-test          ( flag0 flag )
        false = if drop false leave then ( flag0 )
    /n +loop
;

\ Perform register test for the qec node.
: qec-reg-test ( -- success? )
    diagnostic-mode? if
        ." Qec register test -- "
    then
        gl-reg-test
    diagnostic-mode? if
        dup if ." succeeded." else ." failed." then cr
    then
;

headers

\ ***** qec package *****

: reset-qec-global ( -- fail? )
    gcr-reset qecg-control!           \ Issue global reset.
    d# 100 set-ms-timeout
    begin
        qecg-control@ gcr-reset and
    while
        ms-timeout? if ." Global reset failed" cr true exit then
```

Code Example 11-2 QED Test ROM Sample (Continued)

```
repeat
false
;
: identify-chip ( -- okay? )
  qecg-control@ gcr-mode and gcr-mace =
;

external
: open ( -- true )
  map-qec-regs
  identify-chip dup 0= if
    unmap-qec-regs
  then
;
: close ( -- )
  qecg if unmap-qec-regs then
;

: selftest ( -- fail? )
  qecg ( qecg )
  map-qec-regs
  qec-reg-test ( qecg success? )
  unmap-qec-regs
  swap to qecg ( success? )
  0= ( fail? )
;

: reset ( -- )
  qecg
  map-qec-regs
  reset-qec-global drop
  unmap-qec-regs
  to qecg
;

headers
\ -----
\ qe-test.fth
\ Test code for the qe node.

: wlk-test ( mask addr #bits -- success? ) " wlk-test" $call-parent ;
: set-ms-timeout ( #ms -- ) " set-ms-timeout" $call-parent ;
```

Code Example 11-2 QED Test ROM Sample (Continued)

```
: ms-timeout? ( -- flag ) " ms-timeout?" $call-parent ;

\ ***** qe map (from qemap.fth) *****

headers
\ instance variable my-channel# my-channel# off
: my-channel#! ( channel# -- ) my-channel# ! ;

: my-chan# ( -- channel# )
  my-channel# @
;

: mace-regs ( -- devaddr space size )
  my-sbus-addr mace-regs-offset + /mace-regs my-chan# * +
  my-sbus-space /qec-mace-regs
;

: map-mace ( -- )
  mace-regs " map-in" my-parent $call-method to mace
;

: unmap-mace ( -- )
  mace /qec-mace-regs " map-out" my-parent $call-method
  0 to mace
;

: channel-regs ( -- devaddr space size )
  my-sbus-addr channel-regs-offset + /channel-regs my-chan# * +
  my-sbus-space /qec-channel-regs
;

: map-channel ( -- )
  channel-regs " map-in" my-parent $call-method to qecc
;

: unmap-channel ( -- )
  qecc /qec-channel-regs " map-out" my-parent $call-method
  0 to qecc
;

: map-chips ( -- )
  mace 0= if \ Do mapping if it is unmapped.
    map-mace
    map-channel
  then
;

;
```

Code Example 11-2 QED Test ROM Sample (Continued)

```
: unmap-chips ( -- )
  mace if          \ Do unmapping if it is mapped.
    unmap-channel
    unmap-mace
  then
;

\ ***** qe test (from qeregstst.fth) *****

hex

\ Define the mask bits that can be tested for each register.
create ch-reg-masks
  0000.0004 , 0000.0000 , ffff.f800 , ffff.f800 ,
  0000.0001 , 0000.0001 , 001f.001f , 1fc0.3fc0 ,
  0000.ffff , 0000.ffff , 0000.ffff , 0000.ffff ,
  0000.00ff ,
create mace-reg-masks
  00 c, 00 c, 89 c, 00 c, 00 c, 0d c, 00 c, 00 c,
  00 c, 67 c, 00 c, 70 c, f3 c, ef c, 04 c, 5f c,
  00 c, 00 c, 00 c, 00 c, 00 c, 00 c, 00 c, 00 c,
  00 c, 00 c, 00 c, 00 c, 00 c, 00 c, 00 c, 00 c,

\ Test Qec per channel registers.
: ch-reg-test ( -- flag )
  true
  /qec-channel-regs 0 do          ( flag0 )
    ch-reg-masks i + @
    qecc i + d# 32 wlk-test      ( flag0 flag )
    false = if drop false leave then ( flag0 )
  /n +loop
;

\ Test Mace registers.
: mace-reg-test ( -- flag )
  true
  /qec-mace-regs 0 do          ( flag0 )
    mace-reg-masks i + c@
    mace i + 8 wlk-test        ( flag0 flag )
    false = if drop false leave then ( flag0 )
  loop
;

\ Perform register test for the qe node.
```


Code Example 11-2 QED Test ROM Sample (Continued)

```
: qe-reg-test ( -- success? )
  diagnostic-mode? if
    ." Qe register test -- "
  then
    ch-reg-test
    mace-reg-test and
    diagnostic-mode? if
      dup if ." succeeded." else ." failed." then cr
    then
  ;

\ ***** qe0 package *****

headerless
\ For MACE BIU Configuration Control (R11). (RW)
01 constant m-swrst          \ software reset
: mace-biucc ( -- vaddr ) h# 0b mace + ;
: mace-biucc@ ( -- data ) mace-biucc rb@ ;
: mace-biucc! ( data -- ) mace-biucc rb! ;
\ For QEC per channel control reg. (RW)
02 constant c-rst
: qecc-control ( -- vaddr ) qecc ;
: qecc-control@ ( -- data ) qecc-control rl@ ;
: qecc-control! ( data -- ) qecc-control rl! ;

headers
: set-my-channel# ( -- )
\ If don't find the channel property, use 0.
  " channel#" get-my-property if 0 else decode-int nip nip then
  my-channel#!
;
\ Reset (or stop) the qec channel.
\   Issue a soft reset to the desired Mace.
\   Then issue a soft reset to the desired channel in QEC.
\ Chip reset algorithm:
\   Set the reset bit then wait until the reset bit cleared.
\ Timeout in 0.1 sec if fail.
\
: channel-reset ( -- fail? )
  m-swrst mace-biucc!          \ Issue Mace reset.
  d# 100 set-ms-timeout
  begin
    mace-biucc@ m-swrst and
```

Code Example 11-2 QED Test ROM Sample (Continued)

```
while
  ms-timeout? if ." Cannot reset Mace" cr true exit then
repeat
  c-rst qecc-control!          \ Reset QEC channel registers.
  d# 100 set-ms-timeout
begin
  qecc-control@ c-rst and
while
  ms-timeout? if ." Cannot reset QEC channel" cr true exit then
repeat
  false
;

external
: qe0-selftest ( -- flag )    \ Flag 0 if passes test.
  set-my-channel#
  map-chips
  qe-reg-test                ( success? )
  unmap-chips
  0=                          ( fail? )
;

: qe0-open ( -- okay? )
  set-my-channel#
  mac-address drop 6 encode-string " mac-address" property
  true
;

: qe0-close ( -- )
;

: qe0-reset ( -- )
  set-my-channel#
  map-chips channel-reset drop unmap-chips
;

headers
```

Bootable Network Device Driver Example

The example below shows a complete version of a bootable network driver. It implements the `selftest` method callable by OpenBoot `test` commands and the `watch-net` method callable by OpenBoot `watch-net` and `watch-net-all` commands.

Code Example 11-3 QED Bootable Driver Sample

```
\ qed.fth

fcode-version3
  headers
  fload board.fth
  : copyright ( -- )
    ." QED 1.1 " cr
    ." Copyright 1992-1998 Sun Microsystems, Inc. All Rights Reserved" cr
  ;
  : instance ( -- ) fcode-revision 20001 >=
    if instance then
  ;
\ Create qec device node.
  create-qec-properties
  fload qec.fth          \ qec driver.

\ Create qe0 device node.
  new-device
    0 create-qe-properties
    " network" device-type
    fload qeinstance.fth \ qe instance variables.
    : dma-sync ( virt-addr dev-addr size -- ) " dma-sync" $call-parent ;
    fload qe.fth        \ qe driver.
    fload qe-package.fth \ qe external methods.
  finish-device

\ Create qe1 device node.
  new-device
    1 create-qe-properties
    " network" device-type
    fload qeinstance.fth \ qe instance variables.
    fload qe-package.fth \ qe external methods.
  finish-device

\ Create qe2 device node.
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
new-device
  2 create-qe-properties
  " network" device-type
  fload qeinstance.fth      \ qe instance variables.
  fload qe-package.fth     \ qe external methods.
finish-device

\ Create qe3 device node.
new-device
  3 create-qe-properties
  " network" device-type
  fload qeinstance.fth     \ qe instance variables.
  fload qe-package.fth     \ qe external methods.
finish-device
end0

\ -----
\ qec.fth

/locmem #channels / value chmem
chmem 2/ value rxbufsize

fload qed-util.fth        \ Not included, refer to example 2.
fload global.h.fth        \ Not included.
fload qecmap.fth          \ Not included, refer to example 2.
fload qectest.fth         \ Not included, refer to example 2.

: reset-qec-global ( -- fail? )
  gcr-reset qecg-control! \ Issue global reset.
  d# 100 set-ms-timeout
  begin
    qecg-control@ gcr-reset and
  while
    ms-timeout? if ." Global reset failed" cr true exit then
  repeat
    false
;

: qec-init ( -- )
  chmem qecg-memsize!
  rxbufsize qecg-rxsize!
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
chmem rxbufsize - qecg-txsize!
gcr-burst16 qecg-control! \ SBus parity disabled, Rx/Tx equal priority.
;
: identify-chip ( -- okay? ) qecg-control@ gcr-mode and gcr-mace = ;

external
: open ( -- true )
  map-qec-regs
  identify-chip dup if
    qec-init
  else unmap-qec-regs
  then
;
: close ( -- ) qecg if unmap-qec-regs then ;

: selftest ( -- fail? )
  qecg ( qecg )
  map-qec-regs
  qec-reg-test ( qecg success? )
  unmap-qec-regs
  swap to qecg ( success? )
  0= ( fail? )
;

: reset ( -- )
  qecg
  map-qec-regs
  reset-qec-global drop
  unmap-qec-regs
  to qecg
;

headers
\ -----
\ qeinstance.fth
\ Define instance words for qe driver.

\ headerless
\ mace.h.fth:
0 instance value mace \ virtual address of Mace registers base
\ channel.h.fth:
0 instance value qecc \ virtual address of Qec channel registers base
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```

\ gemap.fth:
instance variable my-channel# \ qe channel#
    my-channel# off
\ qecore.fth:
\ CPU base address of tmd, rmd, tbuf, rbuf rings.
0 instance value cpu-dma-base \ base address of dma memory object viewed by cpu
0 instance value tmd0 \ transmit message descriptor#0
0 instance value rmd0 \ receive message descriptor#0
0 instance value tbuf0 \ base address of transmit buffer
0 instance value rbuf0 \ base address of receive buffers
\ IO (or dvice) base address of tmd, rmd, tbuf, rbuf rings.
0 instance value io-dma-base \ base addr of dma memory object viewed by device
0 instance value io-tmd0 \ transmit message descriptor#0
0 instance value io-rmd0 \ receive message descriptor#0
0 instance value io-tbuf0 \ base address of transmit buffer
0 instance value io-rbuf0 \ base address of receive buffers
\ Required total Dma buffer size for all rings.
0 instance value qe-dma-size \ Amount of memory mapped
\ *** Define required variables ***
instance variable status \ Accumulated channel status word.
instance variable restart? \ Restart? flag on after serious error.
instance variable nextrmd \ Point to next rmd.
instance variable nexttmd \ tmd0 nexttmd !, never changes presently
instance variable mode \ To store loopback control & promiscuous info.
6 instance buffer: this-en-addr \ Contain ethernet address
instance defer .receive-error
instance defer .error
instance defer .transmit-error
\ timed-receive.fth:
instance variable alarmtime
instance defer handle-broadcast-packet
\ qetest.fth:
instance variable qe-verbose? \ Flag for displaying diagnostic message.
    qe-verbose? off
instance variable ext-lbt? \ Flag for execution of external loopback test.
    ext-lbt? off
\ qe0-package.fth:
6 instance buffer: macbuf \ Contain mac address.
0 instance value obp-tftp \ Contain ihandle of TFTP package.
instance variable qe-nbytes \ Buffer size of higher layer receiver.
instance variable qe-buf \ Buffer address of higher layer receiver.

headers

```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
\ -----
\ qe.fth

: wlk-test ( mask addr #bits -- success? ) " wlk-test" $call-parent ;
: set-ms-timeout ( #ms -- ) " set-ms-timeout" $call-parent ;
: ms-timeout? ( -- flag ) " ms-timeout?" $call-parent ;

fload mace.h.fth          \ Not included.
fload channel.h.fth      \ Not included.
fload qemap.fth          \ Not included, refer to example 2.
fload qecore.fth
fload timed-receive.fth
fload qeregtest.fth     \ Not included, refer to example 2.
fload qetest.fth
fload qe0-package.fth

\ -----
\ qe0-package.fth
\ Define the required methods for the network qe driver

set-my-channel#
external
: read ( buf len -- -2 | actual-len )
  qe0-read
;
: write ( buf len -- actual-len )
  qe0-write
;
: selftest ( -- flag ) \ Flag 0 if passes test.
  qe0-selftest
;
: watch-net ( -- )
  qe0-watch-net
;
: load ( addr -- len )
  qe0-load
;
: open ( -- okay? )
  qe0-open
;
: close ( -- )
  qe0-close
;
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
: reset ( -- )
  qe0-reset
;

headers
\ -----
\ qecore.fth
\ Main core of QEC/MACE per channel Tx/Rx drivers.
\
\ SQEC has the following features:
\   - Supports four independent IEEE 802.3 10BASE-T twisted pair interfaces.
\   - Supports SBus parity checking.
\   - Supports 32 bit of DVMA addressing.
\   - Automatic rejection/discard of receive/transmit packets
\     when receive/transmit suffers from errors.
\
headerless
\ *** Rx/Tx Ring Descriptor Layout ***

struct ( Rx/Tx Descriptor )
4 field >flags          \ OWN, SOP, EOP, size/length
4 field >addr           \ buffer address
( total-length ) constant /md

hex
\ Definition for >flag field.
\ Bit[10:0] - Rx for W is buffer size, Rx for R is byte count, Tx for W is byte count.
8000.0000      constant own    \ For both Rx & Tx.
4000.0000      constant stp    \ For Tx only.
2000.0000      constant enp    \ For Tx only.
07ff          constant lenmask

\ Value to write to message descriptor to enable it for use
enp stp or own or      constant ready

\ *** buffer sizes and counts ***

\ Xmit/receive buffer structure.
\ This structure is organized to meet the following requirements:
\   - starts on an QEBURSTSIZE (64) boundary.
```


Code Example 11-3 QED Bootable Driver Sample (Continued)

```

\      - qebuf is an even multiple of QEBURSTSIZE.
\      - qebuf is large enough to contain max frame (1518) plus
\          QEBURSTSIZE for alignment adjustments.
\
\ Similar to the 7990 ethernet controller, the QEC and the Software driver
\ communicate via ring descriptors. There are separate Rx & Tx descriptor
\ rings of 256 entries. Unlike 7990 the number of descriptor entries
\ is not programmable (fixed at 256 entries).

decimal
/md constant /rmd      \ rmd size = 8
/md constant /tmd      \ tmd size = 8
1792 constant /rbuf    \ 7*256 receive buffer size at least 1518+128=1636
1600 constant /tbuf    \ transmit buffer size
256 constant #rmds
256 constant #tmds
\ 1 constant #tbufs    \ Just allocate one buffer for transmitter buffer pool.
32 constant #rbufs    \ # buffers allocated for receiver buffer pool.

#rmds /rmd *          value /rmds
#tmds /tmd *          value /tmds

headers

: restart?-on ( -- ) restart? on ;

\ Conversion between cpu dma address and io dma address.
: cpu>io-addr ( cpu-addr -- io-addr ) cpu-dma-base - io-dma-base + ;
: io>cpu-addr ( io-addr -- cpu-addr ) io-dma-base - cpu-dma-base + ;

\ buffer# to address calculations
: rmd#>rmdaddr ( n -- addr ) /rmd * rmd0 + ;
: rbuf#>rbufaddr ( n -- addr ) #rbufs mod /rbuf * io-rbuf0 + ;
: tmd#>tmdaddr ( n -- addr ) /tmd * tmd0 + ;
\ address to buffer# calculations
: rmdaddr>rmd# ( addr -- n ) rmd0 - /rmd / ;

\ *** Qe message descriptor ring access ***

\ Get current rx/tx message descriptor ring pointer (on CPU side).
: nexttmd@ ( -- cpu-tmd-addr ) nexttmd @ ;
: nextrmd@ ( -- cpu-rmd-addr ) nextrmd @ ;

```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```

\ get location of buffer
: addr@ ( rmd/tmd-addr -- buff-addr ) >addr rl@ ;

: status@ ( rmd/tmd-addr -- statusflag ) >flags rl@ ;

\ gets length of incoming message, receive only
: length@ ( rmdaddr -- messagelength ) >flags rl@ lenmask and ;

\ Set current rx/tx message descriptor ring pointer (on CPU side).
: nextcmd! ( cpu-rmd-addr -- ) nextcmd ! ;
: nexttmd! ( cpu-tmd-addr -- ) nexttmd ! ;

\ Store buffer address into message descriptor
: addr! ( buff-addr rmd/tmd-addr -- ) >addr rl! ;

\ Set length of message to be sent - transmit only
: length! ( length rmd/tmd-addr -- ) >flags rl! ;

\ *** Qe synchronization ***

\ Sync the message descriptor after cpu or device writes it.
: qesynciopb ( md -- )
  dup cpu>io-addr /md          ( cpu-addr io-addr size )
  dma-sync
;

\ Sync the transmitting/received buffer after cpu/device writes it.
: qesyncbuf ( md -- )
  dup addr@ dup io>cpu-addr swap      ( md cpu-buf-addr io-buf-addr )
  rot length@                        ( cpu-buf-addr io-buf-addr size )
  dma-sync
;

\ The buffer was already put back, put the descriptor in the chip's ready list
: give-buffer ( rmd/tmd-addr -- )
  dup >flags dup rl@ ready or swap rl!      ( md )
  \ Sync the descriptor so the device sees it.
  qesynciopb                                ( )
;

\ *** Qe error handling ***

: get-qe-status ( -- channel-status )
  qecc-status@ status @ or dup status !

```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
;

\ get receive errors, receive only
: rerrors@ ( -- errorsflag ) get-qe-status c-rerr-mask and ;

\ gets transmit errors, transmit only
: xerrors@ ( -- errorsflag ) get-qe-status c-terr-mask and ;

\ Clear transmit/receive/all error flags
: clear-terrors ( -- ) status @ c-terr-mask not and status ! ;
: clear-rerrors ( -- ) status @ c-rerr-mask not and status ! ;
: clear-errors ( -- ) status off restart? off ;
: clear-tint ( -- ) status @ c-tint not and status ! ;

\ *** Basic initialization routines ***

\ words to set loopback control mode in UTR(R29) & promiscuous mode in MACCC(R13)
\ Bit<7> to control promiscuous mode, Bits<2:1> to control loopback mode,
\ Bit<0> to test the cable connection.
1 constant m-cable

: set-loop-mode ( -- ) mode @ m-loop-mask and m-rpa or mace-utr! ;
: set-prom-mode ( -- ) mode @ m-prom and mace-maccc! ;
: check-cable-mode? ( -- flag ) mode @ m-cable = ;
: external-loopback? ( -- flag ) mode @ m-loop-mask and m-loop-ext = ;

\ Check existence of no-tpe-test property to initialize disable-tpe-link-test bit.
\ Enable tpe-link-test if the property doesn't exist,
\ or disable tpe-link-test if the property exists.
: init-link-test ( -- )
  \ Disable link test for external loopback mode.
  external-loopback? if m-dlnkst mace-phycc! exit then
  " no-tpe-test" get-my-property if 0
  else 2drop m-dlnkst then
  mace-phycc!
;

\ Enable/disable tpe-link-test
: setup-link-test ( enable-flag -- )
  " no-tpe-test" " get-property" eval if
    \ Property doesn't exist, already enabled.
    0= if 0 0 " no-tpe-test" property then
  else 2drop \ Currently disabled.
  if " no-tpe-test" delete-property then
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
    then
;
\
\ After doing a port select of the twisted pair port, the
\ driver needs to give ample time for the MACE to start
\ sending pulses to the hub to mark the link state up.
\ Loop here and check of the link state has gone into a
\ pass state.
\
: link-state-fail? ( -- fail? )
  d# 1000 set-ms-timeout
  begin
    mace-phycc@ m-lnkst and
  while
    ms-timeout? if
      check-cable-mode? if
        ." failed, transceiver cable problem? or check the hub." cr
        true
      else
\      m-dlnkst mace-phycc!
        false
      then
        exit
    then
  repeat
  check-cable-mode? if ." passed." cr then
  false
;

: set-physical-address ( -- )
  m-addrchg mace-iac!
  begin mace-iac@ m-addrchg and 0= until
  m-phyaddr mace-iac!
\ Store least significant byte first.
  this-en-addr 6 bounds do i c@ mace-paddr! loop
  0 mace-iac!
;

: set-address ( en-addr len -- )
  drop this-en-addr 6 move ;

: set-logaddr-filter ( -- )
  m-addrchg mace-iac!
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
begin mace-iac@ m-addrchg and 0= until
m-logaddr mace-iac!
8 0 do 0 mace-laddrf! loop
0 mace-iac!
;

\ Reset (or stop) the qec channel.
\ Issue a soft reset to the desired Mace.
\ Then issue a soft reset to the desired channel in QEC.
\ Chip reset algorithm:
\ Set the reset bit then wait until the reset bit cleared.
\ Timeout in 0.1 sec if fail.
\

: channel-reset ( -- fail? )
m-swrst mace-biucc! \ Issue Mace reset.
d# 100 set-ms-timeout
begin
mace-biucc@ m-swrst and
while
ms-timeout? if ." Cannot reset Mace" cr true exit then
repeat
c-rst qecc-control! \ Reset QEC channel registers.
d# 100 set-ms-timeout
begin
qecc-control@ c-rst and
while
ms-timeout? if ." Cannot reset QEC channel" cr true exit then
repeat
false
;

\ Initialize a single message descriptor
: rmd-init ( rbufaddr rmdaddr -- )
/rbuf over length! \ Buffer length
addr! \ Buffer address
;

\ Set up the data structures necessary to receive a packet
: init-rxring ( -- )
rmd0 nextrmd!
#rmds 0 do i rbuf#>rbufaddr i rmd#>rmdaddr rmd-init loop
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```

;
\
\ Initially first N=#rbufs descriptors with one-to-one association with a
\ buffer are made ready, the rest (256-N) not ready, then turn on receiver.
\ Whenever a receive buffer is processed, the information is copied out,
\ the buffer will be linked to the ((current+N)%256) entry then make the
\ entry is ready. Ie. The window of N ready descriptor/buffer pair is
\ moving around the ring.
\
: enable-rxring ( -- )
  #rbufs 0 do i rmd#>rmdaddr give-buffer loop
;

\ transmit buffer initialize routine
: init-txring ( -- )
  tmd0 nexttmd!
  #tmds 0 do io-tbuf0 i tmd#>tmdaddr addr! loop
;

\ *** Receive packet routines ***

\ Utility words used in .rerr-text & .terr-text.
: bits ( mask #right-bits -- mask' right-bits )
  >r dup /n 8 * r@ - tuck << swap >> ( mask bits ; RS: #bits )
  swap r> >> swap ( mask' bits )
;

: lbit ( mask -- mask' rightest-bit-value ) 1 bits ;
: .rerr-text ( -- )
  rerrors@
  lbit if ." SBus Rx Error Ack " restart?-on then
  lbit if ." SBus Rx Parity " restart?-on then
  lbit if ." SBus Rx Late " restart?-on then
  lbit if ." Data Buffer Too Small " then
\ lbit if ." Rx packet Dropped " then
  lbit drop \ Skip drop error, happens all the time
  lbit drop \ Skip receive interrupt bit.
  lbit if ." CRC error " then
  lbit if ." Framing error " then
  lbit if ." MACE Rx Late Collision " then
  lbit if ." MACE FIFO overflow " then
  lbit if ." MACE Missed Counter Overflow " then
  lbit if ." MACE Runt Counter Overflow " then

```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
lbit if ." MACE Rx Coll Counter Overflow " then
lbit if ." Collision error " then
drop cr
;

: (.receive-error ( -- )
  rerrors@ if .rerr-text then
;
' (.receive-error to .receive-error
' (.receive-error to .error

: to-next-rmd ( -- )
  /rmd nextrmd +!
  nextrmd@ rmd0 - /rmds >= if rmd0 nextrmd! then
;

\ *** Transmit packet routines ***

: to-next-tmd ( -- )
  /tmd nexttmd +!
  nexttmd@ tmd0 - /tmds >= if tmd0 nexttmd! then
;

\ Ignores the size argument, and uses the standard buffer.
: get-buffer ( dummysize -- buffer )
  drop nexttmd@ addr@          ( io-tbuf )
  io>cpu-addr                  ( cpu-tbuf )
;

\ Display time domain reflectometry information
\ : .tdr ( -- ) ;

: .terr-text ( -- )
  xerrors@
  d# 16 bits drop              \ Skip the receiver bits.
  lbit if ." SBus Tx Error Ack " restart?-on then
  lbit if ." SBus Tx Parity " restart?-on then
  lbit if ." SBus Tx Late " restart?-on then
  lbit if ." QEC Chained Tx Descriptor Error " restart?-on then
  lbit if ." QEC Tx Retry Counter Overflow " then
  lbit drop                    \ Skip transmit interrupt bit
  lbit if ." MACE >1518 Babble " then
  lbit if ." MACE Jabber " then
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
lbit if ." MACE FIFO Underflow " then
lbit if ." Tx Late Collision " then
lbit if ." Too Many Retries " then
lbit if ." Lost Carrier (transceiver cable problem?) " then
lbit if ." Excessive Defer " then
drop cr
;

\ print summary of any HARD errors
: (.transmit-error ( -- )
  xerrors@ if .terr-text then
;
' (.transmit-error to .transmit-error

\ Set up CPU page maps
: map-qe-buffers ( -- )
  #rbufs /rbuf *
\ 2KB (8*256) for tmds & 2KB (8*256) for rmds & 4KB for tbuf
\ ie. one page for tmds & rmds, one page for tbuf, the rest for rbufs.
  h# 2000 +
  to qe-dma-size

  \ Allocate and map that space
  qe-dma-size dma-alloc          ( dma-addr )

  \ Set the addresses of the various DMA regions used by the cpu.
  dup to cpu-dma-base
  dup to tmd0   h# 800 + ( next-address )
  dup to rmd0   h# 800 + ( next-address ) \ Enough for 256 entries
  dup to tbuf0  h# 1000 + ( next-address ) \ Enough for max packet
  to rbuf0      ( )
  tmd0 qe-dma-size false dma-map-in ( io-dma-addr )
  \ Set the addresses of the various DMA regions used by the qec chip.
  dup to io-dma-base
  dup to io-tmd0 h# 800 + ( next-address )
  dup to io-rmd0 h# 800 + ( next-address ) \ Enough for 256 entries
  dup to io-tbuf0 h# 1000 + ( next-address ) \ Enough for max packet
  to io-rbuf0    ( )

;
: unmap-qe-buffers ( -- )
  tmd0 io-tmd0 qe-dma-size dma-map-out
  tmd0 qe-dma-size dma-free
```


Code Example 11-3 QED Bootable Driver Sample (Continued)

```
0 to tmd0
;

\ *** Chips initialization routines ***

\ Initializes the QEC/Mace chips.
: channel-init ( -- fail? )
\ *** Initialize QEC per channel registers.
io-rmd0 qecc-rxring!
io-tmd0 qecc-txring!
c-rintmask qecc-rintmask!           \ Mask RINT.
c-tintmask qecc-tintmask!           \ Mask XINT.
my-chan# chmem * dup qecc-lmrxwrite! dup qecc-lmrxread!
rxbufsize + dup qecc-lmtxwrite! qecc-lmtxread!
c-qecerrmask qecc-qecerrmask!
c-macerrmask qecc-macerrmask!
\ *** Initialize MACE registers.
\ 0 mace-xmtfc!
m-apadxmt mace-xmtfc!           \ Set auto pad transmit for transmit frame control
0 mace-rcvfc!           \ Init. receive frame control.
\ Init. Interrupt Mask Register to mask rcvint & cerr and unmask xmtint
\   according QEC spec.
m-cerrm m-rcvintm or mace-imr!
\ Init. Bus Interface Unit Configuration Control to transmit after 64 bytes
\   have been loaded & byte swap.
m-xmtsp64 m-xmtspshift << m-bswp or mace-biucc!
\ Init. FIFO Conf Control to set transmit/receive fifo watermark update
m-xmtfw16 m-rcvfw32 or m-xmtfwu or m-rcvfwu or mace-fifocc!
m-10base-t mace-plscc!           \ Select twisted pair mode.
init-link-test           \ Init. tpe link test mode.
set-physical-address           \ Set mac address.
set-logaddr-filter           \ Set logical address filter.
0 mace-iac!
link-state-fail?           \ Wait and check the link state marked up.
mace-mpc@ drop           \ Read to reset counter and to prevent an invalid int.
set-loop-mode           \ Set UTR
set-prom-mode           \ Set MACCC
m-apadxmt not mace-xmtfc@ and mace-xmtfc!
m-astprcv not mace-rcvfc@ and mace-rcvfc!
;

\ Turn on the Mace, ready to tx/rx packets.
: enable-mace ( -- )
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```

m-enxmt m-enrcv or mace-macccc@ or mace-macccc!
;

\ *** Ethernet on/off routines ***

\ Initializes the QEC/Mace chips, allocating the necessary memory,
\ and enabling the transmitter and receiver.
: net-on ( -- flag )          \ true if net-on succeeds
  clear-errors
  mac-address set-address
  channel-reset 0= if
    init-txring
    init-rxring
    channel-init 0= dup if
      enable-rxring
      enable-mace
    then
  else false
  then
;

\ Stop the activity of this net channel.
: net-off ( -- ) channel-reset drop init-link-test ;

\ *** Main receive routines ***

\
\ Whenever a receive buffer is processed, the information is copied out,
\ the buffer will be linked to the ((current+N)%256)th entry then make the
\ entry is ready ie.the window of N ready descriptor/buffer pair is
\ moving around the ring.
\
\ If 256 (#rmds) is multiples of N (#rbufs=32), we don't need to link the
\ next-ready-rmd with the current processed rx buffer dynamically. They can
\ be set at the initialization time statically. For run time, we just need
\ to make the ((current+N)%256)th rmd ready.
\
: return-buffer ( buf-handle -- )
  rmdaddr>rmd#                ( [io-rbuf] rmd# )
  #rbufs + #rmds mod          ( [io-rbuf] next-ready-rmd# )
  rmd#>rmdaddr                ( [io-rbuf] next-ready-rmd )
  dup addr@ over rmd-init      ( next-ready-rmd ; Set length )
  give-buffer                  ( ; Make it ready )

```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
to-next-rmd                \ Bump SW nextcmd to next one
;

: receive-ready? ( -- packet-waiting? )
  restart? @ if net-on drop then
  nextcmd@                  ( rmd )
  \ Sync RMD before CPU looking at it.
  dup qesynciopb            ( rmd )
  status@ own and 0=        ( flag )
;

: receive ( -- buf-handle buffer len )          \ len non-zero if packet ok
  nextcmd@ dup addr@          ( rmd io-rbuf-addr )
  io>cpu-addr                 ( rmd cpu-rbuf-addr )
  over length@                ( rmd cpu-rbuf-addr len )
  rerrors@ if
    .receive-error clear-rerrors
  then
  dup if                       ( rmd cpu-rbuf-addr len )
  \ Sync the received buffer before CPU looking at it.
  nextcmd@ qesyncbuf          ( rmd cpu-rbuf-addr len )
  then
;

\ *** Main transmit routines ***

: set-timeout ( interval -- ) get-msecs + alarmtime ! ;

: timeout? ( -- flag ) get-msecs alarmtime @ >= ;

: 10us-wait ( -- ) d# 10 begin 1- dup 0= until drop ;

\ Wait until transmission completed
: send-wait ( -- )
\ Wait the packet to get to the local memory, ready for MACE to xmit.
  d# 2000 set-timeout          \ 2 second timeout.
  begin
  get-qe-status
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```

        c-tint and                \ Transmit interrupt bit set?
        timeout? or              \ Or timeout?
until
timeout? if
    ." TINT was not set!" cr true exit
then
\ Transmit completion, sync TMD before looking at it.
nexttmd@ dup qesynciopb        ( tmd )
status@ own and if            ( flag )
    ." Tx descriptor still owned by QEC!" cr
then
\ Wait the packet to get to net, make sure at most one xmit packet in MACE FIFO.
d# 1000 set-timeout          \ 1 second timeout.
begin
    10us-wait
    qecc-lmtxwrite@ qecc-lmtxread@ =
    timeout? or
until
timeout? if
    ." Tx packet not out to net!" cr
then
false
;

\ This send routine does not enforce the minimum packet length. It is
\ used by the loopback test routines.
: short-send ( buffer length -- error? )
    clear-tint                \ Erase tint status bit.
\ discard buffer address, assumes using nexttmd
nip nexttmd@                  ( length tmd )
tuck length!                  ( tmd ; Set length )
\ Sync the transmit buffer so the device sees it.
dup qesyncbuf                 ( tmd )
give-buffer                   ( ; Give tmd to chip )
c-tmd qecc-control!          \ Bang the chip, let chip look at it right away
send-wait                     ( fail? ) \ wait for completion
xerrors@ dup if               ( fail? error? )
    .transmit-error clear-terrors
then or                        ( error? )
to-next-tmd                   ( error? )
restart? @ if net-on drop then ( error? )
c-hard-terr-mask and          ( hard-error? )
;

```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
\ Transmit packet routine, no S/W retry on this layer.
: net-send ( buffer length -- error? ) \ error? is contents of chan-status
  d# 64 max          \ force minimum length to be 64
  short-send        ( error? )
;

\ -----
\ timed-receive.fth
\ Implements a network receive that will timeout after a certain interval.

decimal

: multicast? ( handle data-address length -- handle data-address length flag )
  \ Check for multicast/broadcast packets
  over          ( ... data-address )
  c@ h# 80 and dup if \ Look at the multicast bit
    ( handle data-address length multicast? )
    handle-broadcast-packet
  then
;

: receive-good-packet ( -- [ buffer-handle data-address length ] | 0 )
  begin
    begin
      timeout? if false exit then
      receive-ready?
    until
      receive dup 0=
  while
    .error 2drop return-buffer
  repeat
;

: receive-unicast-packet ( -- [ buffer-handle data-address length ] | 0 )
  begin
    receive-good-packet dup 0= if exit then
    multicast?
  while
    2drop return-buffer
  repeat
;

\ Receive a packet, filtering out broadcast packets and timing
\ out if no packet comes in within a certain time.
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```

: timed-receive ( timeout-msecs -- [ buffer-handle data-address length ] err?)
  set-timeout receive-unicast-packet ?dup 0=
;

\ -----
\ getest.fth
\ Define Qec/Mace loopback-test, net-init & watch-test routines.

\ This file contains Qec/Mace selftest routines.
\ It defines the following external words:
\   loopback-test ( internal/external-flag -- success? )
\   net-init ( -- success? )
\   watch-test ( -- )
\ Also it defines the following external variable.
\   qe-verbose? - Flag to indicate if want the test messages displayed.
\   ext-lbt? - Flag to indicate if run the external loopback test.
\
\ The algorithme for the loopback test:
\   Set internal or external loopback with no promiscuous mode.
\   Turn on the Qec/Mace Ethernet port.
\   If it succeeds, send out a short packet containing walking 0/1 patterns.
\   If it succeeds, wait for a period, check if receive the loopback packet.
\   If so, verify the length of the received packet is right.
\   Also check if the data of the received packet is right.
\   Return true if everything is fine, otherwise return false.

hex
headerless
create loopback-prototype
  ff c, 00 c, \ Ones and zeroes
  01 c, 02 c, 04 c, 08 c, 10 c, 20 c, 40 c, 80 c, \ Walking ones
  fe c, fd c, fb c, f7 c, ef c, 0df c, 0bf c, 7f c, \ Walking zeroes
  55 c, aa c,

: loopback-buffer ( -- addr len )
  d# 32 get-buffer ( addr )
  mac-address drop over 6 move \ Set source address
  mac-address drop over 6 + 6 move \ Set destination address
  loopback-prototype over d# 12 + d# 20 move \ Set buffer contents
  d# 32
;

: pdump ( addr -- )

```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
base @ >r hex
dup      d# 10 bounds do i c@ 3 u.r loop cr
d# 10 +  d# 10 bounds do i c@ 3 u.r loop cr
r> base !
;

\ Print loopback control type for verbose mode.
: .loopback ( -- )
  mode @ m-loop-mask and
  ?dup if
    dup m-loop-ext = if ." External " drop
    else ." Internal " m-loop-intmen = if ." (including Mendec) " then
    then
    ." loopback test -- "
  then
;

\ Print loopback control type for non-verbose mode,
\ it is used after any error occurs.
: ?.loopback ( -- )
  qe-verbose? @ 0= if .loopback then
;

: switch-off ( -- false ) qe-verbose? off false ;

: bad-rx-data ( buf-handle data-address -- false )
  ?.loopback
  ." Received packet contained incorrect data. Expected: " cr
  loopback-prototype pdump
  ." Observed:" cr
  d# 12 + pdump
  switch-off
;

\ Check the data of the received packet, return true if data is ok.
: check-data ( buf-handle data-address length -- ok? )
  drop ( buf-handle data-address )
  dup d# 12 + loopback-prototype d# 20 comp
  if bad-rx-data
  else drop ( buf-handle )
    return-buffer
    qe-verbose? @ if ." succeeded." cr then
    mode off true
  then
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
;
\ Check the length & data of the received packet, return true if data & len ok.
: check-len&data ( buf-handle data-address length -- ok? )
  \ The CRC is appended to the packet, thus it is 4 bytes longer than
  \ the packet we sent.
  dup d# 36 <>
  if ?.loopback
    ." Wrong packet length; expected 36, observed " .d cr
    switch-off
  else check-data
  then
;

headers
\ Run internal or external loopback test, return true if the test passes.
: loopback-test ( internal/external -- pass? )
  mode !
  qe-verbose? @ if ." " .loopback then
  net-on if
    loopback-buffer short-send if
      ?.loopback ." send failed." cr
      switch-off
    else
      d# 2000 timed-receive if
        ?.loopback
        ." Did not receive expected loopback packet." cr
        switch-off
      else ( buf-handle data-address length )
        check-len&data
      then
    then
  else
    switch-off
  then
  net-off mode off
;

\ If there is a normal external loopback test, then we don't need this.
\ MACE external loopback test requires a special cable. Don't run external
\ loopback test for selftest & watch-net.
: check-cable? ( -- ok? )
  m-cable mode ! ." Link state check -- "
```


Code Example 11-3 QED Bootable Driver Sample (Continued)

```
net-on          ( success? )
net-off mode off
;
\ Turn on the Ethernet port after pass loopback test.
\ Return true if net-init succeeds, otherwise return false if it fails.
: net-init ( -- flag )
  mode @          \ Save requested mode because loopback changes it.
  m-loop-int loopback-test
  if              ( mode-saved ; Pass internal loopback test. )
    ext-lbt? @    \ Run external loopback test if the ext-lbt? flag is set.
  \ qe internal loopback with mendec is equivalent to external loopback of le.
    if m-loop-intmen loopback-test else true then ( mode-saved )
    swap mode !   \ Restore the mode.
    if net-on     \ Pass loopback test, turn on the ethernet port.
      else false
      then
    else mode ! false
    then
;

headerless
: wait-for-packet ( -- )
  begin key? receive-ready? or until
;

headers
\ Check for incoming Ethernet packets.
\ Use promiscuous mode to check for all incoming packets.
: watch-test ( -- )
  ." Looking for Ethernet packets." cr
  ." \.' is a good packet. 'X' is a bad packet." cr
  ." Type any key to stop." cr
  begin
    wait-for-packet
    receive-ready?
    if receive
      if ." ." else ." X" then
        drop return-buffer
      then
    key? dup if key drop then
  until
;
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```

\ -----
\ qe0-package.fth
\ Implements the architectural interface for the qe driver

headerless
\
\ The network driver uses the standard "obp-tftp" support package for
\ implementation. The "obp-tftp" package implements the Internet Trivial File
\ Transfer Protocol (TFTP) for use in network booting. The "obp-tftp" package
\ defines the following methods to be used by the network driver:
\     open    ( -- okay? )
\     close   ( -- )
\     load    ( addr -- size )
\ The "obp-tftp" package uses the read and write methods of the network driver
\ for receiving and transmitting packets. The package assumes the size of the
\ maximum transfer packet is 1518 bytes. If the network driver needs bigger
\ maximum packet size, then it requires the method "max-transfer" defined,
\ the method will be called by the obp-tftp package to define the maximum
\ transfer packet size.
\
: init-obp-tftp ( -- okay? )
  " obp-tftp" find-package if          ( phandle )
    my-args rot open-package          ( ihandle )
  else 0
  then
  dup to obp-tftp                      ( ihandle | 0 )
  dup 0= if
    ." Can't open OBP standard TFTP package" cr
  then
;

: set-my-channel# ( -- )
\ If don't find the channel property, use 0.
  " channel#" get-my-property if 0 else decode-int nip nip then
  my-channel#!
;

headers
: qe-xmit ( bufaddr nbytes -- #sent )
  tuck get-buffer                      ( nbytes bufaddr ether-buffer )
  tuck 3 pick move                     ( nbytes ether-buffer )
  over net-send if drop 0 then         ( #sent )
;

```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
: qe-poll ( bufaddr nbytes -- #received )
  qe-nbytes ! qe-buf !          ( )
  receive-ready? 0= if 0 exit then \ Bail out if no packet ready
  receive ?dup if                ( rmd ether-buffer length )
    dup >r                        ( rmd ether-buffer length )
    qe-nbytes @ min                ( rmd ether-buffer length' )
    qe-buf @ swap move             ( rmd )
    return-buffer r>              ( #received )
  else
    drop return-buffer 0          ( 0 )
  then
;
: set-vectors ( -- )
  ['] (.receive-error to .error
  ['] (.transmit-error to .transmit-error
  ['] noop to handle-broadcast-packet
;
: map-qe ( -- )
  mace 0= if \ Do mapping if it is unmapped.
    map-chips
    map-qe-buffers
  then
;
: unmap-qe ( -- )
  mace if \ Do unmapping if it is mapped.
    unmap-qe-buffers
    unmap-chips
  then
;
: qe-loopback-test ( -- flag ) \ flag true if passes test
  set-vectors
  mode off qe-verbose? on
  ext-lbt? on
  net-init
  ext-lbt? off
  dup if net-off drop check-cable? then
  qe-verbose? off
;
: (watch-net) ( -- )
  map-qe
  set-vectors
  m-prom mode !
  qe-verbose? off
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
    ext-lbt? off
    net-init  if watch-test net-off  then
    unmap-qe
;

external
: qe0-read  ( buf len -- -2 | actual-len )
  qe-poll  ?dup 0=  if -2  then
;
: qe0-write ( buf len -- actual-len ) qe-xmit ;
: qe0-selftest ( -- flag )      \ Flag 0 if passes test.
  map-qe
  qe-reg-test      ( success? )
  if
    qe-loopback-test 0=      \ Alternate the return flag.
  else
    true
  then      ( failure? )
  unmap-qe
;
: qe0-watch-net ( -- )
  qe0-selftest 0=  if (watch-net)  then
;

: qe0-load  ( addr -- len ) " load" obp-tftp $call-method ;
: qe0-open  ( -- okay? )
  map-qe
  set-vectors
  mode off  qe-verbose? off
  net-init 0=  if unmap-qe false exit  then

  mac-address drop macbuf 6 move      \ Update macbuf.
  macbuf 6 encode-string " mac-address" property

  init-obp-tftp 0=  if close false exit  then
  true
;
: qe0-close ( -- )
  obp-tftp ?dup  if close-package  then
  mace  if net-off  then
  unmap-qe
;
: qe0-reset ( -- )
```

Code Example 11-3 QED Bootable Driver Sample (Continued)

```
mace if net-off
     else map-chips net-off unmap-chips then
;
headers
```


This chapter describes programming requirements for serial devices, and gives examples of serial device drivers. Serial devices are byte-oriented, sequentially-accessed devices such as asynchronous communication lines (often attached to a dumb terminal).

Required Methods

The `serial` device driver must declare the `serial` device type, and must implement the methods `open` and `close`, as well as the following:

`install-abort (--)`

Instruct the driver to begin periodic polling for a keyboard abort sequence. `install-abort` is executed when the device is selected as the console input device.

`read (addr len -- actual)`

Read *len* bytes of data from the device into memory starting at *addr*. Return the number of bytes actually read, *actual*, or `-2` if no bytes are currently available from the device. `-1` is returned if other errors occur.

`remove-abort (--)`

Instruct the driver to cease periodic polling for a keyboard abort sequence. `remove-abort` is executed when the console input device is changed from this device to another.

`write (addr len -- actual)`

Write *len* bytes of data to the device from memory starting at *addr*. Return the number of bytes actually written, *actual*.

Required Properties

These are the standard properties of a serial driver:

Table 12-1 Serial Driver Required Properties

Property Name	Value
<code>name</code>	" SUNW, zs "
<code>reg</code>	list of registers {device-dependent}
<code>device_type</code>	" serial "

Device Driver Examples

The following three examples are serial device drivers for the Zilog 8530 SCC (UART) chip.

- The first sample is a short driver which simply creates a device node and declare the properties for the device.
- The second sample is a more sophisticated driver that defines methods to control and access the device.
- The third sample shows the complete serial device driver.

Simple Serial FCode Program

```
\ This driver creates a device node and publishes the minimum required set of
\ properties.
fcode-version3
  hex
  " SUNW,zs" name
  my-address 10.0000 + my-space 8 reg
  7 encode-int " interrupts"  property
end0
```

Extended Serial FCode Program

Code Example 12-1 Extended Serial FCode Program

```
\ In addition to publishing the properties, this sample driver
\ provides methods to access and control the serial ports.
\
\ The following main methods are provided:
\ - usea  ( -- )
\   Selects serial port A. All subsequent operations will
\   be directed to port A
\ - useb  ( -- )
\   Selects serial port B. All subsequent operations will
\   be directed to port B
\ - uemit ( char -- )
\ Emits a given character to the selected serial port.
\ - ukey  ( -- key )
\ Retrieves a character from the selected serial port.
\ - read  ( addr len -- #read )
\ Reads "len" number of characters from the selected port,
\ and store them at "addr".
\ - write ( addr len -- #written )
\ Writes "len" number of characters from the buffer located
\ at "addr" to the selected serial port.

fcode-version3
hex

  " SUNW,zs" name
  my-address 10.0000 + my-space 8 reg
  7 encode-int " interrupts"  property
```

≡ 12

Code Example 12-1 Extended Serial FCode Program (Continued)

```
: >phys-addr ( offset -- phys.lo phys.mid phys.hi )
>r my-address r> 0 d+ my-space
;
: do-map-in ( offset size -- virt )
>r >phys-addr r> " map-in" $call-parent
;
: do-map-out ( virt size -- ) " map-out" $call-parent ;

: /string ( addr len n -- addr+n len-n ) tuck - -rot + swap ;

1 constant RXREADY \ received character available
4 constant TXREADY \ transmit buffer empty

0 instance value uart \ define uart as an "per-instance" value.
0 instance value uartbase
h# ff instance value mask-#data \ mask for #data bits
h# 10 instance buffer: mode-buf

\ The following line assumes that A2 selects the channel within the chip
: usea ( -- ) uartbase 4 + to uart ;
: useb ( -- ) uartbase to uart ;
: uctl! ( c -- ) uart rb! ;
: uctl@ ( -- c ) uart rb@ ;

\ The following line assumes that A1 chooses the command vs. data port
: udata! ( c -- ) uart 2 + rb! ;
: udata@ ( -- c ) uart 2 + rb@ ;

\ Test for "break" character received.
: ubreak? ( -- flag ) 10 uctl! uctl@ h# 80 and 0<> ;

\ Clear the break flag
: clear-break ( -- )
begin ubreak? 0= until \ Let break finish
udata@ drop \ Eat the null character
30 uctl! \ Reset errors
;

: uemit? ( -- flag ) uctl@ TXREADY and ;
: uemit ( char -- ) begin uemit? until udata! ;
```

Code Example 12-1 Extended Serial FCode Program (Continued)

```

: ukey? ( -- flag ) uctl@ RXREADY and ;
: ukey ( -- key ) begin ukey? until udata@ ;

: uwrite          ( addr len -- #written )
  tuck bounds ?do ( len )
    i c@ uemit    ( len )
  loop           ( len )
;

: uread ( addr len -- #read )          \ -2 for none available right now
  ukey? 0= if 2drop -2 exit then      ( addr len )
  tuck                                 ( len addr len )
  begin dup 0<> ukey? 0<> and while   ( len addr len )
    over ukey mask-#data and swap c!  ( len addr len )
    1 /string                          ( len addr' len' )
  repeat                               ( len addr' len' )
  nip -                                 ( #read )
;

external
: read ( addr len -- #read ) uread ;
: write ( addr len -- #written ) uwrite ;

end0

```

Complete Serial FCode Program

Code Example 12-2 Complete Serial FCode Program

```

\ In addition to the methods defined in the above driver sample,
\ this version defines more methods to initialize, test, and access
\ the serial ports.
\ The new main methods are:
\ - inituarts ( -- )
\   Initializes both serial ports A and B.
\ - open ( -- okay? )
\   Maps in the uart chip. Selects port A on default, then check
\   my-args, if port B was specified, then selects port B instead.
\ - close ( -- )
\   Unmap the uart chip.
\ - selftest ( -- )

```

≡ 12

Code Example 12-2 Complete Serial FCode Program (Continued)

```
\   Performs selftest on both Port A and B.
\ - install-abort ( -- )
\   Sets up alarm to do poll-tty every 10 miliseconds.
\ - remove-abort ( -- )
\   Removes the poll-tty alarm.

fcode-version3
hex

    " SUNW,zs" name
my-address 10.0000 + my-space 8 reg
7 encode-int " interrupts" property
" serial" device-type

: >phys-addr ( offset -- phys.lo phys.mid phys.hi )
  >r my-address r> 0 d+ my-space
;

: do-map-in ( offset size -- virt ) >r >phys-addr r> " map-in" $call-parent ;
: do-map-out ( virt size -- ) " map-out" $call-parent ;

: /string ( addr len n -- addr+n len-n ) tuck - -rot + swap ;

fload inituarts.fth
fload ttydriver.fth
end0

\-----
\ inituarts.fth

hex
headerless
create uart-init-table
\ 9 c, c0 c,    \ Master reset channel a (80), channel b (40)

 9 c, 2 c,    \ Don't respond to intack cycles (02)

4 c, 44 c,    \ No parity (00), 1 stop bit (04), x16 clock (40)

3 c, c0 c,    \ receive 8 bit characters (c0)
5 c, 60 c,    \ transmit 8 bits (60)
e c, 82 c,    \ Processor clock is baud rate source (02)
```

Code Example 12-2 Complete Serial FCode Program (Continued)

```

b c, 55 c,      \ TRxC = xmit clk (01), enable TRxC (04), Tx clk is baud (10),
                \ Rx clk is baud (40)
c c,  e c,      \ Time constant low
d c,  0 c,      \ Time constant high

3 c, c1 c,      \ receive 8 bit characters (c0), enable (01)
5 c, 68 c,      \ transmit 8 bits (60), enable (08)
e c, 83 c,      \ Processor clock is baud rate source (02), Tx enable (01)

0 c, 10 c,      \ Reset status bit latches

ff c, ff c,     \ Mark end of data

\-----
\ ttydriver.fth - Driver for Zilog 8530 SCC (UART) chips.

hex
0 instance value uartbase

create default-mode
\  0      1      2      3      4      5      6      7
  00 c,  00 c,  00 c,  c1 c,  44 c,  68 c,  00 c,  00 c,

\  8      9      a      b      c      d      e      f
  00 c,  02 c,  00 c,  55 c,  0e c,  00 c,  83 c,  00 c,

      0 instance value  uart      \ define uart as an "per-instance" value.
h# ff instance value  mask-#data \ mask for #data bits
h# 10 instance buffer: mode-buf

create masks  1f c,  7f c,  3f c,  ff c,

\ The following line assumes that A2 selects the channel within the chip
: usea  ( -- )  uartbase 4 + to uart  ;
: useb  ( -- )  uartbase to uart  ;
: uctl! ( c -- ) uart  rb!  ;
: uctl@ ( -- c ) uart  rb@  ;

\ The following line assumes that A1 chooses the command vs. data port
: udata! ( c -- ) uart  2 + rb!  ;
: udata@ ( -- c ) uart  2 + rb@  ;

\ Write all the initialization sequence to both uarts

```

≡ 12

Code Example 12-2 Complete Serial FCode Program (Continued)

```
: inituart ( -- )
  uart-init-table
  begin dup c@ ff <> while
    dup c@ uctl! dup cal+ c@ uctl!
    /c 2* +
  repeat
  drop
;

: inituarts ( -- ) usea inituart useb inituart usea ;

\ Test for "break" character received.
: ubreak? ( -- break? ) 10 uctl! uctl@ h# 80 and 0<> ;

\ Clear the break flag
: clear-break ( -- )
  begin ubreak? 0= until \ Let break finish
  udata@ drop           \ Eat the null character
  30 uctl!              \ Reset errors
;

1 constant RXREADY      \ received character available
4 constant TXREADY      \ transmit buffer empty

: uemit? ( -- emit? ) uctl@ TXREADY and ;
: uemit ( char -- ) begin uemit? until udata! ;

: ukey? ( -- key? ) uctl@ RXREADY and ;
: ukey ( -- key ) begin ukey? until udata@ ;

: uwrite ( addr len -- #written )
  tuck bounds ?do ( len )
  i c@ uemit ( len )
  loop ( len )
;

: uread ( addr len -- #read ) \ -2 for none available right now
  ukey? 0= if 2drop -2 exit then ( addr len )
  tuck ( len addr len )
  begin dup 0<> ukey? 0<> and while ( len addr len )
    over ukey mask-#data and swap c! ( len addr len )
    1 /string ( len addr' len' )
  repeat ( len addr' len' )
```

Code Example 12-2 Complete Serial FCode Program (Continued)

```
        nip -                                ( #read )
;
: poll-tty ( -- )
  ttylock @ if exit then
  ubreak? if clear-break user-abort then
;

external
: open ( -- okay? )
  phys-addr 8 do-map-in to uartbase
  usea
  my-args                                ( arg-str )
  ascii , left-parse-string if          ( rem addr )
    c@ ascii b = if                      ( rem )
      2drop                               ( )
      useb                                ( )
    then                                  ( rem )
  else                                    ( rem addr )
    drop 2drop                            ( )
  then                                     ( )

  true
;

: close ( -- ) uartbase 8 do-map-out ;

headers
: utest ( -- 0 ) h# 7f bl ?do i uemit loop 0 ;

external
: selftest ( -- error? )
  open 0= if ." Can't open device" true exit then
  my-args if ( addr )
    c@ case
      ascii a of usea endof
      ascii b of useb endof
      ( default ) ." Bad zs port letter" drop false exit
    endcase
  else \ No port letter so test both ports.
    drop
    usea utest
    useb utest
    or close exit ( fail? )
```

≡ 12


Code Example 12-2 Complete Serial FCode Program (Continued)

```
    then
    utest          ( fail? )
    close

;
: read  ( addr len -- #read )    uread  ;
: write ( addr len -- #written ) uwrite ;
: install-abort ( -- ) ['] poll-tty d# 10 alarm ;
: remove-abort  ( -- ) ['] poll-tty 0 alarm ;

\ "seek" might be implemented to select a load file name
\ Implement "load" ( optional )
headers
```


PCI FCode Driver Example

13 

This example PCI FCode driver illustrates how to handle PCI-related registers in FCode.

```
id: %Z%M% %I% %E%
purpose: A sample network PCI FCode driver showing use of Vital Product Data and
access to ROM.
copyright: Copyright 1994-1998 Sun Microsystems, Inc. All Rights Reserved

\ Each sample PCI card will define one hme device node:
\
\
\          pci
\          |
\          |
\          |
\          hme
\
\ The general pathname (after pci) for a hme node is
\     hme@D,1
\ where D is Device#.
\
```

≡ 13

```
\ name      hme
\          Phys.hi   Phys.mid   Phys.lo   Size.hi   Size.lo
\          -----
\ reg       00BB.XX00 0000.0000 0000.0000 0000.0000 0000.0000 (Cfg)

\          02BB.XX10 0000.0000 0000.0000 0000.0000 0000.7020 (memory)

\
\          where "BB" = PCI Bus #
\          where "XX" = PCI Device # | Function #
\

Fcode-version3

hex

\ 3.x token interpreter extends bit[31] all the way to bit[63].
\ To create numbers with bit[31] set, but not extended to bit[63],
\ we need to use x-num
ff ff ff ff bljoin constant num-32
: x-num ( n -- l ) num-32 and ;
: and ( a b -- a-and-b ) and x-num ;
: or ( a b -- a-or-b ) or x-num ;
: lshift ( x n -- x' ) lshift x-num ;
: rshift ( x n -- x' ) rshift x-num ;

f200.0000 x-num constant xreg-mask
\ my-address gives two 32 bit numbers for PCI case.
my-address      constant my-bus-addr-mid  constant my-bus-addr-low
my-space        constant my-bus-space
2 constant pci-hme-intr      \ for PCI card

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

```

...
...

\ for mac id on the fcode prom itself
h# 24 constant vpdp-loc
h# c000 value vpd-base
0 value cheer-rombase
0 value vpd-addr
h# 1.0000 value /cheer-rom

\ HappyMeal Enet Address Map
hex
000.0000    constant    global-regs-offset
7020        constant    /total-reg-space

h# 4000      constant max-frame-size( d# 1536 for le )
d# 48        constant address-bits
h# 10        constant cfg-bar0
h# 4          constant cfg-cmd-reg
h# 30        constant cfg-rom-base
h# 200.0000 constant 32-bit-mem-ss

: encode-ints ( nn .. n1 n -- adr len )
  0 0 encode-bytes  rot 0 ?do rot encode-int encode+ loop
;

: xdrreg ( addr space size -- adr len )
  >r encode-phys r> 0 2 encode-ints encode+
;

: offset>physical-addr ( offset -- paddr.lo paddr.mid paddr.hi )
  my-bus-addr >r + r> my-bus-space
  32-bit-mem-ss or cfg-bar0 or \ OR in "ss" = memory, base reg=10
;

: create-hme-attributes ( -- )

  " SUNW,hme"      name
  " SUNW,cheerio" encode-string " model" property
  " pci108e,1001" encode-string " pci108e,1001" encode-string
                    encode+ " compatible" property

```

≡ 13

```
my-bus-addr my-bus-space 0 xdrreg
global-regs-offset offset>physical-addr /total-reg-space
xdrreg encode+
" reg" property

max-frame-size encode-int " max-frame-size" property
address-bits encode-int " address-bits" property
pci-hme-intr encode-int " interrupts" property
" network" device-type
" %I%" encode-string " version" property \ FCode PROM version
;

: enable-cheer-cmd-reg ( -- )
\ PCI Command Register Settings
\ h# 100 = SERR# Enable
\ h# 40 = Parity Error Enable
\ h# 4 = Mastering Enable
\ h# 2 = Memory Access Enable
my-bus-space cfg-cmd-reg + " config-w@" $call-parent ( cmd-reg )
\ Set PCI Command bits
h# 146 or
my-bus-space cfg-cmd-reg + " config-w!" $call-parent
;

: enable-cheer-rom ( -- )
my-bus-space cfg-rom-base + " config-l@" $call-parent ( cmd-reg )
\ enable rom accesses
h# 1 or
my-bus-space cfg-rom-base + " config-l!" $call-parent
;

: disable-cheer-rom ( -- )
my-bus-space cfg-rom-base + " config-l@" $call-parent ( cmd-reg )
\ disable rom accesses
h# 1 invert and
my-bus-space cfg-rom-base + " config-l!" $call-parent
;

: map-cheer-rom ( -- )
0 0 my-bus-space 32-bit-mem-ss or cfg-rom-base or /cheer-rom
" map-in" $call-parent to cheer-rombase
;
```

```
: unmap-cheer-rom ( -- )
  cheer-rombase /cheer-rom " map-out" $call-parent
  0 to cheer-rombase
;

\ the Vital Product Data for sample ethernet card looks like this:
\
\ (Offsets are in decimal.)
\ Offset          Item                               Value
\ 0      Large Resource Type VPD Tag (0x10)        0x90
\ 1      Length                                     0x0009
\ 3      VPD Keyword                               "NA"
\ 5      Length                                     6
\ 6      Ethernet Address                          0x080020.??????
\ 12     Small Resource Type End Tag (0xf)          0x79
\ 13     Data (nominally checksum)                 0
\
\

: uw@ ( adr -- w ) dup c@ swap 1+ c@ swap bwjoin ;

: set-vpdp-value ( -- ) \ set pointer to vpd
  enable-cheer-cmd-reg
  map-cheer-rom
  enable-cheer-rom
  cheer-rombase vpdp-loc + rw@ cheer-rombase + to vpd-addr
;

: unset-vpdp-value ( -- )
  disable-cheer-rom
  unmap-cheer-rom
  0 to vpd-addr
;

: get-macid-header ( -- tag len1 keyword len2 )
  vpd-addr dup c@ ( tag )
  swap 1+ dup uw@ ( tag len1 )
  swap 2+ dup uw@ ( tag len1 keyword )
  swap 2+ c@      ( tag len1 keyword len2 )
;
```

≡ 13

```
: verify-macid-header-ok? ( tag len1 keyword len2 -- ok? )
  h# 6 <> if 2drop drop false exit then
  ( tag len1 keyword )
  ( "NA" ) h# 4e41 <> if 2drop false exit then
  ( tag len1 )
  h# 9 <> if drop false exit then
  h# 90 <> if false else true then
;
6 buffer: my-loc-mac-addr

: loc-mac-prop ( addr -- )
  6 encode-bytes " local-mac-address" property
;
: make-loc-mac ( -- ) my-loc-mac-addr loc-mac-prop ;
: get-macid ( -- valid? )
  get-macid-header
  verify-macid-header-ok? if
    vpd-addr 6 + my-loc-mac-addr 6 cmove
    true
  else
    diagnostic-mode? if
      ." Wrong Vital Product Data/Network Address header" cr
    then
    false
  then
;
: make-macid ( -- )
  get-macid if
    make-loc-mac
  then
;

: create-macid ( -- )
  set-vpdp-value
  make-macid
  unset-vpdp-value
;
create-hme-attributes \ Create ENET port device node.
create-macid

...
```

```

...

headers
: do-map-in ( offset slot# #bytes -- virtual ) " map-in" $call-parent ;
: do-map-out ( adr len -- ) " map-out" $call-parent ;
: do-dma-map-in ( vaddr n cache? -- devaddr ) " dma-map-in" $call-parent ;
: do-dma-map-out ( vaddr devaddr n -- ) " dma-map-out" $call-parent ;
: do-dma-alloc ( size -- addr ) " dma-alloc" $call-parent ;
: do-dma-free ( addr size -- ) " dma-free" $call-parent ;

: do-dma-sync ( virt-adr dev-adr size -- )
  " dma-sync" [' ] $call-parent catch if
    3drop 2drop
  then
;

...
...

0 instance value obp-tftp          \ Contain ihandle of TFTP package.

: init-obp-tftp ( -- okay? )
  " obp-tftp" find-package if      ( phandle )
    my-args rot open-package      ( ihandle )
  else 0
  then
  dup to obp-tftp                  ( ihandle | 0 )
  dup 0= if
    ." Can't open OBP standard TFTP package" cr
  then
;

...
...

external
: read ( buf len -- actual-len )
  ...

;

```

```
: write ( buf len -- actual-len )
  ...
  ...
;

: seek ( -- okay? )
  ." Unimplemented driver procedure: seek " cr 0
;

: selftest ( -- failed? ) \ Flag 0 if passes test.
  ...
  ...
;

: watch-net ( -- ) \ to watch network activity
  ...
  ...
;

: close ( -- )
  ...
  ...
  obp-tftp ?dup if close-package then
  ...
  ...
;

: open ( -- okay? ) \ return true on successful open
  ...
  ...

  init-obp-tftp 0= if close false exit then
  true
;

: load ( adr -- len )
  " load" obp-tftp $call-method
;

end0
```


This dictionary describes all of the FCodes defined by *IEEE Standard 1275-1994* and supported in the 3.x tokenizer. This dictionary includes the pre-defined FCode words that you can use as part of FCode source code programs. It also includes tokenizer directives and macros. Appendix A, “FCode Reference”, contains a command summary, with words grouped by function.

The words are listed alphabetically in this chapter, sorted by the first alphabetic character in the word’s name. For example, the words `mod` and `*/mod` are adjacent to each other. Words having no alphabetic characters in their names are placed at the beginning of the chapter, in ASCII order.

The boot PROM and tokenizer are case-insensitive (all Forth words are converted to lowercase internally). The only exceptions are literal text, such as text inside " strings and text arguments to the `ascii` command, which are left in the original form. In general, you may use either uppercase or lowercase. By convention, OpenBoot drivers are written in lowercase.

Defining words create a header by calling `external-token`, `named-token`, or `new-token`. See the definitions of these words for more details.

All FCode byte values listed in this chapter are in hexadecimal.

!

stack: (x a-addr --)
code: 72

Stores x at *a-addr*. For more portable code, use `l!` if you explicitly want a 32-bit access. *a-addr* must be aligned as given by `variable`.

See also: `c!`, `w!`, `l!`, `x!`, `rb!`, `rw!`, `rl!`, `rx!`

"

stack: (`x a-addr --`)
 code: `12 len xx xx xx ...`
 generates: `b(") len-byte xx-byte ... xx-byte`

Gathers the immediately following text string or hex data until reaching the terminator `"<whitespace>`.

At execution time, the address and length of the string is left on the stack. For example:

```
" SUNW,new-model" encode-string " model" property
```

You can embed control characters and 8-bit binary numbers in strings. This is similar in principle to the `\n` convention in C, but syntactically tuned for Forth. This feature applies to the string arguments of the words `"` and `. "`

The escape character is `'\"'`. Here is the list of escape sequences:

Table 14-1 Escape Sequences in Text Strings

Syntax	Function
<code>" "</code>	quote (<code>"</code>)
<code>"n</code>	newline
<code>"r</code>	carriage return
<code>"t</code>	tab
<code>"f</code>	formfeed
<code>"l</code>	linefeed
<code>"b</code>	backspace
<code>"!</code>	bell
<code>"^x</code>	control x, where x is any printable character
<code>"(hh hh)</code>	Sequence of bytes, one byte for each pair of hex digits hh . Non-hex characters will be ignored

`"` followed by any other printable character not mentioned above is equivalent to that character.

" (means to start parsing pairs of hexadecimal digits as one or more 8-bit characters in the range 0x00 through 0xFF, delimited by a trailing) and ignoring non-hexadecimal digits between pairs of hexadecimal digits. Both uppercase and lowercase hexadecimal digits are recognized. Since non-hex characters (such as space or comma) are ignored between " (<space> and <space>), these characters make useful delimiters.

Any characters thus recognized are appended to any previous text in the string being assembled. After the) is recognized, text assembly continues until a trailing "<whitespace>.

For example:

```
" This is "(01 32,8e)abc"nA test xyzzy "!! abcdefg" "hijk" ^bl"
          ^^^^^^      ^           ^ ^           ^      ^
          3 bytes      newline      2 bells      "      control b
```

Note – The use of "n for line breaks is discouraged. The preferred method is to use cr, rather than embedding the line break character inside a string. Use of cr results in more accurate display formatting, because Forth updates its internal line counter when cr is executed.

When " is used outside a colon definition, only two interpreted strings of up to 80 characters each can be assembled concurrently. This limitation does not apply in colon definitions.

See also: b(")

#

stack: (ud1 -- ud2)
code: C7

Converts a digit *ud1* in pictured numeric output conversion. Typically used between <# and #>.

See also: fcode-version3

#>

stack: (ud -- str len)
code: C9

Ends pictured numeric output conversion. *str* is the address of the resulting output array. *len* is the number of characters in the output array. *str* and *len* together are suitable for `type`. See `(.)` and `(u.)` for typical usages.

See also: `fcode-version3`

```
stack:( "name ( -- xt )
code:  " 11 FCode (name)
generates: b(')
```

Returns the execution token (*xt*) of the word immediately following `'` in the input stream. `'` should only be used *outside* of definitions. See `b(')`, `[']` for more details.

For example:

```
defer opt-word  ( -- ) ' noop is opt-word
```

```
stack: ( -- )
code: none
```

Causes the compiler/interpreter to ignore subsequent text after the `"("` up to a delimiting `)"`. Note that a space is required after the `(.` Although either `(` or `\` may be used equally well for documentation, by common convention we use `(...)` for stack comments and `\ ...` for all other text comments and documentation.

For example:

```
: 4drop          ( a b c d -- )
  2drop          ( a b )
  2drop          ( )
;
```

```
stack: ( n -- str len )
code: 47 2D 96 9A 49 98 97
generates: dup abs <# u#s swap sign u#>
```

Converts a number into a text string according to the value in `base`. This is the numeric conversion primitive, used to implement display words such as "." If `n` is negative, the first character in the array will be a minus (-) sign.

For example:

```
" CPU boot: show-version ( -- )
  .rom version is " base @ d# 16 base ! ( old-base )
  firmware-version ( old-base version )
  lwsplit (.) type ascii . emit .h cr base !      ( )
```

*

stack: (nu1 nu2 -- prod)
code: 20

`prod` is the arithmetic product of `nu1` times `nu2`. If the result cannot be represented in one stack entry, the least significant bits are kept.

*/

stack: (n1 n2 n3 -- quot)
code: 30 20 31 21

Calculates $n1 * n2 / n3$.

+

stack: (nu1 nu2 -- sum)
code: 1E

`sum` is the arithmetic sum of `nu1` plus `nu2`.

+!

stack: (nu a-addr --)
code: 6C

`nu` is added to the value stored at `a-addr`. This sum replaces the original value at `a-addr`. `a-addr` must be aligned as given by variable.

,

stack: (x --)
code: D3

Reserves one cell of storage in data-space and stores `x` in the cell. The data space pointer must be aligned prior to the execution of `,.`

For example, to create an array containing integers 40004000 23 45 6734:

```
create my-array 40004000 , 23 , 45 , 6734 ,
```

stack: (nu1 nu2 -- diff)
code: 1F

diff is the result of subtracting *nu1* minus *nu2*.

stack: (nu --)
code: 9D

Displays the absolute value of *nu* in a free field format with a leading minus sign if *nu* is negative, and a trailing space.

If the base is hexadecimal, `.` displays the number in unsigned format, since signed hex display is hardly ever wanted. Use `s.` to display signed hex numbers.

See also: `s.`

stack: ([text<">] --)
code: 12 len xx xx ... 90
generates: b(") len text type

This word compiles a text string, delimited by "`<whitespace>`" e.g.
`." hello world" .`

At execution time, the string is displayed. This word is equivalent to using
`" text" type .`

`."` is normally used only in a definition. The text string will be displayed later when that definition is called. You may wish to follow it with `cr` to flush out the text buffer immediately. Use `.(` for any printing to be done immediately.

See also: `.", .(, tokenizer[`

stack: ([text<>] --)
code: 12 len xx xx ... 90

Gathers a text string, delimited by `)`, to be immediately displayed. For example:

```
.( hello world)
```

This word is equivalent to: `" text" type`

Use `.(` to print out text immediately. (You may wish to follow it with a `cr` to flush out the text buffer immediately.) `.(` may be called either inside or outside of definitions; the text is immediately displayed in either case.

Note that during FCode interpretation the string will typically be printed on serial port A, since any frame buffer present may not yet be activated when devices are being probed. Use `."` for any printing to be done when new words are later executed.

See also `."`, `tokenizer[`

`/`
stack: (*n1* *n2* -- quot)
code: 21

Returns *n1* divided by *n2*. An error condition results if the divisor (*n2*) is zero. See `/mod`.

`:`
stack: ("new-name< >" -- colon-sys) (E: ... -- ???)
code: (header) B7

generates: `new-token|named-token|external-token b(:)`

Begins a new definition, terminated by `;` ; Used in the form:

```
: my-newname ... ;
```

Later usage of `my-newname` is equivalent to usage of the contents of the definition.

See `named-token`, `new-token`, and `external-token` for more information on header formats.

`;`
stack: (colon-sys --)

code: C2
generates: b(;

Ends the compilation of a colon definition.

See also: :

<

stack: (n1 n2 -- less_than?)
code: 3A

less_than? is true if *n1* is less than *n2*. *n1* and *n2* are signed integers.

<#

stack: (--)
code: 96

Initializes pictured numeric output conversion. You can use the words:

```
<# # #s hold sign #>
```

to specify the conversion of a number into an ASCII character string stored in right-to-left order. See (.) and (u.) for example usages.

<<

stack: (x1 u -- x2)
code: 27
generates: lshift

x2 is the result of logically left shifting *x1* by *u* places. Zeroes are shifted into the least-significant bits. Synonymous with `lshift`.

For example:

```
: bljoin ( byte.low byte.lowmid byte.highmid byte.high -- 1 )  
  8 << + 8 << + 8 << +  
;
```

<=

stack: (n1 n2 -- less_than_or_equal?)
code: 43

less_than_or_equal? is true if *n1* is less than or equal to *n2*. *n1* and *n2* are signed integers.

<>

stack: (x1 x2 -- not_equal?)
code: 3D

not_equal? is true if *x1* is not equal to *x2*. *x1* and *x2* are signed integers.

=

stack: (x1 x2 -- equal?)
code: 3C

equal? is true if *x1* is equal to *x2*. *x1* and *x2* are signed integers.

>

stack: (n1 n2 -- greater_than?)
code: 3B

greater_than? is true if *n1* is greater than *n2*. *n1* and *n2* are signed integers.

>=

stack: (n1 n2 -- greater_than_or_equal?)
code: 42

greater_than_or_equal? is true if *n1* is greater than or equal to *n2*. *n1* and *n2* are signed integers.

>>

stack: (x1 u -- x2)
code: 28
generates: rshift

x2 is the result of logically right shifting *x1* by *u* places. Zeroes are shifted into the most-significant bits. Use >>a for signed shifting. Synonym for `rshift`.

For example:

```
: wbsplit ( w -- b.low b.high )
  dup h# ff and swap 8 >>
  h# ff and
;
```

?

stack: (a-addr --)

code: 6D 9D
generates: @ .

Fetches and prints the value at the given address. A standard Forth word, primarily used interactively.

@

stack: (a-addr -- x)
code: 6D

x is the value stored at *a-addr*. *a-addr* must be aligned as given by variable.

See also: c@, w@, l@, rb@, rw@, rl@

[

stack: (--)
code: none

Enter interpretation state.

[']

stack: ([old-name< >] -- xt)
code: 11 FCode
generates: b() old-FCode#

' or ['] is used to generate the execution token (*xt*) of the word immediately following the ' or ['].

' should only be used *outside* definitions; ['] may be used either inside or outside definitions. Examples shown usually use ['], since it will always generate the intended result:

```
: my-probe ... ['] my-install is-install ... ;
```

or

```
['] my-install is-install
```

In normal Forth, ' may be used in definitions for the creation of language extensions, but such usage is not applicable to FCode Programs.

\

stack: ([rest-of-line<eol> --)

code: none

Causes the compiler/interpreter to ignore the rest of the input line after the \ . \ can occur anywhere on an input line. Note that a space must be present after \ .

For example:

```
0 value his-ihandle \ place to save someone's ihandle
```

See also: (, (s

l

stack: (--)

code: none

Enter compilation state.

0

stack: (-- 0)

code: A5

Leaves the value 0 on the stack. The only numbers that are not encoded using `b(lit)` are the values -1, 0, 1, 2, or 3. Because these numbers occur so frequently, they are assigned individual FCodes to save space.

0<

stack: (n -- less_than_0?)

code: 36

less_than_0? is true if *n* is less than zero (negative).

0<=

stack: (n -- less_than_or_equal_to_0?)

code: 37

less_than_or_equal_to_0? is true if *n* is less than or equal to zero.

0<>

stack: (n -- not_equal_to_0?)

code: 35

not_equal_to_0? is true if *n* is not zero.

0=

stack: (n -- equal_to_0?)
code: 34

equal_to_0? is true if *n* is zero. This word will invert any flag.

0>

stack: (n -- greater_than_0?)
code: 38

greater_than_0? is true if *n* is greater than zero.

0>=

stack: (n -- greater_than_or_equal_to_0?)
code: 39

greater_than_or_equal_to_0? is true if *n* is greater than or equal to zero.

1

stack: (-- 1)
code: A6

Leaves the value 1 on the stack. The only numbers that are not encoded using `b(lit)` are the values -1, 0, 1, 2, or 3. Because these numbers occur so frequently, these values are assigned individual FCodes to save space.

1+

stack: (nu1 -- nu2)
code: A6 1E
generates: 1 +

nu2 is the result of adding 1 to *nu1*.

1-

stack: (nu1 -- nu2)
code: A6 1F
generates: 1 -

nu2 is the result of subtracting 1 from *nu1*.

-1

stack: (-- -1)
code: A4

Leaves the value -1 on the stack. The only numbers that are not encoded using `b(lit)` are the values -1, 0, 1, 2, or 3. Because these numbers occur so frequently, these values are assigned individual FCodes to save space.

2

stack: (-- 2)
code: A7

Leaves the value 2 on the stack. The only numbers that are not encoded using `b(lit)` are the values -1, 0, 1, 2, or 3. Because these numbers occur so frequently, these values are assigned individual FCodes to save space.

2!

stack: (x1 x2 a-addr --)
code: 77

x1 and *x2* are stored in consecutive locations starting at *a-addr*. *x2* is stored at the lower address. This is equivalent to: `swap over ! cell+ !.`

2*

stack: (x1 -- x2)
code: 59

x2 is the result of shifting *x1* left one bit. A zero is shifted into the vacated bit position. This is equivalent to multiplying by 2.

2+

stack: (nu1 -- nu2)
code: A7 1E
generates: 2 +

nu2 is the result of adding 2 to *nu1*.

2-

stack: (nu1 -- nu2)
code: A7 1F
generates: 2 -

nu2 is the result of subtracting 2 from *nu1*.

2/

stack: (x1 -- x2)
code: 57

x2 is the result of arithmetically shifting *x1* right one bit. The sign is included in the shift and remains unchanged. This is equivalent to dividing by 2.

2@

stack: (a-addr -- x1 x2)
code: 76

x1 and *x2* are two numbers stored in consecutive 32-bit locations starting at *a-addr*. *x2* is the number that was stored at the lower address. This is equivalent to: `dup cell+ @ swap @.`

3

stack: (-- 3)
code: A8

Leaves the value 3 on the stack. The only numbers that are not encoded using `b(lit)` are the values -1, 0, 1, 2, or 3. Because these numbers occur so frequently, these values are assigned individual FCodes to save space.

>>a

stack: (x1 u -- x2)
code: 29

x2 is the result of arithmetically right shifting *x1* by *u* places. The sign bit of *x1* is shifted into the most-significant bits (i.e. sign extend the high bit).

For example:

```
ok ffff.0000 6 >>a .h
fffffc00
ok ffff.0000 6 >> .h
3fffc00
```

abort

stack: (... --) (R: ... --)
code: 2 16

Aborts program execution, clearing the data and return stacks. Control returns to the `ok` prompt. Called after encountering fatal errors.

For example:

```
: probe-loop ( addr -- )
  begin dup l@ drop key? if abort then again
  \ generate a tight probe loop until any key is pressed.
;
```

abs

stack: (n -- u)
code: 2D

u is the absolute value of *n*. If *n* is the maximum negative number, *u* is the same value since the maximum negative number in two's complement notation has no positive equivalent.

accept

stack: (addr len1 -- len2)
code: 88 6D 4B 8A 88 6D 49 88 72
generates: span @ -rot expect span @ swap span !

Get an edited input line, storing it at *addr*.

again

stack: (C: dest-sys --) (--)
code: 13
generates: bbranch - offset

Used in the form `begin...again` to generate an infinite loop. Use a keyboard abort, or `abort` or `exit`, to terminate such a loop. Use this word with caution!

For example:

```
: probe-loop ( addr -- )  
  \ generate a tight probe loop until any key is pressed.  
  begin dup l@ drop key? if abort then again  
;
```

See also: `repeat`, `until`, `while`

alarm

stack: (xt n --)
code: 2 13

Arranges to execute the package method *xt* at periodic intervals of *n* milliseconds (to the best accuracy possible). If *n* is 0, stop the periodic execution of *xt* in the current instance context (leaving unaffected any periodic execution of *xt* that was established in a different instance).

xt is the execution token, as returned by [']. Each time the method is called, the current instance will be set to the same as the current instance at the time that `alarm` was executed and the current instance will then be restored to its previous value afterwards.

xt must be the execution token of a method which neither expects stack arguments nor leaves stack results i.e. whose stack diagram is (--).

A common use of `alarm` would be to implement a console input device's polling function.

For example:

```
: my-checker ( -- ) test-dev-status if user-abort then ;
: install-abort ( -- ) ['] my-checker d# 10 alarm ;
```

alias

```
stack: ("new-name< >old-name< >" --)
code: none
```

`alias` creates a new name, with the exact behavior of some other existing name. The new name can then be used interchangeably with the old name and have the same effect.

The tokenizer does *not* generate any FCode for an `alias` command, but instead simply updates its own lookup table of existing words. Any occurrence of *new-name* causes the assigned FCode value of *old-name* to be generated. One implication is that *new-name* will not appear in the OpenBoot dictionary after the FCode Program is compiled. If this behavior is undesirable, use a colon definition instead.

If the original FCode source text is downloaded and interpreted directly, without being tokenized or detokenized, then any new `alias` words *will* show up and be usable directly.

For example:

```
alias pkg-prop get-package-property
```


align

stack: (--)
code: none

Allocates dictionary bytes as necessary to leave the top of the dictionary variable aligned.

aligned

stack: (n1 -- n1 | a-addr)
code: AE

Increases *n1* as necessary to yield a variable aligned address. If *n1* is already aligned, returns *n*. Otherwise, returns the next higher variable aligned address, *a-addr*.

alloc-mem

stack: (len -- a-addr)
code: 8B

Allocates a buffer of *len* of physical memory that has been aligned to the most stringent requirements of the processor. If successful, returns the associated virtual address. If not successful, `throw` will be called with an appropriate error message as with `abort`.

Memory allocated by `alloc-mem` is not suitable for DMA.

See also: `abort`, `dma-alloc`, `free-mem`, `throw`.

To detect an out-of-memory condition:

```
h# 100 ['] alloc-mem catch ?dup if
  throw
else
  ( virt ) constant my-buff
then
```

allot

stack: (len --)
code: none
generates: 0 max 0 ?do 0 c, loop

Allocates *len* bytes in the dictionary. If the operation fails, a `throw` will be called with an appropriate error message as with `abort`. Error conditions can be detected and handled properly with the phrase `['] allot catch`.

and

stack: (x1 x2 -- x3)
code: 23

x3 is the bit-by-bit logical and of *x1* with *x2*.

ascii

stack: ([text< >] -- char)
code: 10 00 00 00 xx
generates: b(lit) 00 00 00 value

Skips leading space delimiters and puts the ASCII value of the first letter in *text* on the stack. For example:

```
ascii C ( equals hex 43 )  
ascii c ( equals hex 63 )
```

b(")

stack: (-- str len)
code: 12 len xx xx xx ...

An internal word, generated by `"`, `.` and `.` (which leaves a text string on the stack. Never use the word `b(")` in source code.

b(')

stack: (-- xt)
code: 11 FCode#

An internal word, generated by `'` and `[']` which leaves the execution token of the immediately following word on the stack. The FCode for `b(')` should always be followed by the FCode of the desired word. Never use the word `b(')` in source code.

b(:)

stack: (--)
code: B7

An internal word generated by the defining word `:`. Never use the word `b(:)` in source code.

b(;

stack: (--)
code: C2

An internal word generated by `;` to end a colon definition. Never use the word `b(;)` in source code.

base

stack: (-- addr)
code: A0

`base` is the `variable` that contains the current numeric conversion radix to be used when the FCode Program is executing, such as 10 for decimal, 16 for hex, 8 for octal, and so on. Like any variable, `base` leaves its address on the stack.

For example, to print the current value of `base`, use:

```
base @ .d
```

The tokenizer words `decimal` or `hex` are also available for changing the value in `base` as desired. However, these words behave differently depending whether they occur in a definition or outside of a definition.

If `decimal` or `hex` occur *in* a definition, then it will be compiled, later causing a change to the value in `base` when that definition is executed.

If `decimal` or `hex` occur *outside* of a definition, however, then it is interpreted as a command to the tokenizer program itself, thus affecting the interpretation of all subsequent numbers in the text.

Note that changes to `base` affect the numeric base of the User Interface, which can create much confusion for any user (the default value for `base` is hexadecimal). If you *must* change the base, it is recommended that you save and then restore the original base, as in:

```
: .o ( n -- ) \ Print n in octal
  base @ swap ( oldbase n )
  8 base ! .( oldbase )
  base !
;
```

In general, only numeric *output* will be affected by the value in `base`. Fixed numbers in FCode source are interpreted by the tokenizer program. Most numeric input is controlled by `decimal`, `hex`, `d#`, and `h#`, but these words only affect the tokenizer input base; they but do *not* affect the value in `base`.

For example:

```

( assume the initial value in base is 16, i.e. User Interface is in hex )
( no assumptions should be made about the initial tokenizer base )
version1
hex      ( tokenizer in base 16; later execution, using base, in base 16 )
20 .    ( compile decimal 32, later print "20" when FCode executes )
decimal  ( tokenizer is in base 10, later execution is in base 16 )
20 .    ( compile decimal 20, later print "14" since FCode executes in hex )
: TEST ( -- )
    8base !( still compiling in decimal, later change base when TEST executes )
    20 .    ( compiles decimal 20, prints "24" since base was just changed )
    h# 20 .d ( compiles decimal 32, prints "32"; no permanent base changes )
    20 .    ( compiles decimal 20, prints "24" )
;
20 .    ( compile decimal 20, later print "14" )
TEST    ( prints "24 32 24"; has a side-effect of changing the base )
20 .    ( compile decimal 20, later print 24 since TEST changed base )
hex     ( tokenizer is in base 16; later execution, using base, still in base 8 )
20 .    ( compile decimal 32, later print "40" )

```

If this all seems confusing, simply follow these guidelines:

Good: Initially declare `hex` just after `fcode-version2`, and make liberal use of `d#`, `h#`, `.h` and `.d`.

Bad: Changing base either directly or by calling `decimal` or `hex` in a definition.

branch

`stack: (--)`
`code: 13 offset`

An internal word generated by `again`, `repeat`, and `else` which causes an unconditional branch. Never use the word `branch` in source code.

branch

stack: (flag --)
code: 14 offset

An internal word generated by `until`, `while`, and `if` which causes a conditional branch. Never use the word `branch` in source code.

buffer:)

stack: (n --)
code: BD

An internal word generated by the defining word `buffer:` which allocates *n* bytes of storage space. Never use the word `buffer:)` in source code.

because

stack: (sel-- sel)
code: C4

An internal word generated by `case`. Never use the word `b(case)` in source code.

b(constant)

stack: (n --)
code: BA

An internal word generated by the defining word `constant`. Never use the word `b(constant)` in source code.

b(create)

stack: (--)
code: BB

An internal word generated by the defining word `create`. Never use the word `b(create)` in source code.

b(defer)

stack: (--)
code: BC

An internal word generated by the defining word `defer`. Never use the word `b(defer)` in source code.

b(do)

stack: (end start --)
code: 17
generates: +offset

An internal word generated by `do`. Never use the word `b(do)` in source code.

b(?do)

stack: (end start --)
code: 18 +offset

An internal word generated by `?do`. Never use the word `b(?do)` in source code.

begin

stack: C: -- dest-sys) (--)
code: B1
generates: b(<mark)

Marks the beginning of a conditional loop, such as `begin...until`, `begin...while... repeat`, or `begin...again`. See these other words for more details.

behavior

stack: (defer-xt -- contents-xt)
code: DE

This command is used to retrieve the execution contents of a defer word.

A typical use would be to fetch and save the current execution of a defer word, change the behavior temporarily and later restore the original behavior. For example:

```
defer my-func
0 value old-func
['] framus is my-func ...
['] my-func behavior is old-func
['] foo is my-func
... my-func ...
old-func is my-func
```

bell

stack: (-- 0x07)
code: AB

Leave the ASCII code for the bell character on the stack.

b(endcase)

stack: (sel | <nothing>--)
code: C5

An internal word generated by `endcase`. Never use the word `b(endcase)` in source code.

b(endof)

stack: (--)
code: C6 +offset

An internal word generated by `endof`. Never use the word `b(endof)` in source code.

between

stack: (n min max -- min<=n<=max?)
code: 44

min<=n<=max? is true if *n* is between *min* and *max*, inclusive of both endpoints.

See `within` for a different form of comparison.

b(field)

stack: (addr -- addr+offset)
code: BE

An internal word generated by the defining word `field`. Never use the word `b(field)` in source code.

bl

stack: (-- 0x20)
code: A9

Leaves the ASCII code for the space character on the stack.

blank

stack: (addr len --)
code: A9 79
generates: bl fill

Sets *len* bytes of memory beginning at *addr* to the ASCII character value for space (hex 20). No action is taken if *len* is zero.

b(leave)

stack: (--)
code: 1B

An internal word generated by `leave`. Never use the word `b(leave)` in source code.

blink-screen

stack: (--)
code: 1 5B

A defer word, called by the terminal emulator, when it has processed a character sequence that calls for ringing the console bell, but the console input device package has no `ring-bell` method.

`blink-screen` is initially empty, but *must* be loaded with a system-dependent routine in order for the terminal emulator to function correctly. The routine must cause some momentary discernible effect that leaves the screen in the same state as before.

This can be done with `to`, or it can be loaded automatically with `fb1-install` or `fb8-install` (which loads `fb1-blink-screen` or `fb8-blink-screen`, respectively). These default routines invert the screen (twice) by xor-ing every visible pixel. This is quite slow.

A replacement routine simply disables the video for 20 milliseconds or so, i.e.

```
: my-blink-screen ( -- ) video-off 20 ms video-on ;  
...  
  \ load default behaviors with fbx-install, then:  
  ['] my-blink-screen to blink-screen  
;
```

Of course, this example assumes that your display hardware is able to quickly enable and disable the video without otherwise affecting the state.

b(lit)

stack: (-- n)
code: 10 xx xx xx xx

An internal word used to save numbers. Never use the word `b(lit)` in source code.

The only numbers that are not encoded using `b(lit)` are the values -1, 0, 1, 2, or 3. Because these numbers occur so frequently, these values are assigned individual FCodes to save space.

bljoin

stack: (byte.lo byte2 byte3 byte.hi -- quad)
code: 7F

Merges four bytes into a single 32-bit word. Incorrect results may be generated unless the high bytes of each input stack item are zero.

b(loop)

stack: (--)
code: 15 -offset

An internal word generated by `loop`. Never use the word `b(loop)` in source code.

b(+loop)

stack: (n --)
code: 16 -offset

An internal word generated by `+loop`. Never use the word `b(+loop)` in source code.

b(<mark)

stack: (--)
code: B1

An internal word generated by `begin`. Never use the word `b(<mark)` in source code.

body>

stack: (a-addr -- xt)
code: 85

Converts the data field address of a word to its execution token.

>body

stack: (xt -- a-addr)
code: 86

Converts the execution token of a word to its data field address.

b(of)

stack: (testval --)
code: 1C +offset

An internal word generated by of. Never use the word b(of) in source code.

bounds

stack: (start cnt -- start+cnt start)
code: AC

Converts a starting value and count into the form required for a do or ?do loop. For example, to perform a loop 20 times, counting up from 4000 to 401f inclusive, use:

```
4000 20 bounds do...loop
```

This is equivalent to:

```
4020 4000 do...loop
```

b(>resolve)

stack: (--)
code: B2

An internal word generated by repeat and then. Never use the word b(>resolve) in source code.

bs

stack: (-- 0x08)
code: AA

Leaves the ASCII code for the backspace character on the stack.

b(to)

stack: (--)
code: C3

An internal word generated by `to`. Never use the word `b(to)` in source code.

buffer:

stack: (len "new-name< >" --)(E: -- addr)
code: (header)
generates: new-token | named-token | external-token b(buffer:)

Allocates *len* bytes of storage space and creates a name, *new-name*. When *new-name* is executed, it leaves the address of the first byte of the buffer on the stack.

For example:

```
200 buffer: my-name  
my-name ( addr )
```

b(value)

stack: (n --)
code: B8

An internal word generated by the defining word `value`. Never use the word `b(value)` in source code.

b(variable)

stack: (--)
code: B9

An internal word generated by the defining word `variable`. Never use the word `b(variable)` in source code.

bwjoin

stack: (byte.lo byte.hi -- w)
code: B0

Merges two bytes into the low 16-bits of a stack entry whose upper bytes are zeroed. Incorrect results may be generated unless the high bytes of each input stack item are zero.

bxjoin

stack: (b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi -- o)
code: 02 41

Joins 8 bytes to form an octlet. Combines the eight least-significant bits of each operand to form an octlet. Other operand bits are ignored.

byte-load

stack: (addr xt --)
code: 02 3E

Interprets the FCode Program located at *addr*. If *xt* is 1, use *rb@* to read the FCode Program, otherwise use *xt* as the execution token of the definition to be used to read the FCode Program. Continue reading and interpreting the program until *end0* is encountered.

Be aware that *byte-load* does not itself create a new device node as a “container” for any properties and methods defined by the FCode Program that *byte-load* evaluates. If the FCode Program is intended to create such a node, appropriate preparation must be done before and after executing *byte-load*. For example, *new-device* and *set-args* can be executed before and *finish-device* can be executed after *byte-load* is executed.

If *byte-load* is to be executed from the User Interface, additional set up is usually necessary before executing *new-device*.

c!

stack: (x addr --)
code: 75

Stores the least significant 8 bits of *x* in the byte at *addr*.

See also: *rb!*

c,

stack: (byte --)
code: D0

Compiles a byte into the dictionary. *c,* can be used, in conjunction with *create*, to create an array-type structure, as:

```
create yellow 77 c, 23 c, ff c, ff c, 47 c, 22 c, ...
```

Later execution of *yellow* leaves the address of the first byte of the array (the address of the byte “77”) on the stack.

/c

stack: (-- n)
code: 5A

Leaves the number of address units to a byte (i.e. 1) on the stack.

See also: /w, /l, /n

/c*

stack: (nu1 -- nu2)
code: 66
generates: chars

Synonym for `chars`.

c@

stack: (addr -- byte)
code: 71

Fetches the byte at address *addr* and leaves it on top of the stack with the high order bytes filled with zeroes.

See also: `rb@`

ca+

stack: (addr1 index -- addr2)
code: 5E

Increments *addr1* by *index* times the value of /c. `ca+` should be used in preference to `+` when calculating addresses because it more clearly expresses the intent of the operation and is more portable.

ca1+

stack: (addr1 -- addr2)
code: 62

Synonym for `char+`.

\$call-method

```
stack: ( ??? method-str method-len ihandle -- ??? )
code: 02 0E
```

Executes the device interface method *method-str method-len* in the open package instance *ihandle*. The question marks (???) indicate that the contents of the stack before and after the method is called depend on the particular method being called.

For example:

```
: dma-alloc ( #bytes -- vaddr ) " dma-alloc" my-parent $call-method ;
```

See also: `open-package`.

call-package

```
stack: ( ??? xt ihandle -- ??? )
code: 02 08
```

Executes the device interface method *xt* in the open package instance *ihandle*. The question marks (???) indicate that the contents of the stack before and after the method is called depend on the particular method being called.

For example:

Code Example 0-1

```
0 value label-ihandle \ place to save the ihandle of other
package
0 value offset-method \ place to save the xt of found method
: init ( -- )
  my-args " disk-label" $open-package ( ihandle )
  to label-ihandle
  " offset" label-ihandle
  ihandle>phandle ( name-addr name-len phandle )
  find-method if
    to offset-method
  else
    ." Can't find offset method "
  then
;init
: add-offset ( d.byte# -- d.bytes# )
  offset-method label-ihandle call-package
;
```

See also: `find-method`, `open-package`

Scall-parent

stack: (??? method-str method-len -- ???)
code: 02 09

Calls the method named by *method-str method-len* in the parent instance. If the called package has no such method, an error is signaled with `throw`.
Equivalent to:

```
my-parent $call-method
```

The question marks (???) indicate that the contents of the stack before and after the method is called depend on the particular method being called.

For example:

```
: my-dma-alloc ( -- vaddr ) h# 2000 " dma-alloc" $call-parent ;
```

carret

stack: (-- 0x0D)
code: 10 00 00 00 0D
generates: `b(lit) 00 00 00 0x0D`

Leaves the ASCII code for “carriage return” on the stack.

Xcase

stack: (sel -- sel)
code: C4
generates: `b(case)`

Starts a `case` statement that selects its action based on the value of *sel*.
Example of use:

```
: foo ( selector -- )  
  case  
    0 of ." It was 0" endof  
    5 of ." It was 5" endof  
   -2 of ." It was -2" endof  
    ( selector ) ." It was " dup u. \ default clause  
  endcase  
;
```

`of` tests the top of the stack against the selector at run time. If they are the same, the selector is dropped and the following Forth code is executed. If they are not the same, execution continues at the point just following the matching `endof`.

The default clause is optional. When an `of` clause is executed, the selector is *not* on the stack. When a default clause is executed, the selector *is* on the stack. The default clause may use the selector, but must *not* remove it from the stack (it will be automatically removed by `endcase`). If the default case adds to the stack, the selector must be moved to top of stack from which it will be dropped. For example:

```

: bar (selector -- value )
  case
    3 of 21 endof
    4 of 33 endof
    27 swap \ default clause
  endcase
;
```

`case` statements can be used both inside and outside of colon definitions.

catch

```
stack: ( ??? xt -- ??? error-code | ??? false )
code: 02 17
```

Creates a new error handling context and executes `xt` in that context.

If a `throw` (see below) is called during the execution of `xt`,

1. The error handling context is removed
2. The stack depth is restored to the depth that existed prior to the execution of `xt` (not counting the `xt` stack item)
3. The error code that was passed to `throw` is pushed onto the stack
4. `catch` returns

If `throw` is not called during the execution of `xt`, the error handling context is removed and `catch` returns a `false`. The stack effect is otherwise the same as if `xt` were executed using `execute`.

For example:

```
: add-n-check-limit ( n1 n2 n3 -- n )
  + + dup h# 30 > if true throw then
;
: add-me ( n1 n2 n3 -- a b c | n1+n2+n3 )
  ['] add-n-check-limit catch if
    ." Sum exceeds limit " .s
  else
    ." Sum is within limit. Sum = " .s
  then cr
;
```

Note that, given this definition:

```
1 2 3 add-me
```

shows:

```
Sum is within limit. Sum = 6
```

while:

```
10 20 30 add-me
```

may show something like:

```
Sum exceeds limit 50 9 12
```

Note – On a non-zero `throw`, only the stack depth is guaranteed to be the same as before `catch`, not the data stack contents.

cell+

stack: (addr1 -- addr2)
code: 65

Increments *addr1* by the value of */n*. `cell+` should be used in preference to `wal+` or `lal+` when the intent is to address items that are the same size as items on the stack.

cells

stack: (nu1 -- nu2)
code: 69

nu2 is the result of multiplying *nu1* by */n*, the length in bytes of a normal stack item. This is useful for converting an index into a byte offset.

char-height

stack: (-- height)
code: 01 6C

A value, containing the standard height (in pixels) for all characters to be drawn. This number, when multiplied by `#lines`, determines the total height (in pixels) of the active text area.

This word *must* be set to the appropriate value if you wish to use *any* `fb1-` or `fb8-` utility routines or `>font`. This can be done with `to`, but is normally done by calling `set-font`.

chars

stack: (nu1 -- nu2)
code: 66

nu2 is the result of multiplying *nu1* by */c*, the length in bytes of a byte. This is useful for converting an index into a byte offset.

char-width

stack: (-- width)
code: 01 6D

A value, containing the standard width (in pixels) for all characters to be drawn. This number, when multiplied by `#columns`, determines the total width (in pixels) of the active text area.

This word *must* be set to an appropriate value if you want to use *any* `fb1-` or `fb8-` utility routines. This can be done with `to`, but is normally done by calling `set-font`.

child

stack: (parent-phandle -- child-phandle)
code: 02 3B

Returns the phandle of the package that is the first child of the package *parent-phandle*.

child returns zero if the package *parent-phandle* has no children.

You will generally use *child*, together with *peer*, to enumerate (possibly recursively) the children of a particular device. One common use could be for bus adapter device drivers to use the phrase `my-self ihandle>phandle` to develop the *parent-phandle* argument.

For example:

```
: my-children ( -- ) \ shows phandles of all children
  my-self ihandle>phandle child ( first-child )
  begin ?dup while dup .h peer repeat
;
```

close-package

stack: (ihandle --)
code: 02 06

Closes the package instance identified by *ihandle* by calling that package's *close* method and then destroying the instance.

For example:

```
: tftp-load-avail? ( -- exist? )
  0 0 " obp-tftp" $open-package ( ihandle )
  dup ihandle>phandle " load" rot
  find-method if drop true else false then
  close-package
;
```

cmove>

stack: (adr1 adr2 len --)
code: 78
generates: move

Copy *len* bytes of an array starting at *adr1* to *adr2*. This word is an alias for *move*.

column#

stack: (-- column#)
code: 01 53

A value, set and controlled by the terminal emulator, that contains the current horizontal position of the text cursor. A value of 0 represents the left-most cursor position of the text window, *not* the left-most pixel of the frame buffer.

column# can (and should) be looked at as needed if your FCode Program is implementing its own set of frame buffer primitives.

For example:

```
: set-column ( column# -- )  
  0 max #columns 1- min to column#  
;
```

See also: window-left.

#columns

stack: (-- columns)
code: 01 51

This is a value that returns the number of columns of text in the text window i.e. the number of characters in a line, to be displayed using the boot PROM's terminal emulator.

#columns *must* be set to a proper value in order for the terminal emulator to function correctly. The `open` method of any package that uses the terminal emulator package must set #columns to the desired width of the text region. This can be done with `to`, or it can be handled automatically as one of the functions performed by `fb1-install` or `fb8-install`.

For example:

```
: set-column (column# -- )  
  0 max #columns 1- in to column#  
;
```

See also: `is-install`, `fb1-install`, `fb8-install`

comp

stack: (addr1 addr2 len -- n)
code: 7A

Compares two strings of length *len* starting at addresses *addr1* and *addr2* and continuing for *len* bytes. *n* is 0 if the arrays are the same. *n* is 1 if the first differing character in the array at *addr1* is numerically greater than the corresponding character in the array at *addr2*. *n* is -1 if the first differing character in the array at *addr1* is numerically less than the corresponding character in the array at *addr2*.

For example:

```
ok " this" drop " that" comp .h
1
ok " thisismy" drop " this" comp .h
0
ok " thin" drop " this" comp .h
ffffffff
```

compile,

stack: (xt --)
code: DD

Compiles the behavior of the word given by *xt*.

[compile]

stack: ([old-name< >] --)
code: none

Compiles the immediately-following command.

constant

stack: (x "new-name< >" --) (E: -- value)
code: (header) BA
generates: new-token | named-token | external-token b(constant)

Creates a named constant. The name is initially created with:

```
108 constant purple
```

where 108 is the desired value for purple.

Later occurrences of `purple` will leave the 108 on the stack. If you wish to change the value of a constant in a program, you should use `value` instead of `constant`.

control

stack: ([text< >] -- char)
code: 10 00 00 00 xx
generates: b(lit) 00 00 00 xx-byte

Causes the compiler/interpreter to interpret the next letter as a control-code. For example:

```
control c ( equals 03 )
```

count

stack: (pstr -- addr len)
code: 84

Converts a packed string into a byte-array format. *pstr* is the address of a packed string, where the byte at address *pstr* is the length of the string and the string itself starts at address *pstr*+1.

Packed strings are generally not used in FCode. Virtually all string operations are in the "addr len" format.

For example:

```
h# 100 alloc-mem constant my-buff  
" This is a string" my-buff pack ( pstr ) count type
```

cpeek

stack: (addr -- false | byte true)
code: 02 20

Tries to read the 8-bit byte at address *addr*. Returns the data and *true* if the access was successful. A *false* return indicates that a read access error occurred.

See also: `rb@`

cpoke

stack: (byte addr -- okay?)
code: 02 23

Attempts to write the 8-bit byte at address *addr*. Returns *true* if the access was successful. A *false* return indicates that a write access error occurred.

Note – `cpoke` may be unreliable on bus adapters that buffer write accesses.

See also: `rb!`

cr

stack: (--)
code: 92

A `defer` word used to terminate the line on the display and go to the next line. The default implementation transmits a carriage return and line feed to the display, clears `#out` and adds 1 to `#line`.

Use `cr` whenever you want to start a new line of output, or to force the display of any previously buffered output text. This forcing is valuable for outputting error messages, to ensure that the error message is sent *before* any system crash.

For example:

```
: show-info ( -- )  
  ." This is the first line of output " cr  
  ." This is the second line of output " cr  
;
```

(cr

stack: (--)
code: 91

Outputs only the carriage return character (`carret`, `0x0D`). The most common use of `(cr` is for reporting the progress of a test that has many steps. By using `(cr` instead of `cr`, the progress report appears on a single line instead of scrolling.

create

stack: ("new-name< >" --) (E: -- addr)
code: (header) BB
generates: new-token | named-token | external-token b(create)

Creates the name *new-name*. When *new-name* is subsequently executed, it returns the address of memory immediately following *new-name* in the dictionary. You can use `create` to build an array-type structure, as:

```
create white 12 c, 08 c, 36 c, 25 c, 27 c, 5 c, ...
```

Later execution of `green` leaves the address of the first byte of the array (here, the address of the byte “12”) on the stack. The returned address will be two-byte aligned.

`create` may *not* be used in definitions in an FCode Program. The common Forth construct `create...does>` is not supported.

d#

```
stack: ( [number< >] -- n )
code: 10 value
generates: b(lit) xx-byte xx-byte xx-byte xx-byte
```

Causes the compiler/interpreter to interpret the next number in decimal (base 10), regardless of any previous settings of `hex`, `decimal` or `octal`. Only the immediately following number is affected, the default numeric base setting is unchanged. For example:

```
hex
d# 100 ( equals decimal 100 )
100   ( equals decimal 256 )
```

See also: `h#`.

d+

```
stack: ( d1 d2 -- d.sum )
code: D8
```

Adds two double numbers, leaving the double sum on the stack.

For example:

```
ok 1234.0000 0056.7800 9abc 3400.009a d+ .s
1234.9abc 3456.789a
```


d-

stack: (d1 d2 -- d.diff)
 code: D9

Subtracts two double numbers, leaving the double result on the stack.

For example:

```
ok 0 6 1 0 d- .s
ffff.ffff 5
ok 4444.8888 aaaa.bbbb 2222.1111 5555.2222 d- .s
2222.7777 5555.9999
```

.d

stack: (n --)
 code: A0 6D 49 10 00 00 00 0A A0 72 9D A0 72
 generates: base @ swap d# 10 base ! . base !

Displays *n* in decimal with a trailing space. The value of `base` is not permanently affected.

decimal

stack: (--)
 code: 10 00 00 00 0A A0 72
 generates: d# 10 base !

If used outside of a definition, commands the tokenizer program to interpret subsequent numbers in decimal (base 10).

If used in a definition, appends the phrase `10 base !` to the FCode Program that is being created thus affecting later numeric output when the FCode Program is executed.

See also: `base`

decode-bytes

stack: (prop-addr1 prop-len1 data-len --
 prop-addr2 prop-len2 data-addr data-len)
 code: none
 generates: >r over r@ + swap r@ - rot r>

Decodes *data-len* bytes from a property value array and returns the remainder of the array and the decoded byte array.

decode-int

stack: (prop-addr1 prop-len1 -- prop-addr2 prop-len2 n)
code: 02 1B

Decodes a number from the beginning of a property value array and returns the remainder of the property value array and the number *n*.

For example:

```
: show-clock-frequency ( -- )
  " clock-frequency" get-inherited-property 0= if
    ." Clock frequency: " decode-int .h cr 2drop
  then
;
```

decode-phys

stack: (prop-addr1 prop-len1 -- prop-addr2 prop-len2 phys.lo ... phys.hi)
code: 01 28

Decodes a unit address from a property value array and returns the remainder of the array and the decoded list of address components. The number of cells in the list *phys.lo ... phys.hi* is determined by the value of the `#address-cells` property of the parent node.

decode-string

stack: (prop-addr1 prop-len1 -- prop-addr2 prop-len2 str len)
code: 02 1C

Decodes a string from the beginning of a property value array and returns the remainder of the property value array and the string *str len*.

For example:

```
: show-model ( -- )
  " model" get-my-property 0= if decode-string type 2drop then
;
```

default-font

```
stack: ( -- fontaddr charwidth charheight #fontbytes #firstchar #chars )
code: 01 6A
```

Returns all the necessary information about the character font that is built into the boot PROM. This font defines the appearance of every character to be displayed. To load this font, simply pass these parameters to `set-font`, with:

```
default-font set-font
```

The actual parameters returned by `default-font` are:

`fontaddr` - The address of the beginning of the built-in font table

`charwidth` - The width of each character in pixels

`charheight` - The height of each character in pixels

`#fontbytes` - The separation (in bytes) between each scan line entry

`#firstchar` - The ASCII value for the first character actually stored in the font table.

`#chars` - The total number of characters stored in the font table.

defer

```
stack: ( "new-name< >" -- ) (E: -- ??? )
```

```
code: (header) bc
```

```
generates: new-token|named-token|external-token b(defer)
```

Creates a command *new-name* that is a `defer` word i.e. a word whose behavior can be altered with `to`. *new-name* is initially created with execution behavior that indicates that it is an uninitialized `defer` word. For example:

```
ok defer blob
ok blob
<--deferred word not initialized
```

Later, this behavior can then be altered to be that of another existing word by placing that second word's execution token on the stack and loading it into *new-name* with `to`. For example:

```
['] foobar to blob
```

`defer` words are useful for generating recursive routines. For example:

```
defer hold2 \ Will execute action2
: action1
...
  hold2 ( really action2 )
... ;
: action2
...
  action1
... ;
' action2 to hold2
```

`defer` words can also be used for creating words with different behaviors depending on your needs. For example:

```
defer .special ( n -- ) \ Print a value, using special techniques
: print-em-all ( -- )
... .special
... .special
... .special
;

( .d prints in decimal
( .h prints in hexadecimal )
( .sp prints in a custom format )
: print-all-styles
['] .d to .special print-em-all
['] .h to .special print-em-all
['] .sp to .special print-em-all
;
```

In FCode source, `defer` cannot appear inside a colon definition.

See also: `behavior`.

delete-characters

stack: (n --)
code: 01 5E

Deletes *n* characters to the right of the cursor.

`delete-characters` is one of the `defer` words of the display device interface. The terminal emulator package executes `delete-characters` when it has processed a character sequence that requires the deletion of characters to the right of the cursor. The cursor position is unchanged, the cursor character and the first *n*-1 characters to the right of the cursor are deleted. All remaining characters to the right of the cursor, including the highlighted character, are moved left by *n* places. The end of the line is filled with blanks.

This word is initially empty, but *must* be loaded with an appropriate routine in order for the terminal emulator to function correctly. This can be done with `to`, or it can be loaded automatically with `fb1-install` or `fb8-install` (which loads `fb1-delete-characters` or `fb8-delete-characters`, respectively).

See also: `fb8-install`.

delete-lines

stack: (n --)
code: 01 60

Deletes *n* lines at and below the cursor line.

`delete-lines` is one of the `defer` words of the display device interface. The terminal emulator package executes `delete-lines` when it has processed a character sequence that requires the deletion of lines of text below the line containing the cursor. All lines below the deleted lines are scrolled upwards by *n* lines, and *n* blank lines are placed at the bottom of the active text area.

Use this word for scrolling, by temporarily moving the cursor to the top of the screen and then calling `delete-lines`.

This word is initially empty, but *must* be loaded with an appropriate routine in order for the terminal emulator to function correctly. This can be done with `to`, or it can be loaded automatically with `fb1-install` or `fb8-install` (which loads `fb1-delete-lines` or `fb8-delete-lines`, respectively).

See also: `fb8-install`.

delete-property

stack: (name-str name-len --)
code: 02 1E

Deletes the property named by *name-str name-len* in the active package, if such a property exists.

For example:

```
: unmap-me ( -- )
  my-reg my-size " map-out" $call-parent
  " address" delete-property
;
```

depth

stack: (-- u)
code: 51

u is the number of entries contained in the data stack, not counting itself. Note that when an FCode Program is called, there could be other items on the stack from the calling program.

depth is especially useful for before/after stack depth checking, to determine if the stack was corrupted by a particular operation.

device-name

stack: (str len --)
code: 02 01

Creates a name property with the given string value. For example:

```
" SUNW,ffb" device-name
```

This is equivalent to using the `name` macro or

```
encode-string " name" property
```

except that `device-name` performs the same function with only 2 bytes of FCode, instead of 10 bytes. This word could be useful for devices with extremely limited FCode space.

See also: `name` .

diagnostic-mode?

stack: (-- diag?)
code: 01 20

FCode Programs can use `diagnostic-mode?` to control the extent of the selftests performed. If `diagnostic-mode?` is true, more extensive tests can be performed and more messages displayed.

For example:

```
diagnostic-mode?  
if    do-extended-tests  
else  do-normal-tests  
then
```

FCode should not generate character output during probing unless `diagnostic-mode?` is true, or unless an error is encountered. Error output during probing typically goes to the system serial port.

`diagnostic-mode?` will return true if any of the following conditions are met:

- `diag-switch?` is true.
- A machine diagnostic switch (system-dependent) is ON.
- Other system-dependent indicators request extensive diagnostics.

digit

stack: (char base -- digit true | char false)
code: A3

If the character *char* is a digit in the specified base, returns the numeric value of that digit under *true*, else returns the character under *false*. Appropriate characters are hex 30-39 (for digits 0-9) and hex 61-66 (for digits a-f), depending on base.

For example:

```
: probe-slot ( slot# -- ) ... ;  
: probe-slots ( addr cnt -- )  
  bounds ?do  
    i c@ d# 16 digit if probe-slot else drop then  
  loop  
;
```

do

stack: (C: -- dodest-sys) (limit start --) (R: -- sys)
 code: 17 +offset
 generates: b(do) +offset

Begins a counted loop in the form `do...loop` or `do...+loop`. The loop index begins at *start*, and terminates based on *limit*. See `loop` and `+loop` for details on how the loop is terminated. The loop is always executed at least once. For example:

```
8 3 do i . loop \ would print 3 4 5 6 7
9 3 do i . 2 +loop \ would print 3 5 7
```

`do` may be used either inside or outside of colon definitions.

?do

stack: (C: -- dodest-sys) (limit start --) (R: -- sys)
 code: 18 +offset
 generates: b(?do) +offset

Begin a counted loop in the form `?do...loop` or `?do...+loop`. The loop index begins at *start*, and terminates based on *limit*. See `loop` and `+loop` for details on how the loop is terminated. Unlike `do`, if *start* is equal to *limit* the loop is executed zero times. For example:

```
8 1 ?do i . loop \ would print 1 2 3 4 5 6 7
2 1 ?do i . loop \ would print 1
1 1 ?do i . loop \ would print nothing
1 1 do i . loop \ would print 1 2 3 4 5 6 7 8 9 ...
...
```

`?do` can be used in place of `do` in nearly all circumstances. `?do` may be used either inside or outside of colon definitions.

draw-character

stack: (char --)
 code: 01 57

A defer word that is called by the terminal emulator in order to display a single character on the screen at the current cursor location.

This word is initially empty, but *must* be loaded with an appropriate routine in order for the terminal emulator to function correctly. This can be done with `to`, or it can be loaded automatically with `fb1-install` or `fb8-install` (which loads `fb1-draw-character` or `fb8-draw-character`, respectively).

draw-logo

stack: (line# addr width height --)
code: 01 61

A defer word that is called by the system to display the power-on logo (the graphic displayed on the left side during power-up, or by banner).

This word is initially empty, but *must* be loaded with an appropriate routine in order for the terminal emulator to function correctly. This can be done with `to`, or it can be loaded automatically with `fb1-install` or `fb8-install` (which loads `fb1-draw-logo` or `fb8-draw-logo`, respectively).

It is possible to pack a custom logo into the FCode PROM and then re-initialize `draw-logo` to output the custom logo instead.

`draw-logo` is called by the system using the following parameters:

line# - The text line number at which to draw the logo.

addr - The address of the logo template to be drawn. In practice, this will always be either the address of the `oem-logo` field in NVRAM, the address of a custom logo in the FCode PROM, or the address of the built-in system logo. In any case, the logo is an array of 64x64 (hex) pixels.

width - The width of the passed-in logo (in pixels).

height - The height of the passed-in logo (in pixels).

drop

stack: (x --)
code: 46

Removes one item from the stack.

2drop

stack: (x1 x2 --)
code: 52

Removes two items from the stack.

3drop

stack: (x1 x2 x3 --)
code: 46 52
generates: drop 2drop

Removes three items from the stack.

dup

stack: (x -- x x)
code: 47

Duplicates the top stack item.

2dup

stack: (x1 x2 -- x1 x2 x1 x2)
code: 53

Duplicates the top two stack items.

3dup

stack: (x1 x2 x3 -- x1 x2 x3 x1 x2 x3)
code: A7 4E A7 4E A7 4E
generates: 2 pick 2 pick 2 pick

Duplicates the top three stack items.

?dup

stack: (x -- 0 | x x)
code: 50

Duplicates the top stack item unless it is zero.

else

stack: (C: orig-sys1 -- orig-sys2) (--)
code: 13 +offset B2
generates: bbranch +offset b(>resolve)

Begin the `else` clause of an `if...else...then` statement. See `if` for more details.

emit

stack: (char --)
code: 8F

A `defer` word that outputs the indicated ASCII character. For example, `h# 41 emit` outputs an “A”, `h# 62 emit` outputs a “b”, `h# 34 emit` outputs a “4”.

emit-byte

stack: (FCode# --)
 code: N
 generates: n

An FCode-only command used to manually output a desired byte of FCode. Use it together with `tokenizer[` as follows:

```
tokenizer[
  44 emit-byte 20 emit-byte
]tokenizer
```

`emit-byte` would be useful, for example, if you wished to generate a new FCode command that the tokenizer did not understand. This command should be used with caution or else an invalid FCode Program will result.

See also: `tokenizer[`

encode+

stack: (prop-addr1 prop-len1 prop-addr2 prop-len2 -- prop-addr3 prop-len3)
 code: 01 12

Merge two property value arrays into a single property value array. The two input arrays must have been created sequentially with no intervening dictionary allocation or other property value arrays having been created. This can be called repeatedly, to create complex, multi-valued property value arrays for passing to `property`.

For example, suppose you wished to create a property named `myprop` with the following information packed sequentially:

```
"size" 2000 "vals" 3 128 40 22
```

This could be written in FCode as follows:

```
: encode-string,num ( addr len number -- )
  >r encode-string
  r> encode-int encode+
;
" size" 2000 encode-string,num
" vals"   3 encode-string,num encode+
128      encode-int      encode+
```

```
40      encode-int      encode+
22      encode-int      encode+
" myprop" property
```

encode-bytes

stack: (data-addr data-len -- prop-addr prop-len)

code: 01 15

Encodes a byte array into a property value array. The external representation of a byte array is the sequence of bytes itself, with no appended null byte.

For example:

```
my-idprom h# 20 encode-bytes " idprom" property
```

encode-int

stack: (n -- prop-addr prop-len)

code: 01 11

Convert an integer into a property value array, suitable for passing as a value to property. For example:

```
1152 encode-int " hres" property
```

encode-phys

stack: (phys.lo ... phys.hi -- prop-addr prop-len)

code: 01 13

Encodes a unit-address into a property value array by property encoding the list of cells denoting a unit address in the order of *phys.hi* followed by the encoding of the component that appears on the stack below *phys.hi*, and so on, ending with the encoding of the *phys.lo* component.

The number of cells in the list *phys.lo ... phys.hi* is determined by the value of the "#address-cells" property of the parent node.

For example:

```
my-address my-space encode-phys " resetloc" property
```

encode-string

stack: (str len -- prop-addr prop-len)
code: 01 14

Converts an ordinary string, such as created by `"`, into a property value array suitable for `property`. For example:

```
" NSL,DTE,AUU" encode-string " authors" property
```

end0

stack: (--)
code: 00

A word that marks the end of an FCode Program. This word must be present at the end of your program or erroneous results may occur.

If you want to use `end0` inside a colon definition, for example in a conditional clause, use something like:

```
: exit-if-version1 fcode-revision h# 20000 < if ['] end0 execute then ;
```

end1

stack: (--)
code: FF

An alternate word for `end0`, to mark the end of an FCode Program. `end0` is recommended.

`end1` is not intended to appear in source code. It is defined as a guard against misprogrammed ROMs. Unprogrammed regions of PROM usually appear as all ones or all zeroes. Having both `0x00` and `0xFF` defined as end codes stops the FCode interpreter if it enters an unprogrammed region of a PROM.

endcase

stack: (C: case-sys --) (sel | <nothing> --)
code: C5
generates: b(endcase)

Marks the end of a case statement. See `case` for more details.

endof

stack: (C: case-sys1 of-sys -- case-sys2) (--)
code: C6 +offset
generates: b(endof) +offset

Marks the end of an `of` clause in a `case` statement. See `case` for more details.

erase

stack: (addr len --)
code: 95 79
generates: 0 fill

Sets *len* bytes of memory beginning at *addr* to zero. No action is taken if *len* is zero.

erase-screen

stack: (--)
code: 01 5A

A `defer` word that is called once during the terminal emulator initialization sequence in order to completely clear all pixels on the display. This word is called just *before* `reset-screen`, so that the user doesn't actually see the frame buffer data until it has been properly scrubbed.

This word is initially empty, but *must* be loaded with an appropriate routine in order for the terminal emulator to function correctly. This can be done with `to`, or it can be loaded automatically with `fb1-install` or `fb8-install` (which loads `fb1-erase-screen` or `fb8-erase-screen`, respectively).

eval

stack: (??? str len -- ???)
code: CD
generates: evaluate

Synonym for `evaluate`.

evaluate

stack: (??? str len -- ???)
code: CD

Executes a string as a sequence of Forth commands. The overall stack effect depends on the commands being executed. For example:

```
" 4000 20 dump" evaluate
```

You can use `evaluate`, like `$find`, to find and execute Forth commands that are not FCodes.

The same cautions apply to `evaluate` as for `$find` in that programs executing Forth commands are likely to encounter portability problems when moved to other systems.

execute

stack: (??? xt -- ???)
code: 1D

Executes the word definition whose execution token is *xt*. An error condition exists if *xt* is not an execution token.

For example:

```
: my-word ( addr len -- )  
  ." Given string is: " type cr  
;  
" great" ['] my-word execute
```

exit

stack: (--) (R: sys --)
code: 33

Compiled in a colon definition. When encountered, execution leaves the current word and returns control to the calling word. If used in a `do` loop must be preceded by `unloop`.

For example:

```
: probe-loop ( addr -- )  
  \ generate a tight probe loop until any key is pressed.  
  begin dup l@ drop key? if drop exit then again  
;
```

See also: `leave`, `unloop`.

expect

stack: (addr len --)
code: 8A

A *defer* word that receives a line of characters from the keyboard and stores them into memory, performing line editing as the characters are typed. Displays all characters actually received and stored into memory. The number of received characters is stored in *span*.

The transfer begins at *addr* proceeding towards higher addresses one byte per character until either a carriage return is received or until *len* characters have been transferred. No more than *len* characters will be stored. The carriage return is not stored into memory. No characters are received or transferred if *len* is zero.

For example:

```
h# 10 buffer: my-name-buff
: hello ( -- )
  ." Enter Your First name " my-name-buff h# 10 expect
  ." Sun Microsystems Welcomes " my-name-buff span @ type cr
;
```

external

stack: (--)
code: none

After issuing *external*, all subsequent definitions are created so that names are later compiled into RAM, regardless of the value of the NVRAM variable *fcode-debug?*. *external* is used to define the package methods that may be called from other software external to the package, and whose names must therefore be present.

external stays in effect until *headers* or *headerless* is encountered.

For example:

```
external
: open ( -- ok? ) ... ;
```


external-token

stack: (--)
code: CA

A token-type that is used to indicate that this word should always be compiled with the name header present. Activated by `external`, all subsequent words are created with `external-token` until deactivated with either `headers` or `headerless`. See `named-token` for more details. This word should never be used in source code.

false

stack: (-- false)
code: A5
generates: 0

Leaves the value for `false` (i.e. zero) on the stack.

fb1-blink-screen

stack: (--)
code: 01 74

The built-in default routine to blink or flash the screen momentarily on a generic 1-bit-per-pixel frame buffer. This routine is loaded into the `defer` word `blink-screen` by calling `fb1-install`.

This routine is invalid unless the FCode Program has called `fb1-install` and has initialized `frame-buffer-adr` to a valid virtual address.

This word is implemented simply by calling `fb1-invert-screen` twice. In practice, this can be quite slow (around one full second). It is quite common for a frame buffer FCode Program to replace `fb1-blink-screen` with a custom routine that simply disables the video for 20 milliseconds or so. For example:

```
: my-blink-screen ( -- ) video-off 20 ms video-on ;  
...  
fb1-install  
...  
['] my-blink-screen to blink-screen
```

fb1-delete-characters

stack: (n --)
code: 01 77

The built-in default routine to delete *n* characters at and to the right of the cursor, on a generic 1-bit-per-pixel frame buffer. This routine is loaded into the defer word `delete-characters` by calling `fb1-install`.

This routine is invalid unless the FCode Program has called `fb1-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

The cursor position is unchanged, the cursor character and the next *n-1* characters to the right of the cursor are deleted, and the remaining characters to the right are moved left by *n* places. The end of the line is filled with blanks.

fb1-delete-lines

stack: (n --)
code: 01 79

The built-in default routine to delete *n* lines, starting with the line below the cursor line, on a generic 1-bit-per-pixel frame buffer. This routine is loaded into the defer word `delete-lines` by calling `fb1-install`.

This routine is invalid unless the FCode Program has called `fb1-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

The *n* lines at and below the cursor line are deleted. All lines above the cursor line are unchanged. The cursor position is unchanged. All lines below the deleted lines are scrolled upwards by *n* lines, and *n* blank lines are placed at the bottom of the active text area.

fb1-draw-character

stack: (char --)
code: 01 70

The built-in default routine for drawing a character on a generic 1-bit-per-pixel frame buffer, at the current cursor location. This routine is loaded into the defer word `draw-character` by calling `fb1-install`.

This routine is invalid unless the FCode Program has called `fb1-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

If `inverse?` is true, then characters are drawn inverted (white-on-black). Otherwise (the normal case) they are drawn black-on-white.

fb1-draw-logo

stack: (line# logoaddr logowidth logoheight --)
code: 01 7A

The built-in default routine to draw the logo on a generic 1-bit-per-pixel frame buffer. This routine is loaded into the defer word `draw-logo` by calling `fb1-install`.

This routine is invalid unless the FCode Program has called `fb1-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

See `draw-logo` for more information on the parameters passed.

fb1-erase-screen

stack: (--)
code: 01 73

The built-in default routine to clear (erase) every pixel in a generic 1-bit-per-pixel frame buffer. This routine is loaded into the defer word `erase-screen` by calling `fb1-install`.

This routine is invalid unless the FCode Program has called `fb1-install` and has initialized `frame-buffer-adr` to a valid virtual address.

All pixels are erased (not just the ones in the active text area). If `inverse-screen?` is `true`, then all pixels are set to 1, resulting in a black screen. Otherwise (the normal case) all pixels are set to 0, resulting in a white screen.

fb1-insert-characters

stack: (n --)
code: 01 76

The built-in default routine to insert *n* blank characters to the right of the cursor, on a generic 1-bit-per-pixel frame buffer. This routine is loaded into the defer word `insert-characters` by calling `fb1-install`.

This routine is invalid unless the FCode Program has called `fb1-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

The cursor position is unchanged, but the cursor character and all characters to the right of the cursor are moved right by *n* places. An error condition exists if an attempt is made to create a line longer than the maximum line size (the value in `#columns`).

fb1-insert-lines

stack: (n --)
code: 01 78

The built-in default routine to insert *n* blank lines below the cursor on a generic 1-bit-per-pixel frame buffer. This routine is loaded into the defer word `insert-lines` by calling `fb1-install`.

This routine is invalid unless the FCode Program has called `fb1-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

The cursor position on the screen is unchanged. The cursor line is pushed down, but all lines above it are unchanged. Any lines pushed off of the bottom of the active text area are lost.

fb1-install

stack: (width height #columns #lines --)
code: 01 7B

This built-in routine installs all of the built-in default routines for driving a generic 1-bit-per-pixel frame buffer. It also initializes most necessary values needed for using these default routines.

`set-font` must be called, and `frame-buffer-adr` initialized, before `fb1-install` is called, because the `char-width` and `char-height` values set by `set-font` are needed when `fb1-install` is executed.

`fb1-install` loads the following defer routines with their corresponding `fb1-(whatever)` equivalents: `reset-screen`, `toggle-cursor`, `erase-screen`, `blink-screen`, `invert-screen`, `insert-characters`, `delete-characters`, `insert-lines`, `delete-lines`, `draw-character`, `draw-logo`.

The following values are initialized:

`screen-width` - set to the value of the passed-in parameter *width* (screen width in pixels)

`screen-height` - set to the value of the passed-in parameter *height* (screen height in pixels)

`#columns` - set to the smaller of the following two: the passed-in parameter *#columns*, and the NVRAM parameter `screen-#columns`

`#lines` - set to the smaller of the following two: the passed-in parameter `#lines`, and the NVRAM parameter `screen-#rows`

`window-top` - set to half of the difference between the total screen height (`screen-height`) and the height of the active text area (`#lines` times `char-height`)

`window-left` - set to half of the difference between the total screen width (`screen-width`) and the width of the active text area (`#columns` times `charwidth`), then rounded down to the nearest multiple of 32 (for performance reasons)

Several internal values used by various `fb1-` routine are also set.

fb1-invert-screen

stack: (--)
code: 01 75

The built-in default routine to invert every visible pixel on a generic 1-bit-per-pixel frame buffer. This routine is loaded into the defer word `invert-screen` by calling `fb1-install`.

This routine is invalid unless the FCode Program has called `fb1-install` and has initialized `frame-buffer-adr` to a valid virtual address.

All pixels are inverted (not just the ones in the active text area).

fb1-reset-screen

stack: (--)
code: 01 71

The built-in default routine to enable a generic 1-bit-per-pixel frame buffer to display data. This routine is loaded into the defer word `reset-screen` by calling `fb1-install`. (`reset-screen` is called just after `erase-screen` during the terminal emulator initialization sequence.)

This word is initially a NOP. Typically, an FCode Program will define a hardware-dependent routine to enable video, and then replace this generic function with:

```
: my-video-enable ( -- ) ... :  
  
fb1-install  
...  
['] my-video-enable to reset-screen
```

fb1-slide-up

stack: (n --)
code: 01 7C

This is a utility routine. It behaves exactly like `fb1-delete-lines`, except that it doesn't clear the lines at the bottom of the active text area. Its only purpose is to scroll the enable plane for frame buffers that have 1-bit overlay and enable planes.

This routine is invalid unless the FCode Program has called `fb1-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

fb1-toggle-cursor

stack: (--)
code: 01 72

The built-in default routine to toggle the cursor location in a generic 1-bit-per-pixel frame buffer. This routine is loaded into the defer word `toggle-cursor` by calling `fb1-install`. The behavior is to invert every pixel in the one-character-size space for the current position of the text cursor.

This routine is invalid unless the FCode Program has called `fb1-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

fb8-blink-screen

stack: (--)
code: 01 84

The built-in default routine to blink or flash the screen momentarily on a generic 8-bit-per-pixel frame buffer. This routine is loaded into the `defer` word `blink-screen` by calling `fb8-install`.

This routine is invalid unless the FCode Program has called `fb8-install` and has initialized `frame-buffer-adr` to a valid virtual address.

This word is implemented simply by calling `fb8-invert-screen` twice. In practice, this can be very slow (several seconds). It is quite common for a frame buffer FCode Program to replace `fb8-blink-screen` with a custom routine that simply disables the video for 20 milliseconds or so. For example:

```
: my-blink-screen ( -- ) video-off 20 ms video-on ;
...
fb8-install
...
['] my-blink-screen to blink-screen
```

fb8-delete-characters

stack: (n --)
code: 01 87

The built-in default routine to delete *n* characters to the right of the cursor, on a generic 8-bit-per-pixel frame buffer. This routine is loaded into the `defer` word `delete-characters` by calling `fb8-install`.

This routine is invalid unless the FCode Program has called `fb8-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

The cursor position is unchanged. The cursor character and the next *n*-1 characters to the right of the cursor are deleted, and the remaining characters to the right are moved left by *n* places. The end of the line is filled with blanks.

fb8-delete-lines

stack: (n --)
code: 01 89

The built-in default routine to delete *n* lines, starting with the line below the cursor line, on a generic 8-bit-per-pixel frame buffer. This routine is loaded into the defer word `delete-lines` by calling `fb8-install`.

This routine is invalid unless the FCode Program has called `fb8-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

The *n* lines at and below the cursor line are deleted. All lines above the cursor line are unchanged. The cursor position is unchanged. All lines below the deleted lines are scrolled upwards by *n* lines, and *n* blank lines are placed at the bottom of the active text area.

fb8-draw-character

stack: (char --)
code: 01 80

The built-in default routine for drawing a character on a generic 8-bit-per-pixel frame buffer, at the current cursor location. This routine is loaded into the defer word `draw-character` by calling `fb8-install`.

This routine is invalid unless the FCode Program has called `fb8-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

If `inverse?` is true, then characters are drawn inverted (white-on-black). Otherwise (the normal case) they are drawn black-on-white.

fb8-draw-logo

stack: (line# addr width height --)
code: 01 8A

The built-in default routine to draw the logo on a generic 8-bit-per-pixel frame buffer. This routine is loaded into the defer word `draw-logo` by calling `fb8-install`.

This routine is invalid unless the FCode Program has called `fb8-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

See `draw-logo` for more information on the parameters passed.

fb8-erase-screen

stack: (--)
code: 01 83

The built-in default routine to clear (erase) every pixel in a generic 8-bit-per-pixel frame buffer. This routine is loaded into the defer word `erase-screen` by calling `fb8-install`.

This routine is invalid unless the FCode Program has called `fb8-install` and has initialized `frame-buffer-adr` to a valid virtual address.

All pixels are erased (not just the ones in the active text area). If `inverse-screen?` is `true`, then all pixels are set to `0xff`, resulting in a black screen. Otherwise (the normal case) all pixels are set to `0`, resulting in a white screen.

fb8-insert-characters

stack: (n --)
code: 01 86

The built-in default routine to insert *n* blank characters to the right of the cursor, on a generic 8-bit-per-pixel frame buffer. This routine is loaded into the defer word `insert-characters` by calling `fb8-install`.

This routine is invalid unless the FCode Program has called `fb8-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

The cursor position is unchanged, but the cursor character and all characters to the right of the cursor are moved right by *n* places. An error condition exists if an attempt is made to create a line longer than the maximum line size (the value in `#columns`).

fb8-insert-lines

stack: (n --)
code: 01 88

The built-in default routine to insert *n* blank lines below the cursor on a generic 8-bit-per-pixel frame buffer. This routine is loaded into the defer word `insert-lines` by calling `fb8-install`.

This routine is invalid unless the FCode Program has called `fb8-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

The cursor position is unchanged. The cursor line is pushed down, but all lines above it are unchanged. Any lines pushed off of the bottom of the active text area are lost.

fb8-install

```
stack: ( width height #columns #lines -- )
code: 01 8B
```

This built-in routine installs all of the built-in default routines for driving a generic 8-bit-per-pixel frame buffer. It also initializes most necessary values needed for using these default routines.

`set-font` must be called, and `frame-buffer-adr` initialized, before `fb8-install` is called, because the `char-width` and `char-height` values set by `set-font` are needed when `fb8-install` is executed.

`fb8-install` loads the following defer routines with their corresponding `fb8-(whatever)` equivalents: `reset-screen`, `toggle-cursor`, `erase-screen`, `blink-screen`, `invert-screen`, `insert-characters`, `delete-characters`, `insert-lines`, `delete-lines`, `draw-character`, `draw-logo`

The following values are initialized:

`screen-width` - set to the value of the passed-in parameter *width* (screen width in pixels)

`screen-height` - set to the value of the passed-in parameter *height* (screen height in pixels)

`#columns` - set to the smaller of the following two: the passed-in parameter *#columns*, and the NVRAM parameter `screen-#columns`

`#lines` - set to the smaller of the following two: the passed-in parameter *#lines*, and the NVRAM parameter `screen-#rows`

`window-top` - set to half of the difference between the total screen height (`screen-height`) and the height of the active text area (`#lines` times `char-height`)

`window-left` - set to half of the difference between the total screen width (`screen-width`) and the width of the active text area (`#columns` times `char-width`), then rounded down to the nearest multiple of 32 (for performance reasons)

Several internal values are also set that are used by various fb8- routines.

fb8-invert-screen

stack: (--)
code: 01 85

The built-in default routine to XOR (with hex 0xFF) every visible pixel on a generic 8-bit-per-pixel frame buffer. This routine is loaded into the defer word `invert-screen` by calling `fb8-install`.

This routine is invalid unless the FCode Program has called `fb8-install` and has initialized `frame-buffer-adr` to a valid virtual address.

All pixels are inverted (not just those in the active text area).

fb8-reset-screen

stack: (--)
code: 01 81

The built-in default routine to enable a generic 8-bit-per-pixel frame buffer to display data. This routine is loaded into the defer word `reset-screen` by calling `fb8-install`. (`reset-screen` is called just after `erase-screen` during the terminal emulator initialization sequence.)

This word is initially a NOP. Typically, an FCode Program will define a hardware-dependent routine to enable video, and then replace this generic function with:

```
: my-video-enable ( -- ) ... :  
  fb8-install  
  ...  
  ['] my-video-enable to reset-screen
```

fb8-toggle-cursor

stack: (--)
code: 01 82

The built-in default routine to toggle the cursor location in a generic 8-bit-per-pixel frame buffer. This routine is loaded into the defer word `toggle-cursor` by calling `fb8-install`. The behavior is to XOR every pixel with 0xFF in the one-character-size space for the current position of the text cursor.

This routine is invalid unless the FCode Program has called `fb8-install` and `set-font` and has initialized `frame-buffer-adr` to a valid virtual address.

fcode-revision

stack: (-- n)
code: 87

Returns a 32-bit number identifying the version of the device interface. The high 16 bits is the major version number and the low 16 bits is the minor version number. The revision of the device interface described by *IEEE Standard 1275-1994* is “3.0”. In a system compatible with that specification, `fcode-revision` will return `0x0003.0000`.

For example:

```
: exit-if-not-1275-1994 ( -- )  
  fcode-revision h# 30000 < if ['] end0 execute then  
;
```

fcode-version1

stack: (--)
code: FD 00
generates: `version1 (null)(reserved)(length)`

This tokenizer macro is used to start FCode programs intended to be compatible with early OpenBoot systems.

`fcode-version1` generates the FCode header for an FCode program (based on tokenizer switches). If the default tokenizer switches are used, `fcode-version1` begins the header with the `version1` FCode as:

```
(fd) version1      ( 1 byte )  
  
(00) null byte    ( 1 byte )  
  
(xxyy) reserved   ( 2 bytes )  
  
(aabbccdd) length ( 4 bytes )
```

The `length` field specifies the total usable length of FCode data, from `version1` to `end0` inclusive. Additional `end0` bytes are appended to the end of the data, if needed, to leave a total length which is evenly divisible by 4. The null byte position may be used to carry a version number or other information.

fcode-version2

stack: (--)
code: F1 00

generates: start1 (null) (reserved) (length)

Starts a version2 FCode program, generating an 8-byte header similar to `fcode-version1`, except that the starting byte is `start1 (f1)` instead of `version1 (fd)`.

For example:

```
fcode-version2

" SUNW,nvsimm" encode-string " name" property

...

end0
```

Caution – FCode programs created with `fcode-version2` will *only* run on OpenBoot 2 or later systems. They will *not* work on OpenBoot 1.0 systems.

error

stack: (--)
code: FC

Displays an “Unimplemented FCode” error message and stops FCode interpretation at the completion of the function whose evaluation resulted in the execution of `error`. All unimplemented FCode numbers resolve to `error` in OpenBoot.

The intended use of `error` is to determine whether or not a particular FCode is implemented, without checking the FCode version number.

For example:

```
: implemented? ( xt -- flag) ['] error <> ;
: my-peer ( prev -- next )
  ['] peer implemented? if
    peer
  else
```

```

        ." peer is not implemented" cr
    then
;

```

field

stack: (E: addr -- addr+offset) (offset size "new-name<>" -- offset+size)

code: (header) BE

generates: new-token | named-token | external-token b(field)

struct and field are used to create named offset pointers into a structure. For each field in the structure, a name is assigned to the location of that field (as an offset from the beginning of the structure).

The structure being described is:

\ size	Bytes	0 - 1
\ flags	Bytes	2 - 5
\ bits	Byte	6
\ key	Byte	7
\ fullname	Bytes	8 - 17
\ initials	Bytes	8 - 9
\ lastname	Bytes	10 - 17
\ age	Bytes	18 - 19

The field definitions are shown below. (The numbers in parentheses show the stack *after* each word is created.)

```

struct          ( 0 )
2 field size    ( 2 ) \ equivalent to: : size      0 + ;
4 field flags   ( 6 ) \ equivalent to: : flags     2 + ;
1 field bits    ( 7 ) \ equivalent to: : bits      6 + ;
1 field key     ( 8 ) \ equivalent to: : key       7 + ;
0 field fullname ( 8 ) \ equivalent to: : fullname  8 + ;
2 field initials ( 10 ) \ equivalent to: : initials  8 + ;
8 field lastname ( 18 ) \ equivalent to: : lastname 10 + ;
2 field age     ( 20 ) \ equivalent to: : age       18 + ;
constant /record ( ) \ equivalent to: 20 constant /record

```

Typical usage of these defined words would be:

```

/record buffer: myrecord    \ Create the "myrecord" buffer

myrecord flags l@          \ get flags data
myrecord key   c@          \ get key data
myrecord size  w@          \ get size data

/record                    \ get total size of the array

```

Note that `struct` is primarily a documentation aid that the initial value of the structure's size (i.e. 0) on the stack.

fill

```

stack: ( addr len byte -- )
code: 79

```

Sets `len` bytes of memory beginning at `addr` to the value `byte`. No action is taken if `len = 0`.

\$find

```

stack: ( name-str name-len -- xt true | name-str name-len false )
code: CB

```

Takes a string from the stack and searches the current search order for it. During normal FCode evaluation, the search order consists of the vocabulary containing the visible methods of the current device node, followed by the Forth vocabulary.

If the word is not found, the original string is left on the stack, with a *false* on top of the stack. If the word is found, the execution token of that word is left on the stack with *true* on top of the stack.

`$find` is an escape hatch, allowing an FCode Program to perform any function that is available in the OpenBoot User Interface but that is not defined as part of the standard FCode interface.

Use `$find` with caution! Different systems or even different versions of OpenBoot may implement different subsets of the User Interface. If your FCode Program depends on a User Interface word, it might not work on some systems.

Example of use:

```
" root-info" $find ( addr len false | xt true )
if execute          \ if found, then do the function
else ( addr len ) type ." was not found!" cr
then
```

find-method

stack: (method-str method-len phandle -- false | xt true)
code: 02 07

Locates the method named by *method-str method-len* in the package *phandle*. Returns *false* if the package has no such method, or *xt* and *true* if the operation succeeds. Subsequently, *xt* can be used with `call-package`.

For example:

```
: tftp-load-avail? ( -- exist? )
  " obp-tftp" find-package if ( phandle )
  " load" rot find-method if ( xt )
  drop true exit
then
then
false
;
```

find-package

stack: (name-str name-len -- false | phandle true)
code: 02 04

Locates a package whose name is given by the string *name-str name-len*. If the package can be located, returns its *phandle* and *true*. Otherwise returns *false*.

The name is interpreted relative to the `/packages` device node. For example, if *name-str name-len* represents the string "disk-label", the package in the device tree at `/packages/disk-label` will be located.

If there are multiple packages with the same name (in the `/packages` node), the *phandle* for the most recently created one is returned.

For example:

```
: tftp-load-avail? ( -- exist? )
  " obp-tftp" find-package if ( phandle )
  " load" rot find-method if ( xt )
```



```

        drop true exit
    then
    then
    false
;

```

finish-device

stack: (--)
code: 01 27

The two words `finish-device` and `new-device` let a single FCode Program declare more than one entry into the device tree. This capability is useful when a single SBus card contains two or more essentially independent devices, to be controlled by two or more separate operating system device drivers.

Typical usage:

```

version1
...driver#1...
finish-device \ terminate device tree entry#1
new-device   \ begin a new device tree entry
...driver#2
finish-device \ terminate device tree entry#2
new-device   \ begin a new device tree entry
...driver#3...
end0

```

There is an implicit `new-device` call at the beginning of an FCode Program (at `version1` or `start1`), and an implicit `finish-device` call at the end of an FCode Program (at `end0`). Thus, FCode Programs that only define a single device and driver will never need to call `finish-device` or `new-device`.

fload

stack: ([filename<cr>] --)
code: none

This command allows FCode text programs to be broken into function blocks for better clarity, portability and re-use. It behaves similarly to the `#include` statement in the C language. Arbitrary nesting of files with `fload` is allowed i.e. an `fload`'d file may include `fload` commands.

When `fload` is encountered, the Tokenizer continues tokenizing the FCode found in the file `filename`. When the filename has been tokenized, tokenizing resumes on the file that called `filename` with `fload`.

For example:

```
fload my-disk-package.fth
```

Note – `fload` commands won't work when downloading text in source-code form. You can either manually merge the files into one larger text file and use `d1` for downloading, or you can tokenize the files first and then download and execute the FCode in binary form.

>font

stack: (char -- addr)
code: 01 6E

This routine converts a character value (ASCII 0-0xFF) into the address of the font table entry for that character. For the normal, built-in font, only ASCII values 0x21-0x7E result in a printable character, other values will be mapped to a font entry for “blank”.

This word is only of interest if you are implementing your own character-drawing routines. Note that `>font` will generate invalid results unless `set-font` has been called to initialize the font table to be used.

fontbytes

stack: (-- bytes)
code: 01 6F

A value, containing the interval between successive entries in the font table. Each entry contains the next scan line bits for the desired character. Each scan line is normally 12 pixels wide, and is stored as one bit per pixel, thus taking 1 1/2 bytes per scan line. The standard value for `fontbytes` is 2, meaning that the next scan line entry is 2 bytes after the previous one (the last 1/2 byte is wasted space).

This word *must* be set to the appropriate value if you wish to use *any* `fb1-` or `fb8-` utility routines or `>font`. This can be done with `to`, but is normally done by calling `set-font`.

The standard value for `fontbytes` is one of the parameters returned by `default-font`.

frame-buffer-adr

```
stack: ( -- addr )
code: 01 62
```

This value returns the virtual address of the beginning of the current frame buffer memory. It *must* be set to an appropriate virtual address (using `to`) in order to use *any* of the `fb1-` or `fb8-` utility routines. It is suggested that this same `value` variable be used in any of your custom routines that require a frame buffer address, although of course you are free to create and use your own variable if you wish.

Generally, you should only map in the frame buffer memory just before you are ready to use it, and unmap it if it is no longer needed. Typically, this means you should do your mapping in your “install” routine, and unmap it in your “remove” routine (see `is-install` and `is-remove`). Here’s some sample code:

```
h# 2.0000 constant /frame \ # of bytes in frame buffer
h# 40.0000 constant foffset \ Location of frame buffer

: video-map ( -- )
  my-address foffset + /frame map-low to frame-buffer-adr
;

: video-unmap ( -- )
  frame-buffer-adr /frame free-virtual
  -1 to frame-buffer-adr \ Flag accidental accesses to a
                          \ now-illegal address
;

: power-on-selftest ( -- )
  video-map
  ( test video memory )
  video-unmap
;
power-on-selftest

: my-install ( -- )
  video-map
  ...
;

: my-remove ( -- )
  video-unmap
  ...
;
```

```
...
['] my-install is-install
['] my-remove is-remove
```

free-mem

stack: (a-addr len --)
code: 8C

Frees up *len* memory allocated by `alloc-mem`. The arguments *a-addr* and *len* must be the same as those used in a previous `alloc-mem` command.

For example:

```
0 value my-string          \ Holds address of temporary
: .upc-string ( addr len -- ) \ convert to uppercase and print.
  dup alloc-mem to my-string          ( addr len )
  tuck my-string swap move           ( len )
  my-string over bounds ?do i c@ upc i c! loop ( len )
  my-string over type                ( len )
  my-string swap free-mem
;
```

free-virtual

stack: (virt size --)
code: 01 05

Destroys an existing mapping and any "address" property.

If the package associated with the current instance has an "address" property whose first value encodes the same address as *virt*, delete that property. In any case, execute the parent instance's `map-out` method with *virt size* as its arguments.

get-inherited-property

stack: (name-str name-len -- true | prop-addr prop-len false)
code: 02 1D

Locates, in the package associated with the current instance or any of its parents, the property whose name is *name-addr name-len*. If the property exists, returns the property value array *prop-addr prop-len* and *false*. Otherwise returns *true*.

The order in which packages are searched is the current instance first, followed by its immediate parent, followed by its parent's parent, and so on. This is useful for properties with default values established by a parent node, with the possibility of a particular child node "overriding" the default value.

For example:

```
: clock-frequency ( -- val.addr len false | true )
  " clock-frequency" get-inherited-property
;
```

get-msecs

stack: (-- n)
code: 01 25

Returns the current value in a free-running system counter. The number returned is a running total, expressed in milliseconds. You can use this for measuring time intervals (by comparing the starting value with the ending value). No assumptions should be made regarding the absolute number returned; only relative interval comparisons are valid.

No assumptions should be made regarding the *precision* of the number returned. In some systems, the value is derived from the system clock, which typically ticks once per second. Thus, the value returned by `get-msecs` on such a system will be seen to increase in jumps of 1000 (decimal), once per second.

For a delay timer of millisecond accuracy, see `ms`.

get-my-property

stack: (name-str name-len -- true | prop-addr prop-len false)
code: 02 1A

Locates, in the package associated with the current instance, the property named by *name-addr name-len*. If the property exists, returns the property value array *val-addr val-len* and *false*. Otherwise returns *true*.

For example:

```
: show-model-name ( -- )
  " model" get-my-property 0= if ( val.addr len )
    ." model name is " type cr
  else ( )
```

```
    ." model property is missing " cr
  then ( )
;
```

get-package-property

stack: (name-str name-len phandle -- true | prop-addr prop-len false)
code: 02 1F

Locates, in the package *phandle*, the property named by *name-addr name-len*. If the property exists, returns the property value array *prop-addr prop-len* and *false*. Otherwise returns *true*.

For example:

```
: show-model-name ( -- )
  my-self ihandle>phandle ( phandle )
  " model" rot get-package-property 0= if ( val.addr len )
    ." model name is " type cr
  else ( )
    ." model property is missing " cr
  then ( )
;
```

get-token

stack: (fcode# -- xt immediate?)
code: DA

Returns the execution token *xt* of the word associated with FCode number *fcode#* and a flag *immediate?* that is *true* if and only if that word will be executed (rather than compiled) when the FCode Evaluator encounters its FCode number while in compilation state.

h# number

stack: (--)
code: 10 xx xx xx xx xx xx xx xx
generates: b(lit) value

Causes the compiler/interpreter to interpret the immediately following number as a hexadecimal number (base sixteen), regardless of any previous settings of *hex*, or *decimal*. Only the immediately following number is affected. The value of *base* is unchanged.

For example:

```
decimal
h# 100 ( equals decimal 256 )
100    ( equals decimal 100 )
```

See also: d#.

.h

stack: (n --)
code: a0 6d 49 10 00 00 00 10 a0 72 9d a0 72
generates: base @ swap d# 16 base ! . base !

Displays *n* in hex (using .) The value of base is not permanently affected.

headerless

stack: (--)
code: none

Causes all subsequent FCode definitions to be created without the name field (the “head”). (See `named-token` and `new-token`.) This is sometimes done to save space in the final FCode PROM, or possibly to make it more difficult to reverse-engineer an FCode Program.

All such headerless words can be used normally in the FCode Program, but cannot be called interactively from the User Interface for testing and development purposes.

Unless PROM space and/or dictionary space is a major consideration, try not using headerless words, because they make debugging more difficult.

`headerless` remains in effect until `headers` or `external` is encountered.

For example:

```
headerless
h# 3 constant reset-scsi
```

headers

stack: (--)
code: none

Causes all subsequent definitions to be saved with the name field (the “head”) intact. This is the initial default behavior.

Note that even normal FCode words (with heads) cannot be called interactively from the User Interface unless the NVRAM parameter `fcode-debug?` has been set to `true` before a system reset.

`headers` remains in effect until `headerless` or `external` is encountered.

For example:

```
headers
: cnt@ ( -- w )
  transfer-count-lo rb@
  transfer-count-hi rb@
  bwjoin
;
```

here

stack: (-- addr)
code: AD

`here` returns the address of the next available dictionary location.

hex

stack: (--)
code: 10 00 00 00 10 a0 72
generates: b(lit) 16 base !

If used outside of a definition, commands the tokenizer program to interpret subsequent numbers in hex (base 16). If used in a definition, changes the value in `base` affecting later numeric output when the FCode Program is executed.

See also: `base`.

hold

stack: (char --)
code: 95

Inserts `char` into a pictured numeric output string. Typically used between `<#` and `#>`.

For example:

```

: .32 ( n -- )
  base @ >r hex
  <# # # # # ascii . hold # # # # #> type
  r> base !
  space
;

```

i

stack: (-- index) (R: sys -- sys)

code: 19

index is a copy of the loop index. May only be used inside of a `do` or `?do` loop.

For example:

```

: simple-loop ( start len -- )
  bounds ?do i .h cr loop
;

```

if

stack: (C: -- orig-sys) (do-next? --)

code: 14 +offset

generates: b?branch +offset

Execute the following code if *do-next?* is true. Used in the form:

```
do-next? if...else...then
```

or

```
do-next? if...then
```

If *do-next?* is true, the words following `if` are executed and the words following `else` are skipped. The `else` part is optional. If *do-next?* is false, words from `if` through `else`, or from `if` through `then` (when no `else` is used), are skipped.

ihandle>phandle

```
stack: ( ihandle -- phandle )
code: 02 0B
```

Returns the *phandle* of the package from which the instance *ihandle* was created. This is often used with `get-package-property` to read the properties of the package corresponding to a given *ihandle*.

For example:

```
: show-parent ( -- )
  my-parent ihandle>phandle " name" rot
  get-package-property 0= if
    ." my-parent is " type cr
  then
;

```

insert-characters

```
stack: ( n -- )
code: 01 5D
```

`insert-characters` is one of the defer words of the display device interface. The terminal emulator package executes `insert-characters` when it has processed a character sequence that calls for opening space for characters to the right of the cursor. Without moving the cursor, `insert-characters` moves the remainder of the line to the right, thus losing the *n* rightmost characters in the line, and fills the *n* vacated character positions with the background color.

This word is initially empty, but *must* be loaded with an appropriate routine in order for the terminal emulator to function correctly. This can be done with `to`, or it can be loaded automatically with `fb1-install` or `fb8-install` (which loads `fb1-insert-characters` or `fb8-insert-characters`, respectively).

insert-lines

```
stack: ( n -- )
code: 01 5F
```

`insert-lines` is one of the defer words of the display device interface. The terminal emulator package executes `insert-lines` when it has processed a character sequence that calls for opening space for lines of text below the

cursor. Without moving the cursor, `insert-lines` moves the cursor line and all following lines down, thus losing the *n* bottom lines. and fills the *n* vacated lines with the background color.

This word is initially empty, but *must* be loaded with an appropriate routine in order for the terminal emulator to function correctly. This can be done with `to`, or it can be loaded automatically with `fb1-install` or `fb8-install` (which loads `fb1-insert-lines` or `fb8-insert-lines`, respectively).

instance

stack: (--)
code: C0

Modifies the next occurrence of `value`, `variable`, `defer` or `buffer:` to create instance-specific data instead of static data. Re-allocates the data each time a new instance of this package is created.

For example:

```
-1 instance value my-chip-reg
```

inverse?

stack: (-- white-on-black?)
code: 01 54

This value is part of the display device interface. The terminal emulator package sets `inverse?` to `true` when the escape sequences that it has processed have indicated that subsequent characters are to be shown with foreground and background colors exchanged, and to `false`, indicating normal foreground and background colors, otherwise.

The `fb1-` and `fb8-` frame buffer support packages draw characters with foreground and background colors exchanged if `inverse?` is `true`, and with normal foreground and background colors if `inverse?` is `false`.

`inverse?` affects the character display operations `draw-character`, `insert-characters`, and `delete-characters`, but not the other operations such as `insert-lines`, `delete-lines` and `erase-screen`.

`inverse-screen?` should be monitored as needed if your FCode Program is implementing its own set of frame buffer primitives.

See also: `inverse-screen?`

inverse-screen?

stack: (-- black?)
code: 01 55

This value is part of the display device interface. The terminal emulator package sets `inverse-screen?` to `true` when the escape sequences that it has processed have indicated that the foreground and background colors are to be exchanged for operations that affect the background, and to `false`, indicating normal foreground and background colors, otherwise.

The `fb1-` and `fb8-` frame buffer support packages perform screen drawing operations other than character drawing operations with foreground and background colors exchanged if `inverse-screen?` is `true`, and with normal foreground and background colors is `false`.

`inverse-screen?` affects background operations such as `insert-lines`, `delete-lines` and `erase-screen`, but not character display operations such as `draw-character`, `insert-characters` and `delete-characters`.

When `inverse-screen?` and `inverse?` are both true, the colors are exchanged over the entire screen, and subsequent characters are not highlighted with respect to the (inverse) background. For exchanged screen colors and highlighted characters, the setting are `inverse-screen? true` and `inverse? false`.

`inverse-screen?` should be monitored as needed if your FCode Program is implementing its own set of frame buffer primitives.

invert

stack: (x1 -- x2)
code: 26

`x2` is the one's complement of `x1` i.e. all the one bits in `x1` are changed to zero, and all the zero bits are changed to one.

For example:

```
: clear-lastbit ( -- )  
  my-reg r1@ 1 not and my-reg r1!  
;
```

See also `0=`.

invert-screen

stack: (--)
code: 01 5C

`invert-screen` is one of the defer words of the display device interface. The terminal emulator package executes `invert-screen` when it has processed a character sequence that calls for exchanging the foreground and background colors (e.g. changing from black-on-white to white-on-black).

`invert-screen` changes all pixels on the screen so that pixels of the foreground color are given the background color, and vice versa, leaving the colors that will be used by subsequent text output unaffected.

This word is initially empty, but *must* be loaded with an appropriate routine in order for the terminal emulator to function correctly. This can be done with `to`, or it can be loaded automatically with `fb1-install` or `fb8-install` (which loads `fb1-invert-screen` or `fb8-invert-screen`, respectively).

is-stack

stack: (param [old-name <>] --)
code: C3 old FCode #

Synonym for `to`.

Creates `open`, `write`, and `draw-logo` methods for display devices.

For any SBus frame buffer that is to be used by the boot PROM before or during booting, `is-install` declares the FCode procedure that should be used to install (i.e. initialize) that frame buffer. Note that this is distinct from any once-only power-on initialization that should be performed during the probing process itself.

is-install

stack: (xt --)
code: 01 1C

Creates `open`, `write`, and `draw-logo` methods for display devices.

For any SBus frame buffer that is to be used by the boot PROM before or during booting, `is-install` declares the FCode procedure that should be used to install (i.e. initialize) that frame buffer. Note that this is distinct from any once-only power-on initialization that should be performed during the probing process itself.

The `is-install` routine and `is-remove` routine should comprise a matched pair that may be performed alternately as many times as needed. Typically, the `is-install` routine performs mapping functions and some initialization, and the `is-remove` performs any cleanup functions and then does a complementary unmapping.

A partial, typical code example follows:

```

version1
...
: power-on ( -- )          \ Once-only, power-on initialization
  map-register
  init-register
  unmap-register
;
...
: map-devices ( -- )      \ Map register and buffer
  map-register
  map-buffer
;
...
: install-me ( -- )       \ Do this to start using this device
  map-devices
  initialize-devices
;
: remove-me ( -- )        \ Do this to stop using this device
  reset-buffers
  unmap-devices
;
...
\ This routine executed during the probe of this FCode
: my-probe ( -- )         \ First, define the routine
  power-on                \ Power-on initialization
  ['] install-me is-install \ Declare "install" routine
  ['] remove-me is-remove  \ Declare "remove" routine
  ['] test-me is-selftest  \ Declare "selftest" routine
; \ End of the defintion
my-probe                  \ Now execute the routine
end0

```

is-remove

```
stack: ( xt -- )
code: 01 1D
```

Creates a `close` method for display devices that will de-allocate a frame buffer that is no longer going to be used. Typical de-allocation would include unmapping memory and clearing buffers. For example:

```
version1
...
: remove-me ( -- ) \ Do this to stop using this device
  reset-buffers
  unmap-devices
;
...
\ This routine executed during the "probe" of this FCode
: my-probe ( -- ) \ First, define the routine
  power-on \ Power-on initialization
  ['] install-me is-install \ Declare "install" routine
  ['] remove-me is-remove \ Declare "remove" routine
  ['] test-me is-selftest \ Declare "selftest" routine
; \ End of the definition
my-probe \ Now, execute this routine
end0
```

The routine loaded with `is-remove` should form a matched pair with the routine loaded with `is-install`. See `is-install` for more details.

is-selftest

stack: (xt --)
code: 01 1E

Creates a `selftest` method for display devices that will perform a self test of the frame buffer. For example:

```

version1
...
: test-me ( -- fail? ) \ self test method
...
;
...
\ This routine executed during the "probe" of this FCode
: my-probe ( -- ) \ First, define the routine
  power-on \ Power-on initialization
  ['] install-me is-install \ Declare "install" routine
  ['] remove-me is-remove \ Declare "remove" routine
  ['] test-me is-selftest \ Declare "selftest" routine
; \ End of the definition
my-probe \ Now, execute this routine
end0

```

This declaration is typically performed in the same place in the code as `is-install` and `is-remove`.

The self test routine should return a status parameter on the stack indicating the results of the test. A zero value indicates that the test passed. Any nonzero value indicates that the self test failed, but the actual meaning for any nonzero value is not specified. (`memory-test-suite` returns a flag meeting these specifications.)

`selftest` is not automatically executed. For automatic testing, devices should perform a quick sanity check as part of the `install` routine. See “selftest” on page 72.

(is-user-word)

stack: (E: ... -- ???) (name-str name-len xt --)
code: 02 14

Creates a Forth word (not a package method) whose name is given by *name-str* *name-len* and whose behavior is given by the execution token *xt* which must refer to a static method. This allows an FCode Program to define new User Interface commands.

For example:

```
" xyz-abort" ' my-abort (is-user-word)
```

j

stack: (-- index) (R: sys -- sys)
code: 1A

index is a copy of the index of the next outer loop. May only be used in a nested `do` or `?do` loop. For example:

```
do
  ...
  do ... j ... loop
  ...
loop
```

Usually, `do` loops should not be nested this deeply inside a single definition. Forth programs are generally more readable if inner loops are defined inside a separate word.

key

stack: (-- char)
code: 8E

A `defer` word that reads the next ASCII character from the keyboard. If no character has been typed since `key` or `expect` was last executed, `key` waits until a new character is typed. All valid ASCII characters can be received. Control characters are not processed by the system for any editing purpose. Characters received by `key` are not displayed.

For example:

```
: continue? ( -- continue? )
  ." Want to Continue? Enter Y/N" key dup emit
  dup ascii Y = ascii y rot = or
;
```

See also: `key?`

key?

stack: (-- pressed?)
code: 8D

A `defer` word returning *true* if a character has been typed on the keyboard since the last time that `key` or `expect` was executed. The keyboard character is not consumed.

Use `key?` to make simple, interruptible infinite loops:

```
begin ... key? until
```

The contents of the loop will repeat indefinitely until any key is pressed.

See also: `key`

l!

stack: (quad qaddr --)
code: 73

The 32-bit value *quad* is stored at location *qaddr*. *qaddr* must be 32-bit aligned.

See also: `r1!`

l,

stack: (quad --)
code: D2

Compile a 32-bit number into the dictionary. The dictionary pointer must be 2-byte-aligned.

For example:

```
\ to create an array containing integers 40004000 23 45 6734  
create my-array 40004000 1, 23 1, 45 1, 6734 1,
```

l@

stack: (qaddr -- quad)
code: 6E

Fetch the 32-bit number stored at *qaddr*. *qaddr* must be 32-bit aligned.

See also: `r1@`

-
- /l**
- stack: (-- n)
code: 5C
- n* is the number of address units to a 32-bit word, typically 4.
- /l***
- stack: (nu1 -- nu2)
code: 68
- nu2* is the result of multiplying *nu1* by */l*. This is the portable way to convert an index into a byte offset.
- <l@**
- stack: (qaddr -- n)
code: 02 42
- Fetch quadlet from *qaddr*, sign-extended.
- la+**
- stack: (addr1 index -- addr2)
code: 60
- Increments *addr1* by *index* times the value of */l*. This is the portable way to increment an address.
- la1+**
- stack: (addr1 -- addr2)
code: 64
- Increments *addr1* by the value of */l*. This is the portable way to increment an address.
- lbflip**
- stack: (quad1 -- quad2)
code: 02 27
- Reverse the bytes in a 32-bit datum.
- lbflips**
- stack: (qaddr len --)
code: 02 28
- Reverse the bytes in each 32-bit datum in the given region.
- The region begins at *qaddr* and spans *len* bytes. The behavior is undefined if *len* is not a multiple of */l*.

lbsplit

stack: (quad -- byte1.lo byte2 byte3 byte4.hi)
code: 7E

Splits a 32-bit datum into four bytes. The high bytes of each stack result are all zeroes.

lcc

stack: (char1 -- char2)
code: 82

char2 is the lower case version of *char1*. If *char1* is not an upper case letter, it is unchanged. For example:

```
ok ascii M lcc emit  
m  
ok
```

See also: upc

leave

stack: (--) (R: sys --)
code: 1B
generates: b(leave)

Transfers execution to just past the next `loop` or `+loop`. The loop is terminated and loop control parameters are discarded. May only be used in a `do` or `?do` loop.

`leave` may appear in other control structures that are nested in the `do` loop structure. More than one `leave` may appear in a `do` loop.

For example:

```
: search-pat ( pat addr len -- found? )  
  rot false swap 2swap ( false pat addr len )  
  bounds ?do ( flag pat )  
    i @ over = if drop true swap leave then  
  loop  
  drop  
;
```

See also: exit, unloop

?leave

stack: (exit? --) (R: sys --)
 code: 14 + offset 1B B2
 generates: if leave then

If *exit?* is *true* (nonzero), *?leave* transfers control to just beyond the next `loop` or `+loop`. The loop is terminated and loop control parameters are discarded. If *exit?* is zero, no action is taken. May only be used in a `do` or `?do` loop.

?leave may appear in other control structures that are nested in the `do` loop structure. More than one *?leave* may appear in a `do` loop.

For example:

```

: show-mem ( vaddr -- ) \ display h# 10 bytes
  dup h# 9 u.r 5 spaces h# 10 bounds do i c@ 3 u.r loop
;
: .mem ( vaddr size -- )
  bounds ?do i show-mem key? ?leave h# 10 +loop
;

```

left-parse-string

stack: (str len char -- R-str R-len L-str L-len)
 code: 02 40

Splits the input string at the first occurrence of delimiter *char*. For example:

```
" test;in;g" ascii ; left-parse-string
```

would leave the address and length of two strings on the stack:

“in;g” and “test”.

The delimiter character may be any ASCII character. Note that if the delimiter is not found in the string, the effect is as if the delimiter was found at the very end. For example:

```
" testing" ascii q left-parse-string
```

would leave on the stack “” and “testing”.

line#

stack: (-- line#)
code: 01 52

A value, set and controlled by the terminal emulator, that contains the current cursor line number. A value of 0 represents the topmost line of available text space — *not* the topmost pixel of the frame buffer.

This word should be monitored as needed if your FCode Program is implementing its own set of frame buffer primitives.

For example:

```
: set-line ( line -- ) 0 max #lines 1- min to line# ;
```

See also: window-top.

#line

stack: (-- a-addr)
code: 94

A variable containing the number of output lines since the last user interaction i.e. since the last `ok` prompt. `#line` is incremented whenever `cr` executes. The value in this variable is used to determine when to pause during long display output, such as `dump`. Its value is reset each time the `ok` prompt displays.

See also: `exit?`

linefeed

stack: (-- 0x0A)
code: 10 00 00 00 0A
generates: `b(lit) 00 00 00 0x0A`

Leaves the ASCII code for the linefeed character on the stack.

#lines

stack: (-- rows)
code: 01 50

`#lines` is a value that is part of the display device interface. The terminal emulator package uses it to determine the height (number of rows of characters) of the text region that it manages. The `fb1-` and `fb8-` frame buffer support packages also use it for a similar purpose.

The value of `#lines` must be set to the desired height of the text region. This can be done with `to`, or it can be handled automatically as one of the functions performed by `fb1-install` or `fb8-install`. The value set by `fbx-install` is the smaller of the passed `#lines` parameter and the `screen-#rows` NVRAM parameter.

For example:

```
: set-line ( line -- ) 0 max #lines 1- min to line# ;
```

loop

stack: (C: dodest-sys --) (--) (R: sys1 -- <nothing> | sys2)
code: 15 -offset
generates: b(loop) -offset

Terminates a `do` or `?do` loop. Increments the loop index by one. If the index was incremented across the boundary between *limit-1* and *limit*, the loop is terminated and loop control parameters are discarded. When the loop is not terminated, execution continues just after the corresponding `do` or `?do`.

For example, the following `do` loop:

```
8 0 do...loop
```

terminates when the loop index changes from 7 to 8. Thus, the loop will iterate with loop index values from 0 to 7, inclusive.

`loop` may be used either inside or outside of colon definitions.

+loop

stack: (C: dodest-sys --) (n --) (R: sys1 -- <nothing> | sys2)
code: 16 -offset
generates: b(+loop) -offset

Terminates a `do` or `?do` loop. Increments the loop index by *n* (or decrements the index if *n* is negative). If the index was incremented (or decremented) across the boundary between *limit-1* and *limit* the loop is terminated and loop control parameters are discarded. When the loop is not terminated, execution continues just after the corresponding `do` or `?do`.

The following `do` loop:

```
8 0 do...2 +loop
```

terminates when the loop index crosses the boundary between 7 and 8. Thus, the loop will iterate with loop index values of 0, 2, 4, 6.

By contrast, a `do` loop created as follows:

```
0 8 do...-2 +loop
```

terminates when the loop index crosses the boundary between -1 and 0. Thus, the loop will iterate with loop index values of 8, 6, 4, 2, 0.

`+loop` may be used either inside or outside of colon definitions.

lpeek

```
stack: ( qaddr -- false | quad true )  
code: 02 22
```

Tries to read the 32-bit word at address *qaddr*. Returns *quad* and *true* if the access was successful. A *false* return indicates that a read access error occurred. *qaddr* must be 32-bit aligned.

lpoke

```
stack: ( quad qaddr -- okay? )  
code: 02 25
```

Tries to write *quad* at address *qaddr*. Returns *true* if the access was successful. A *false* return indicates a write access error. *qaddr* must be 32-bit aligned.

Note - `lpoke` may be unreliable on bus adapters that “buffer” write accesses.

lshift

```
stack: ( x1 u -- x2 )  
code: 27
```

Shifts *x1* left by *u* bit-places. Zero-fills the low bits.

lwflip

stack: (quad1 -- quad2)
code: 02 26

Swaps the doublets in a quadlet.

lwflips

stack: (qaddr len --)
code: 02 37

Swaps the order of the 16-bit words in each 32-bit word in the memory buffer *qaddr len*. *qaddr* must be four-byte-aligned. *len* must be a multiple of 4.

For example:

```
ok h# 12345678 8000 1!  
ok 8000 4 lwflips  
ok 8000 1@ .h  
56781234
```

lwsplit

stack: (quad -- w1.lo w2.hi)
code: 7C

Splits the 32-bit value *quad* into two 16-bit words. The upper bytes of the two generated words are zeroes.

lxjoin

stack: (quad.lo quad.hi -- o)
code: 02 43

Joins two quadlets to form an octlet. Combines the 32 least-significant bits of each operand to form an octlet. Ignores the high-order bits of each operand.

mac-address

stack: (-- mac-str mac-len)
code: 01 A4

Usually used only by the "network" device type, this FCode returns the value for the *Media Access Control*, or MAC address, that this device should use for its own address. The data is encoded as a byte array, generally 6 bytes long.

The value returned by `mac-address` is system-dependent.

See also: "mac-address", "local-mac-address", and "network" in Chapter 7, "Properties" and Chapter 11, "Network Devices".

map-low

```
stack: ( phys.lo ... size -- virt )
code: 01 30
```

Creates a mapping associating the range of physical addresses beginning at *phys.lo ... my-space* and extending for *size* bytes in this device's physical address space with a processor virtual address. Return that virtual address *virt*.

Equivalent to:

```
my-space swap " map-in" $call-parent
```

The number of cells in the list *phys.lo ...* is one less than the number determined by the value of the "#address-cells" property of the parent node.

If the requested operation cannot be performed, `throw` is called with an appropriate error message.

Out-of-memory conditions can be detected and handled with the phrase:
['] map-low catch

See also: map-out

mask

```
stack: ( -- a-addr )
code: 01 24
```

This variable defines which bits out of every 32-bit word will be tested by `memory-test-suite`. To test all 32-bits, set `mask` to all ones with:

```
ffff.ffff mask !
```

To test only the low-order byte out of each word, set the lower bits of `mask` with:

```
0000.00ff mask !
```

Any arbitrary combination of bits can be tested or masked.

max

stack: (*n1 n2* -- *n1 | n2*)
 code: 2F

Returns the greater of *n1* and *n2*.

memory-test-suite

stack: (*addr len* -- *fail?*)
 code: 01 22

Performs a series of tests on the memory beginning at *addr* for *len* bytes. If any of the tests fail, *failed?* is true and a failure message is displayed on a system-dependent diagnostic output device.

The actual tests performed are machine specific and often vary depending on whether *diagnostic-mode?* is true or false. Typically, if *diagnostic-mode?* is true, a message is sent to the console output device giving the name of each test.

The value stored in *mask* controls whether only some or all data lines are tested.

For example:

```
: test-result ( -- )
  frame-buffer-adr my-frame-size memory-test-suite ( failed? )
  encode-int " test-result" property
;
```

See also: *diag-switch?*

min

stack: (*n1 n2* -- *n1 | n2*)
 code: 2E

Returns the lesser of *n1* and *n2*.

mod

stack: (*n1 n2* -- *rem*)
 code: 22

rem is the remainder after dividing *n1* by the divisor *n2*. *rem* has the same sign as *n2* or is zero. An error condition results if the divisor is zero.

***/mod**

stack: (n1 n2 n3 -- rem quot)
code: 30 20 31 2A

Calculates $n1 * n2 / n3$ and returns the remainder and quotient. The inputs, outputs, and intermediate products are all 32-bit. *rem* has the same sign as *n3* or is zero. An error condition results if the divisor is zero.

/mod

stack: (n1 n2 -- rem quot)
code: 2A

rem is the remainder and *quot* is the quotient of *n1* divided by the divisor *n2*. *rem* has the same sign as *n2* or is zero. An error condition results if the divisor is zero.

model

stack: (str len --)
code: 01 19

This is a shorthand word for creating a "model" property. By convention, "model" identifies the model name/number for a SBus card, for manufacturing and field-service purposes. A sample usage would be:

```
" SUNW,501-1623-1" model
```

This is equivalent to:

```
" SUNW,501-1623-1" encode-string " model" property
```

The "model" property is useful to identify the specific piece of hardware (the SBus card), as opposed to the "name" property (since several different but functionally-equivalent cards would have the same "name" property, thus calling the same operating system device driver).

See also: `property`, "model" in Chapter 7, "Properties".

move

stack: (src_addr dest_addr len --)
code: 78

len bytes starting at *src_addr* (through *src_addr+len-1* inclusive) are moved to address *dest_addr* (through *dest_addr+len-1* inclusive). If *len* is zero then nothing is moved.

The data are moved such that the *len* bytes left starting at address *dest_addr* are the same data as was originally starting at address *src_addr*. If *src_addr > dest_addr* then the first byte of *src_addr* is moved first, otherwise the last byte (*src_addr+len-1*) is moved first. Thus, moves between overlapping fields are properly handled.

`move` will perform 16-bit, 32-bit or possibly even 64-bit operations (for better performance) if the alignment of the operands permits. If your hardware requires explicit 8-bit or 16-bit accesses, you will probably wish to use an explicitly-coded `do ... loop` instead.

ms

stack: (n --)
code: 01 26

Delays all execution for at least *n* milliseconds, by executing an empty delay loop for an appropriate number of iterations. The maximum allowable delay will vary from system to system, but is guaranteed to be valid for all values up to at least 1,000,000 (decimal). No other CPU activity takes place during delays invoked with `ms`, although generally this is not a problem for FCode drivers since there is nothing else to do in the meantime anyway. If this word is used excessively, noticeable delays could result.

For example:

```
: probe-loop-wait ( addr -- )  
  \ wait h# 10 ms before doing another probe at the location  
  begin dup l@ drop h# 10 ms key? until drop  
;
```

my-address

```
stack: ( -- phys.lo ... )
code: 01 02
```

Returns the low component(s) of the device's probe address, suitable for use with the `map-in` method, and with `reg` and `encode-phys`. The returned number, along with `my-space`, encodes the address of location 0 of this device in a bus-specific format. The number of cells in the list `phys.lo ...` is one less than the number determined by the value of the `"#address-cells"` property of the parent node.

The OpenBoot PROM automatically sets `my-address` to the correct value before each slot is probed. Usually, this value is used to calculate the location(s) of the device registers, which are then saved as the property value of the `"reg"` property and later accessed with `my-unit`.

For example for a SBus device:

```
fcode-version2
  " audio" encode-string " name" property
  my-address my-space encode-phys          \ SBus Configuration
Space
  0 encode-int encode+ 0 encode-int encode+
  ...
  " reg" property
end0
```

my-args

```
stack: ( -- arg-str arg-len )
code: 02 02
```

Returns the instance argument string `arg-str arg-len` that was passed to the current instance when it was created, if the argument string exists. Otherwise returns with a length of 0.

For example:

```
ok " /obio:TEST-ARGS" open-dev my-args type
TEST-ARGS
```

my-parent

stack: (-- ihandle)
code: 02 0A

Returns the *ihandle* of the instance that opened the current instance. For device driver packages, the relationships of parent/child instances mimic the parent/child relationships in the device tree.

For example for an SBus device:

```
: show-parent ( -- )
  my-parent ihandle>phandle " name" rot
  get-package-property 0= if
    ." my-parent is " type cr
  then
;
```

my-self

stack: (-- ihandle)
code: 02 03

A value word that returns the current instance's *ihandle*. If there is no current instance, the value returned is zero.

For example:

```
: show-model-name ( -- )
  my-self ihandle>phandle ( phandle )
  " model" rot get-package-property 0= if ( val.addr,len )
    ." model name is " type cr
  else ( )
    ." model property is missing " cr
  then ( )
;
```

my-space

stack: (-- phys.hi)
code: 01 03

Returns the high component of the device's probe address representing the device space that this card is plugged into. The meaning of the returned value is bus-specific.

For example for an SBus device:

```
fcode-version1
  " audio" encode-string " name" property
  my-address h# 130.0000 + my-space h# 8 reg
  ...
end0
```

See `my-address` for more details.

my-unit

stack: (-- phys.lo ... phys.hi)
code: 02 0D

Returns the unit address *phys.lo ... phys.hi* of the current instance. The unit address is set when the instance is created, as follows:

- If the *node-name* used to locate the instance's package contained an explicit *unit-address*, that is the instance's unit address. This handles the case of a "wildcard" node with no associated "reg" property.
- Otherwise, if the device node associated with the package from which the instance was created contains a "reg" property, the first component of its "reg" property value is the instance's unit address.
- Otherwise, the instance's unit address is 0 0.

The number of cells in the list `phys.lo ... phys.hi` is determined by the value of the "#address-cells" property of the parent node.

/n

stack: (-- n)
code: 5D

The number of address units in a cell.

/n*

stack: (nu1 -- nu2)
code: 69

Synonym for `cells`.

na+

stack: (addr1 index -- addr2)
code: 61

Increments *addr1* by *index* times the value of */n*.

na+ should be used in preference to *wa+* or *la+* when the intent is to address items that are the same size as items on the stack.

na1+

stack: (*addr1* -- *addr2*)
code: 65

Synonym for *cell+*.

name

stack: (*adr len* --)
code: 1 14 12 04 6E 61 6D 65 1 10
generates: encode-string " name" property

A shorthand word for creating a “name” property, used to match a device node with the appropriate Solaris driver. The “name” declaration is required for booting with Solaris, and should be present in every FCode program. For example:

```
" SUNW,bison" name
```

is equivalent to:

```
" SUNW,bison" encode-string " name" property
```

See also *property*, *device-name*.

See “name” in Chapter 7, “Properties”.

named-token

stack: (--) (F: /FCode-string FCode#/ --)
code: B6

Creates a new, possibly-named FCode function. *named-token* should never be used directly in source code.

negate

stack: (*n1* -- *n2*)
code: 2C

n2 is the negation of *n1*. This is equivalent to 0 swap - .

new-device

stack: (--)
code: 01 1F

Starts a new entry in the device tree. This word is used for creating multiple devices in a single FCode Program.

See also: `finish-device`

new-token

stack: (--) (F: /FCode#/ --)
code: B5

Creates a new unnamed FCode function. `new-token` should never be used directly in source code.

next-property

stack: (previous-str previous-len phandle -- false | name-str name-len true)
code: 02 3D

Returns the name of the property following *previous-string* of *phandle*.

name-string is a null-terminated string that is the name of the property following *previous-string* in the property list for device *phandle*. If *previous-string* is zero or points to a zero-length string, *name-string* is the name of *phandle*'s first property. If there are no more properties after *previous-string* or if *previous-string* is invalid (i.e. names a property which does not exist in *phandle*), *name-string* is a pointer to a zero-length string.

nip

stack: (x1 x2 -- x2)
code: 4D

Removes the second item on the stack.

noop

stack: (--)
code: 7B

Does nothing. This can be used to provide short delays or as a placeholder for patching in other commands later.

not

stack: (x1 -- x2)
code: 26

Synonym for `invert`.

See also: `0=`

\$number

stack: (`addr len -- true | n false`)
code: A2

A numeric conversion primitive that converts a string to a number, according to the current `base` value. An error flag is returned if an inconvertible character is encountered.

For example:

```
ok hex
ok " 123F" $number .s
123f 0
ok " 123n" $number .s
ffffffff
```

of

stack: (`C: case-sys1 -- case-sys2 of-sys`) (`sel of-val -- sel | <nothing>`)
code: 1C +offset
generates: `b(of) +offset`

Begins the next test clause in a case statement. See `case` for more details.

off

stack: (`a-addr --`)
code: 6B

Sets the contents at `a-addr` to *false* (i.e. zero).

offset16

stack: (`--`)
code: CC

Instructs the tokenizer program, and the boot PROM, to expect all further branch offsets to be 16-bit values. This word is automatically generated by some current tokenizers.

Once `offset16` is executed, the offset size remains 16 bits for the duration of the FCode Program; it cannot be set back to 8 bits. Multiple calls of `offset16` have no additional effect. `offset16` is only useful in an FCode Program that begins with `version1`. All other starting tokens (`start0`, `start1`, `start2`, and `start4`) automatically set the offset size to 16 bits.

See also: `fcode-version2`

on

stack: (*a-addr* --)
code: 6A

Set the contents at *a-addr* to *true* (i.e. -1).

open-package

stack: (*arg-str* *arg-len* *phandle* -- *ihandle* | 0)
code: 02 05

Creates an instance of the package identified by *phandle*, saves in that instance an argument string specified by *arg-str* *arg-len*, and invokes the package's `open` method. The parent instance of the new instance is the instance that invoked `open-package`.

Returns the instance handle *ihandle* of the new instance if it can be opened. It returns 0 if the package could not be opened, either because that package has no `open` method or because its `open` method returned *false* indicating an error. In this case, the current instance is not changed.

For example:

```
: test-tftp-open ( -- ok? )
  " obp-tftp" find-package if ( phandle )
    0 0 rot open-package if true else false then
  else
    false
  then
;

```

Sopen-package

stack: (*arg-str* *arg-len* *name-str* *name-len* -- *ihandle* | 0)
code: 02 0F

Similar to using `find-package` `open-package` except that if `find-package` fails, 0 is returned immediately, without calling `open-package`.

The name is interpreted relative to the `/packages` device node. For example, if `name-str name-len` represents the string "disk-label", the package in the device tree at `"/packages/disk-label"` will be located.

If there are multiple packages with the same name (in the `/packages` node), the most recently created one is opened.

For example:

```
0 0 " obp-tftp" $open-package ( ihandle )
```

or

```
stack: ( x1 x2 -- x3 )
code: 24
```

`x3` is the bit-by-bit inclusive-or of `x1` with `x2`.

#out

```
stack: ( -- a-addr )
code: 93
```

A variable containing the current column number on the output device. This is updated by `emit`, `cr` and some other words that modify the cursor position. It is used for display formatting.

For example:

```
: to-column ( column -- ) #out @ - 1 max spaces ;
```

over

```
stack: ( x1 x2 -- x1 x2 x1 )
code: 48
```

The second stack item is copied to the top of the stack.

2over

```
stack: ( x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2 )
code: 54
```

Copies the third and fourth stack items to the stack top.

pack

```
stack: ( str len addr -- pstr )
```

code: 83

Stores the string specified by *str len* as a packed string at the location *addr* returning *pstr* (which is the same address as *addr*). The byte at address *pstr* is the length of the string and the string itself starts at address *pstr+1*. A packed string can contain at most 255 characters.

Packed strings are generally not used in FCode. Virtually all string operations are in the *addr len* format.

For example:

```
h# 20 buffer: my-packed-string
" This is test string " my-packed-string pack
```

parse-2int

stack: (str len -- val.lo val.hi)

code: 01 1B

Converts a “*hi,lo*” string into a pair of values according to the current value in *base*.

If the string does not contain a comma, *val.lo* is zero and *val.hi* is the result of converting the entire string. If either component contains non-numeric characters, according to the value in *base*, the result is undefined.

For example:

```
ok " 4,ff001200" parse-2int .s
ff001200 4
ok " 4" parse-2int .s
0 4
```

peer

stack: (phandle -- phandle.sibling)

code: 02 3C

peer returns the phandle *phandle.sibling* of the package that is the next child of the parent package *phandle*.

If there are no more siblings, *peer* returns 0.

If *phandle* is 0, *peer* returns *phandle* of the root node.

Together with `child`, `peer` lets you enumerate (possibly recursively) the children of a particular device. A common application would be for a device driver to use `child` to determine the phandle of a node's first child, and use `peer` multiple times to determine the handles of the node's other children. For example:

```

: my-children ( -- )
  my-self ihandle>phandle child ( first-child )
  begin ?dup while dup . peer repeat
;

```

pick

stack: (xu ... x1 x0 u -- xu ... x1 x0 xu)
code: 4E

Copies the *u*-th stack value, not including *u* itself, where the remaining stack items have indices beginning with 0. *u* must be between 0 and two less than the total number of elements on the stack (including *u*).

```

0 pick is equivalent to dup    ( n1 -- n1 n1 )
1 pick is equivalent to over  ( n1 n2 -- n1 n2 n1 )
2 pick is equivalent to      ( n1 n2 n3 -- n1 n2 n3 n1 )

```

For the sake of readability, the use of `pick` should be minimized.

property

stack: (prop-addr prop-len name-str name-len --)
code: 01 10

Creates a new property with the specified name and previously prop-encoded value. If there is a current instance, creates the property in the package from which the current instance was created. Otherwise, if there is an active package, creates the property in the active package. If there is neither a current instance nor an active package, the result is undefined.

If a property with the specified name already exists in the active package in which the property would be created, replace its value with the new value.

Properties provide a mechanism for an FCode Program to pass information to an operating system device driver. A property consists of a property name string and a property value array. The name string gives the name of the

property, and the value array gives the value associated with that name. For example, a frame buffer may wish to declare a property named "hres" (for horizontal resolution) with a value of 1152.

The `property` command requires two arrays on the stack — the value array and the name string. The name string is an ordinary Forth string, such as any string created with `"`. This string should be written in lower case, since the property name is stored only after converting uppercase letters, if any, to lower case. For example:

```
" A21-b" encode-string " New_verSION" property
```

is stored as if entered

```
" A21-b" encode-string " new_version" property
```

The value array, however, *must* be in the property value array format. See Chapter 7, “Properties” for more information on creating property value arrays.

All properties created by an FCode Program are stored in a “device tree” by OpenBoot. This tree can then be queried by an operating system device driver, using the Client Interface’s `getprop` or `nextprop` services.

The FCode Program and the operating system device driver may agree on any arbitrary set of names and values to be passed, with virtually no restrictions. Several property names, though, are reserved and have specific meanings. For many of them, a shorthand command also exists that makes the property declaration a bit simpler.

For example:

```
" SUNW,new-model" encode-string model
```

See also: `"name"`, `device-name`, `model`, `reg` and Chapter 7, “Properties” for more information.

r>

stack: (-- x) (R: x --)
code: 31

Removes *x* from the return stack and places it on the stack. See `>r` for restrictions on the use of this word.

For example:

```
: copyout ( buf addr len -- len ) >r swap r@ move r> ;
```

r@

stack: (-- x) (R: x -- x)
code: 32

Places a copy of the top of the return stack on the stack.

For example:

```
: copyout ( buf addr len -- len ) >r swap r@ move r> ;
```

See `>r` for more details.

.r

stack: (n size --)
code: 9E

Converts *n* using the value of `base` and then displays it right-aligned in a field *size* digits wide. Displays a leading minus sign if *n* is negative. A trailing space is *not* displayed.

If the number of digits required to display *n* is greater than *size*, displays all the digits required with no leading spaces in a field as wide as necessary.

For example:

```
: formatted-output ( -- )  
  my-length h# 8 .r ." length" cr  
  my-width h# 8 .r ." width" cr  
  my-depth h# 8 .r ." depth" cr  
;
```

>r

stack: (x --) (R: -- x)
code: 30

Removes *x* from the stack and places it on the top of the return stack.

The return stack is a second stack, occasionally useful as a place to temporarily place numeric parameters i.e. to “get them out of the way” for a little while. For example:

```
: encode-intr ( int-level vector -- )  
  >r sbus-intr>cpu encode-int r> encode-int encode+  
;
```

However, since the return stack is also used by the system for transferring control from word to word (and by `do` loops), improper use of `>r` or `r>` is guaranteed to crash your program. Some restrictions that *must* be observed are:

- All values placed on the return stack in a colon definition must be removed before the colon definition is exited by normal termination, `exit` or `throw`, or else the program will crash.
- No values from the return stack should be removed from in a colon definition unless they were placed there in that definition.
- Entering a `do` loop automatically places values onto the return stack. Therefore,
 - Values placed on the return stack before the loop was started will not be accessible from in the loop.
 - Values placed on the return stack in the loop must be removed before `loop`, `+loop`, or `leave` is encountered.
 - The loop indices *i* or *j* will no longer be valid when additional values have been placed on the return stack in the loop.

rb!

stack: (byte addr --)
code: 02 31

Stores an 8-bit byte to a device register at *addr* with identical bit ordering as the input stack item. Data is stored with a single access operation and flushes any intervening write buffers, so that the data reaches its final destination before the next FCode Function is executed.

For example:

```
my-stat! ( byte -- ) my-stat rb! ;
```

rb@

stack: (addr -- byte)
code: 02 30

Fetches *byte* from the device register at *addr*. Data is read with a single access operation. The result has identical bit ordering as the original register data.

For example:

```
my-stat@ ( -- byte ) my-stat rb@ ;
```

reg

stack: (phys.lo ... phys.hi size --)
code: 01 16

This is a shorthand word for declaring the "reg" property. Typical usage for an SBus device:

```
my-address 40.0000 + my-space 20 reg
```

This declares that the device registers are located at offset 40.0000 through 40.001f in this slot. The following code would accomplish the same thing:

```
my-address 40.0000 + my-space encode-phys
20 encode-int encode+
" reg" property
```

Note that if you need to declare more than one block of register addresses, you *must* repeatedly use `encode-phys`, `encode-int` and `encode+` to build the structure to be passed into the "reg" property.

For example, to declare two register fields at 10.0000-10.00ff and 20.0000-20.037f on a SBus device, use the following:

```
my-address 10.0000 + my-space encode-phys          \ Offset#1
100 encode-int encode+                             \ Merge size#1
my-address 20.0000 + my-space encode-phys encode+  \ Merge offset#2
380 encode-int encode+                             \ Merge size#2
" reg" property
```

See also: `property`, "reg" in Chapter 7, "Properties".

repeat

```
stack: ( C: orig-sys dest-sys -- ) ( -- )
code: 13 -offset b2
generates: bbranch -offset b(>resolve)
```

Terminates a `begin...while...repeat` conditional loop. See `while` for more details.

reset-screen

```
stack: ( -- )
code: 01 58
```

`reset-screen` is one of the `defer` words of the display device interface. The terminal emulator package executes `reset-screen` when it has processed a character sequence that calls for resetting the display device to its initial state. `reset-screen` puts the display device into a state in which display output is visible (e.g. enable video).

This word is initially empty, but *must* be loaded with an appropriate routine in order for the terminal emulator to function correctly. This can be done with `to`, or it can be loaded automatically with `fb1-install` or `fb8-install` (which loads `fb1-reset-screen` or `fb8-reset-screen`, respectively). These words are NOPs, so it is very common to first call `fbx-install` and then to override the default setting for `reset-screen` with:

```
['] my-video-on to reset-screen
```

See also: `blink-screen`

rl!

stack: (quad qaddr --)
code: 02 35

Stores a 32-bit word to a device register at *qaddr* with identical bit ordering as the input stack item. *qaddr* must be 32-bit aligned. Data is stored with a single access operation and flushes any intervening write buffers, so that the data reaches its final destination before the next FCode Function is executed.

For example:

```
: my-reg! ( n -- ) my-reg rl! ;
```

rl@

stack: (qaddr -- quad)
code: 02 34

Fetches a 32-bit word from the device register at *qaddr*. *qaddr* must be 32-bit aligned. Data is read with a single access operation. The result has identical bit ordering as the original register data.

For example:

```
: my-reg@ ( -- n ) my-reg rl@ ;
```

roll

stack: (xu ... x1 x0 u -- xu-1 ... x1 x0 xu)
code: 4F

Removes the *u*-th stack value, not including *u* itself, where the remaining stack items have indices beginning with 0. The *u*-th stack item is then placed on the top of the stack, moving the remaining items down one position. *u* must be between 0 and two less than the total number of elements on the stack (including *u*).

```
0 roll is a null operation
1 roll is equivalent to swap    ( n1 n2 -- n2 n1 )
2 roll is equivalent to rot    ( n1 n2 n3 -- n2 n3 n1 )
3 roll is equivalent to      ( n1 n2 n3 n4 -- n2 n3 n4 n1 )
```

For the sake of readability and performance, minimize your use of `roll`.

rot

stack: (x1 x2 x3 -- x2 x3 x1)
code: 4A

Rotates the top three stack entries, bringing the deepest to the top.

-rot

stack: (x1 x2 x3 -- x3 x1 x2)
code: 4B

Rotates the top three stack entries in the direction opposite from `rot`, putting the top number underneath the other two.

2rot

stack: (x1 x2 x3 x4 x5 x6 -- x3 x4 x5 x6 x1 x2)
code: 56

Rotates the top three pairs of numbers, bringing the third pair to the top of the stack.

rshift

stack: (x1 u -- x2)
code: 28

Shifts `x1` right by `u` bit-places while zero-filling the high bits.

rw!

stack: (w waddr --)
code: 02 33

Stores a 16-bit word to a device register at `waddr`. `waddr` must be 16-bit aligned. Data is stored with a single access operation and flushes any intervening write buffers, so that the data reaches its final destination before the next FCode Function is executed.

The register is stored with identical bit ordering as the input stack item.

For example:

```
: my-count! ( w -- ) my-count rw! ;
```

rw@

stack: (waddr -- w)
code: 02 32

Fetches a 16-bit word from the device register at *waddr*. *waddr* must be 16-bit aligned. Data is read with a single access operation. The result has identical bit ordering as the original register data.

For example:

```
: my-count@ ( -- w ) my-count rw@ ;
```

rx@

stack: (o addr -- o)
code: 02 2E

Fetches an octlet from device register at *oaddr*. Reads data with a single-access operation. The result has identical bit ordering as the original register data..

rx!

stack: (o addr --)
code: 02 2F

Stores an octlet to device register at *oaddr*.

Stores data with a single-access operation and flushes any intervening write buffers so that data reaches its final destination before the next “word” is executed. Stores the result with identical bit ordering as the input stack item.

s"

stack: ([text<">] -- text-str text-len)
code: 12
generates: b(") len-byte xx-byte xx-byte ... xx-byte

Gather the immediately-following string delimited by " . Return the location of the string *text-str text-len*.

Since an implementation is only required to provide two temporary buffers, a program cannot depend on the system’s ability to simultaneously maintain more than two distinct interpreted strings. Compiled strings do not have this limitation, since they are not stored in the temporary buffers.

s.

stack: (n --)
code: 47 2D 96 9A 49 98 97 90 A9 8F
generates: (.) type space

Displays the absolute value of *n* in a free-field format according to the current value of *base*. Displays a trailing space and, if *n* is negative, a leading minus sign. Even if the base is hexadecimal, *n* will be printed in signed format

See also: .

#s

stack: (ud -- 0 0)
code: C8

Converts the remaining digits in pictured numeric output.

.s

stack: (... -- ...)
code: 9F

Displays the contents of the data stack (using .) according to *base*. The top of the stack appears on the right. The contents of the stack are unchanged.

For example:

```
ok 1 2 3 .s
1 2 3
ok . . .
3 2 1
```

sbus-intr>cpu

stack: (sbus-intr# -- cpu-intr#)
code: 01 31

Convert the SBus interrupt level (1-7) to the CPU interrupt level. The mapping performed will be system-dependent.

This word is included because of the possibility that, even on systems that nominally do not support SBus, SBus devices might be used via a bus-to-bus bridge.

screen-height

stack: (-- height)
code: 01 63

A value, containing the total height of the display (in pixels). It can also be interpreted as the number of “lines” of memory.

`screen-height` is an internal value used by the `fb1-` and `fb8-` frame buffer support packages. In particular, this value is used in `fbx-invert`, `fbx-erase-screen`, `fbx-blink-screen` and in calculating `window-top`. `fb1-install` and `fb8-install` set it to the value of their height argument.

This function is included for historical compatibility. There is little reason for an FCode Program to use it. In fact, “standard” FCode Programs are forbidden from altering its value directly.

screen-width

stack: (-- width)
code: 01 64

A value, containing the width of the display (in pixels). It can also be interpreted as the number of pixels (in memory) between one screen location and the next location immediately below it. The latter definition takes precedence if there is a conflict (e.g. there are unused/invisible memory locations at the end of each line).

`screen-width` is an internal value used by the `fb1-` and `fb8-` frame buffer support packages. `fb1-install` and `fb8-install` set it to their width argument.

This function is included for historical compatibility. There is little reason for an FCode Program to use it. In fact, “standard” FCode Programs are forbidden from altering its value directly.

set-args

stack: (arg-str arg-len unit-str unit-len --)
code: 02 3F

Sets the address and arguments of a new device node.

unit-string is a text string representation of a physical address in the address space of the parent device. `set-args` translates *unit-string* to the equivalent numerical representation by executing the parent instance’s

`decode-unit` method, and sets the current instance's probe-address (i.e. the values returned by `my-address` and `my-space`) to that numerical representation.

`set-args` then copies the string *arg-string* to instance-specific storage, and arranges for `my-args` to return the address and length of that copy when executed from the current instance.

`set-args` is typically used just after `new-device`. `new-device` creates and selects a new device node, and `set-args` sets its probe-address and arguments. Subsequently, the device node's properties and methods are created by interpreting an FCode Program with `byte-load` or by interpreting Forth source code.

The empty string is commonly used as the arguments for a new device node. For example:

```
0 0 " 3.0" set-args
```

set-font

stack: (addr width height advance min-char #glyphs --)
code: 01 6B

This routine declares the font table to be used for printing characters on the screen. This routine *must* be called if you wish to use *any* of the `fb1-` or `fb8-` utility routines or `>font`.

Normally, `set-font` is called just after `default-font`. `default-font` leaves the exact set of parameters needed by `set-font` on the stack. This approach allows your FCode Program to inspect and/or alter the default parameters if desired. See `default-font` for more information on these parameters.

set-token

stack: (xt immediate? fcode# --)
code: DB

Assigns the FCode number *fcode#* to the FCode function whose execution token is *xt*, with compilation behavior specified by *immediate?* as follows:

- If *immediate?* is zero, then the FCode Evaluator will execute the function's execution semantics if it encounters that FCode number while in interpretation state, or append those execution semantics to the current definition if it encounters that FCode number while in compilation state.
- If *immediate?* is nonzero, the FCode Evaluator will execute the functions's FCode Evaluation semantics anytime it encounters that FCode number.

sign

stack: (n --)
code: 98

If *n* is negative, appends an ASCII "-" (minus sign) to the pictured numeric output string. Typically used between <# and #>. See (.) for a typical usage.

space

stack: (--)
code: A9 8F
generates: bl emit

Displays a single ASCII space character.

spaces

stack: (cnt --)
code: A5 2F A5 18 +offset A9 8F 15 -offset
generates: 0 max 0 ?do space loop

Displays *cnt* ASCII space characters. Nothing is displayed if *cnt* is zero.

span

stack: (-- a-addr)
code: 88

A variable containing the count of characters actually received and stored by the last execution of `expect`.

For example:

```
h# 10 buffer: my-name-buff
: hello ( -- )
  ." Enter Your First name " my-name-buff h# 10 expect
  ." Sun Microsystems Welcomes " my-name-buff span @ type cr
;
```

start0

stack: (--)
code: F0

start0 may only be used as the first byte of an FCode Program. start0:

- Sets the `spread` value to 0 causing the FCode Evaluator to read successive bytes of the current FCode Program from the same address.
- Establishes the use of 16-bit branches.
- Reads an FCode header from the current FCode Program and either discards it or uses it to verify the integrity of the current FCode program in an implementation-dependent manner.

See also: `fcode-version2`, `start1`, `start2`, `start4`, `version1`

start1

stack: (--)
code: F1

start1 may only be used as the first byte of an FCode Program. start1:

- Sets the `spread` value to 1 causing the FCode Evaluator to read successive bytes of the current FCode Program from addresses one address unit apart.
- Establishes the use of 16-bit branches.
- Reads an FCode header from the current FCode Program and either discards it or uses it to verify the integrity of the current FCode program in an implementation-dependent manner.

See also: `fcode-version2`, `start0`, `start2`, `start4`, `version1`

start2

stack: (--)
code: F2

start2 may only be used as the first byte of an FCode Program. start2:

- Sets the `spread` value to 2 causing the FCode Evaluator to read successive bytes of the current FCode Program from addresses two address units apart.
- Establishes the use of 16-bit branches.
- Reads an FCode header from the current FCode Program and either discards it or uses it to verify the integrity of the current FCode program in an implementation-dependent manner.

See also: `fcode-version2`, `start0`, `start1`, `start4`, `version1`

start4

stack: (--)
code: F3

`start4` may only be used as the first byte of an FCode Program. `start4`:

- Sets the `spread` value to 4 causing the FCode Evaluator to read successive bytes of the current FCode Program from addresses four address units apart.
- Establishes the use of 16-bit branches.
- Reads an FCode header from the current FCode Program and either discards it or uses it to verify the integrity of the current FCode program in an implementation-dependent manner.

See also: `fcode-version2`, `start0`, `start1`, `start2`, `version1`

state

stack: (-- a-addr)
code: DC

A variable containing *true* if the system is in compilation state.

struct

stack: (-- 0)
code: A5
generates: 0

Initializes a `struct...field` structure by leaving a zero on the stack to define the initial offset. See `field` for details.

suspend-fcode

stack: (--)
code: 02 15

Tells the FCode interpreter that the device identification properties for the active package have been declared, and that the interpreter may postpone interpreting the remainder of the package if it so chooses.

If the FCode interpreter postpones (suspends) interpretation, it saves the state of the interpretation process so that interpretation can continue later. Attempts to open a suspended package cause the FCode interpreter to resume and complete the interpretation of that package before executing the package's `open` method.

For example:

```
version1
  " SUNW,my-name" name
  " SUNW,my-model" encode-string " model" property
suspend-fcode
...
end0
```

This feature is intended to save memory space and reduce the system start-up time by preventing the compilation of FCode drivers that are not actually used.

swap

```
stack: ( x1 x2 -- x2 x1 )
code: 49
```

Exchanges the top two stack items.

2swap

```
stack: ( x1 x2 x3 x4 -- x3 x4 x1 x2 )
code: 55
```

Exchanges the top two pairs of stack items.

then

```
stack: ( C: orig-sys -- ) ( -- )
code: B2
generates: b(>resolve)
```

Terminates an `if...then` or an `if...else...then` conditional structure. See `if` for more details.

throw

```
stack: ( ... error-code -- ??? error-code | ... )
code: 02 18
```

Transfers control to the most recent dynamically enclosing error handling context, passing the indicated error code to that handler. *error-code* must be nonzero. If the value of *error-code* is zero, the zero is removed from the stack, but no other action is taken.

See `catch` for an example of use.

to

```
stack: ( param [old-name< >] -- )
code: C3 old-FCode#
generates: b(to) old-FCode#
```

Changes the contents of a value or a defer word:

```
number to name ( for a value )
xt to name ( for a defer word )
```

toggle-cursor

```
stack: ( -- )
code: 01 59
```

`toggle-cursor` is one of the defer words of the display device interface. The terminal emulator package executes `toggle-cursor` when it is about to process a character sequence that might involve screen drawing activity, and executes it again after it has finished processing that sequence. The first execution removes the cursor from the screen so that any screen drawing will not interfere with the cursor, and the second execution restores the cursor, possibly at a new position, after the drawing activity related to that character sequence is finished. `toggle-cursor` is also called once during the terminal emulator initialization sequence.

If the text cursor is on, `toggle-cursor` turns it off. If the text cursor is off, `toggle-cursor` turns it on. (On a bitmapped display, a typical implementation of this function inverts the pixels of the character cell to the right of the current cursor position.)

`toggle-cursor` is initially empty, but must be loaded with an appropriate routine in order for the terminal emulator to function correctly. This can be done with `to`, or it can be loaded automatically with `fb1-install` or `fb8-install` (which load `fb1-toggle-cursor` or `fb8-toggle-cursor`, respectively).

If the display device hardware has internal state (for example color map settings) that might have been changed by external software without firmware's knowledge, that hardware state should be re-established to the state that the firmware driver requires when the cursor is toggled to the "off" state (which indicates that firmware drawing operations are about to begin). This

situation can occur, for example, when an operating system is using a display device, but that operating system uses firmware text output services from time to time, e.g. for critical warning messages.

See also: `to`, `fb1-install`, `fb8-install`.

tokenizer[

stack: (--)
code: none

This is a tokenizer command that ends FCode byte generation and begins interpretation of the following text as tokenizer commands (up to the closing `]tokenizer`). A `tokenizer[...]tokenizer` sequence may be used anywhere in an FCode Program, either in any definition or outside of definitions.

One plausible use for `tokenizer[` would be to generate debugging text during the tokenizing process. (A `cr` flushes the text from the output buffer immediately, which is useful if the tokenizer crashes.) For example:

```
...  
tokenizer[  .( step a)  cr  ]tokenizer  
...  
tokenizer[  .( step b)  cr  ]tokenizer  
...
```

`emit-byte` can be used with `tokenizer[` to output a desired byte of FCode. This would be useful, for example, if you wished to generate a new FCode command that the tokenizer did not understand. For example:

```
...  
tokenizer[  1 emit-byte  27 emit-byte  ]tokenizer  
\ manually output finish-device fcode  
...
```

]tokenizer

stack: (--)
code: none

Ends a tokenizer-only command sequence. See `tokenizer[`.

true

```
stack: ( -- true )
code: A4
generates: -1
```

Leaves the value for the `true` flag (which is -1) on the stack.

tuck

```
stack: ( x1 x2 -- x2 x1 x2 )
code: 4C
```

Copies the top stack item underneath the second stack item.

type

```
stack: ( text-str text-len -- )
code: 90
```

A `defer` word that transfers *text-len* characters to the output beginning with the character at address *text-str* and continuing through *text-len* consecutive addresses. No action is taken if *text-len* is zero.

For example:

```
h# 10 buffer: my-name-buff
: hello ( -- )
  ." Enter Your First name " my-name-buff h# 10 expect
  ." Sun Microsystems Welcomes " my-name-buff span @ type cr
;
```

The output will go either to a frame buffer or to a serial port depending on which is enabled.

u#

```
stack: ( u1 -- u2 )
code: 99
```

The remainder of *u1* divided by the value of `base` is converted to an ASCII character and appended to the output string with `hold`. *u2* is the quotient and is maintained for further processing. Typically used between `<#` and `#>`.

u#>

stack: (u -- str len)
code: 97

Pictured numeric output conversion is ended dropping *u*. *str* is the address of the resulting output array. *len* is the number of characters in the output array. *str* and *len* together are suitable for `type`. See `(.)` and `(u.)` for typical usages.

u#s

stack: (u1 -- u2)
code: 9A

u1 is converted, appending each resultant character into the pictured numeric output string until the quotient is zero (see: `#`). A single zero is added to the output string if *u1* was initially zero. Typically used between `<#` and `#>`. See `(.)` and `(u.)` for typical usages.

This word is equivalent to calling `#` repeatedly until the number remaining is zero.

u.

stack: (u --)
code: 9B

Displays *u* as an unsigned number in a free-field format according to the value in `base`. A trailing space is also displayed.

For example:

```
ok hex -1 u.  
fffffff
```

u<

stack: (u1 u2 -- unsigned-less?)
code: 40

Returns *true* if *u1* is less than *u2* where *u1* and *u2* are treated as unsigned integers.

u<=

stack: (u1 u2 -- unsigned-less-or-equal?)
code: 3F

Returns *true* if *u1* is less than or equal to *u2* where *u1* and *u2* are treated as unsigned integers.

u>

stack: (u1 u2 -- unsigned-greater?)
code: 3E

Returns *true* if *u1* is greater than *u2* where *u1* and *u2* are treated as unsigned integers.

u>=

stack: (u1 u2 -- unsigned-greater-or-equal?)
code: 41

Returns *true* if *u1* is greater than or equal to *u2* where *u1* and *u2* are treated as unsigned integers.

(u.)

stack: (u -- addr len)
code: 96 9A 97
generates: <# u#s u#>

This is a numeric conversion primitive used to implement display words such as `u.` . It converts an unsigned number into a string according to the value in `base`.

For example:

```
ok hex d# -12 (u.) type
ffffffff4
```

u2/

stack: (x1 -- x2)
code: 58

x2 is the result of *x1* logically shifted right one bit. A zero is shifted into the vacated sign bit.

For example:

```
ok -2 u2/ .s
7fffffff
```

um*

stack: (u1 u2 -- ud.prod)
code: D4

Multiplies two unsigned 32-bit numbers yielding an unsigned 64-bit product.

For example:

```
ok hex 3 3 u*x .s
9 0
ok 4 ffff.ffff u*x .s
fffffffc 3
```

um/mod

stack: (ud u -- urem uquot)
code: D5

Divides an unsigned 64-bit number by an unsigned 32-bit number yielding an unsigned 32-bit remainder and quotient

u/mod

stack: (u1 u2 -- urem uquot)
code: 2B

rem is the remainder and *quot* is the quotient after dividing *u1* by *u2*. All values and arithmetic are unsigned. All values are 32-bit.

For example:

```
ok -1 5 u/mod .s
0 33333333
```

unloop

stack: (--) (R: sys --)
code: 89

Discards loop control parameters.

until

stack: (C: dest-sys --) (done? --)
 code: 14 -offset
 generates: b?branch -offset

Marks the end of a begin...until conditional loop. When until is encountered, *done?* is removed and tested. If *done?* is *true*, the loop is terminated and execution continues just after the until. If *done?* is *false*, execution jumps back to just after the corresponding begin.

For example:

```
: probe-loop ( addr -- )
  \ generate a tight 'scope loop until a key is pressed.
  begin dup l@ drop key? until drop
;
```

upc

stack: (char1 -- char2)
 code: 81

char2 is the upper case version of *char1*. If *char1* is not a lower case letter, it is left unchanged.

For example:

```
: continue? ( -- continue? )
  ." Want to Continue? Enter Y/N" key dup emit
  upc ascii Y =
;
```

See also: lcc

u.r

stack: (u size --)
 code: 9C

u is converted according to the value of *base* and then displayed as an unsigned number right-aligned in a field *size* digits wide. A trailing space is *not* displayed.

If the number of digits required to display *u* is greater than *size*, all the digits are displayed with no leading spaces in a field as wide as necessary.

For example:

```
: formatted-output ( -- )
  my-base      h# 8 u.r ." base" cr
  my-offset    h# 8 u.r ." offset" cr
;
```

user-abort

stack: (... --) (R: ... --)
code: 02 19

Used in an `alarm` routine to signify that the user has typed an abort sequence. When `alarm` finishes, instead of returning to the program that was interrupted by the execution of `alarm`, it enters the OpenBoot command interpreter by calling `abort`.

For example:

```
: test-dev-status ( -- error? ) ... ;
: my-checker ( -- ) test-dev-status if user-abort then ;
: install-abort ( -- ) ['] my-checker d# 10 alarm ;
```

value

stack: (E: -- x) (x "new-name< >" --)
code: (header) B8
generates: new-token|named-token|external-token b(value)

Creates and initializes a `value` with the name *new-name*. When later executed, *new-name* leaves its value on the stack. The value of *new-name* can be changed with `to`.

For example:

```
ok 123 value foo foo .
123
ok 456 to foo foo .
456
```

In FCode Source, `value` cannot appear inside a colon definition.

variable

stack: (E: -- a-addr) ("new-name< >" --)
code: (header) B9
generates: new-token | named-token | external-token b(variable)

Creates an uninitialized variable named *new-name*. When later executed, *new-name* leaves its address on the stack. The alignment of the returned address is system-dependent. The address holds a 32-bit value.

The value of *new-name* can be changed with ! and fetched with @ .

For example:

```
ok variable foo 123 foo ! foo @ .  
123  
ok 456 foo ! foo ?  
456
```

In FCode Source, value cannot appear inside a colon definition.

version1

stack: (--)
code: FD

version1 may only be used as the first byte of an FCode Program.

version1:

- Sets the `spread` value to 1 causing the FCode Evaluator to read successive bytes of the current FCode Program from successive addresses.
- Establishes the use of 8-bit branches.
- Reads an FCode header from the current FCode Program and either discards it or uses it to verify the integrity of the current FCode program in an implementation-dependent manner.

See also: `fcode-version2`, `start0`, `start1`, `start2`, `start4`

versionx?

stack: (-- flag)
code: ??

A group of tokenizer macros to determine the FCode version of the system running the FCode interpreter. They include:

Word	Generates
version1?	version b(lit) 2000.0000 <
version2?	version b(lit) 2000.0000 >= version b(lit) 3000.0000 <
version2.0?	version b(lit) 2000.0000 =
version2.1?	version b(lit) 2000.0001 =
version2.2?	version b(lit) 2000.0002 =
version2.3?	version b(lit) 2000.0003 =

Each returns `true` if the named version matches the system running the FCode interpreter.

w!

(n adr --)
code: 74

The low-order 16-bits of `n` are stored at location `adr` (through `adr+1`). The higher byte is stored at `adr`; the lower byte is stored at `adr+1`. `adr` must be on a 16-bit boundary; it must be evenly divisible by 2.

w,

(n --)
code: D1

Compile two bytes into the dictionary. The dictionary pointer must be two-byte-aligned.

See `c,` for limitations.

w@

```
( adr -- n )
code: 6F
```

Fetch the 16-bit number stored at *adr* (through *adr+1*). The higher byte is at *adr*; the lower byte is at *adr+1*. The remaining high bytes of *n* are set to zero. *adr* must be on a 16-bit boundary; it must be evenly divisible by 2.

/w

```
( -- n )
code: 5B
```

n is the size in bytes of a 16-bit word: 2.

/w*

```
stack: ( nu1 -- nu2 )
code: 67
```

nu2 is the result of multiplying *nu1* by */w*. This is the portable way to convert an index into a byte offset.

<w@

```
stack: ( waddr -- n )
code: 70
```

Fetches the 16-bit number stored at *waddr* and extends its sign into the upper bytes. *waddr* must be 16-bit-aligned.

For example:

```
ok 9123 8000 w! 8000 <w@ .h
ffff9123
ok 8000 w@ .h
9123
```

wa+

```
stack: ( addr1 index -- addr2 )
code: 5F
```

Increments *addr1* by *index* times the value of */w*. This is the portable way to increment an address.

wal+

stack: (addr1 -- addr2)
code: 63

Increments *addr1* by the value of */w*. This is the portable way to increment an address.

wbflip

stack: (w1 -- w2)
code: 80

w2 is the result of exchanging the two low-order bytes of the number *w1*. The two upper bytes of *w1* must be zero, or erroneous results will occur.

wbflips

stack: (waddr len --)
code: 02 36

Swaps the order of the bytes in each 16-bit word in the memory buffer *waddr len*.

waddr must be 16-bit-aligned. *len* must be a multiple */w*.

wbsplit

stack: (w -- b1.lo b2.hi)
code: AF

Splits the two lower bytes of *w* into two separate bytes (stored as the lower byte of each resulting item on the stack). The upper bytes of *w* must be zero.

while

stack: (C: dest-sys -- orig-sys dest-sys) (continue? --)
 code: 14 +offset
 generates: b?branch +offset

Tests the exit condition for a `begin...while...repeat` conditional loop. When the `while` is encountered, `continue?` is removed from the stack and tested. If `continue?` is *true*, execution continues from just after the `while` through to the `repeat` which then jumps back to just after the `begin`. If `continue?` is *false*, the loop is exited by causing execution to jump ahead to just after the `repeat`.

For example:

```
: probe-loop ( addr -- )
  \ generate a tight 'scope loop until a key is pressed.
  begin key? 0= while dup l@ drop repeat drop
;
```

window-left

stack: (-- border-width)
 code: 01 66

A value, containing the offset (in pixels) of the left edge of the active text area from the left edge of the visible display. The “active text area” is where characters are actually printed. (There is generally a border of unused blank area surrounding it on all sides.) `window-left` contains the size of the left portion of the unused border.

The size of the right portion of the unused border is determined by the difference between `screen-width` and the sum of `window-left` plus the width of the active text area (`#columns times char-width`).

This word is initially set to 0, but should always be set explicitly to an appropriate value if you wish to use *any* `fb1-` or `fb8-` utility routines. This can be done with `to`, or it can be set automatically by calling `fb1-install` or `fb8-install`.

When set with `fbx-install`, a calculation is done to set `window-left` so that the available unused border area is split between the left border and the right border. (The calculated value for `window-left` is rounded down to the nearest multiple of 32, though. This allows all pixel-drawing to proceed more efficiently.) If you wish to use `fbx-install` but desire a different value for `window-left`, simply change it with `to` *after* calling `fbx-install`.

window-top

stack: (-- border-height)
code: 01 65

A value, containing the offset (in pixels) of the top of the active text area from the top of the visible display. The “active text area” is where characters are actually printed. (There is generally a border of unused blank area surrounding it on all sides.) `window-top` contains the size of the top portion of the unused border.

The size of the bottom portion of the unused border is determined by the difference between `screen-height` and the sum of `window-top` plus the height of the active text area (`#lines times char-height`).

This word is initially set to 0, but should always be set explicitly to an appropriate value if you wish to use *any* `fb1-` or `fb8-` utility routines. This can be done with `to`, or it can be set automatically by calling `fb1-install` or `fb8-install`. When set with `fbx-install`, a calculation is done to set `window-top` so that the available unused border area is split between the top border and the bottom border. If you wish to use `fbx-install` but desire a different value for `window-top`, simply change it with `to` *after* calling `fbx-install`.

within

stack: (n min max -- min<=n<max?)
code: 45

min<=n<max? is *true* if *n* is between *min* and *max*, inclusive of *min* and exclusive of *max*.

See also: `between`.

wljoin

stack: (w.lo w.hi -- quad)
code: 7D

Merges two 16-bit numbers into a 32-bit number. The high bytes of *w.lo* and *w.hi* must be zero.

wpeek

stack: (waddr -- false | w true)
code: 02 21

Tries to read the 16-bit word at address *waddr*. Returns *w* and *true* if the access was successful. A *false* return indicates that a read access error occurred. *waddr* must be 16-bit aligned.

wpoke

stack: (w waddr -- okay?)
code: 02 24

Tries to write the 16-bit word at address *waddr*. Returns *true* if the access was successful. A *false* return indicates that a write access error occurred. *waddr* must be 16-bit aligned.

Note: *wpoke* may be unreliable on bus adapters that buffer write accesses.

wxjoin

stack: (w.lo w.2 w.3 w.hi -- o)
code: 02 44

Joins 4 doublets to form an octlet. Combines the sixteen least-significant bits of each operand to form an octlet. Ignores the high-order bits of each operand.

x,

stack: (o --)
code: 02 45

Compiles an octlet, *o*, into the dictionary (doubly-aligned).

x@

stack: (oaddr -- o)
code: 02 46

Fetches and octlet from an octlet-aligned address.

x!

stack: (o oaddr --)
code: 02 47

Stores an octlet to an octlet-aligned address.

/x

stack: (-- n)
code: 02 48

The number of address units in an octlet, typically eight.

xa+

stack: (addr1 index -- addr2)
code: 02 4a

.Increments addr1 by index times the value of /x.

xa1+

stack: (addr1 -- addr2)
code: 02 4b

Increments addr1 by the value of /x.

xbflip

stack: (oct1 -- oct2)
code: 02 4c

Reverses the bytes in an octlet..

xbflips

stack: (oaddr len --)
code: 02 4d

Reverses the bytes in each octlet in the given region. The region begins at oaddr and spans len bytes. The behavior is undefined if len is not a multiple of /x.

xbsplit

stack: (o -- b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi)
code: 02 4e

Splits an octlet into 8 bytes. The bits of greater significance than the eight least-significant bits of each of the eight resulting values are zero.

xlflip

stack: (oct1 -- oct2)
code: 02 4f

Reverses the quadlets in an octlet. Does not reverse the bytes in each quadlet.

xlflips

stack: (oaddr len --)
code: 02 50

Reverses the quadlets in each octlet in the given region. Does not reverse the bytes in each quadlet. The region begins at oaddr and spans len bytes.

xlsplit

stack: (o -- quad.lo quad.hi)
code: 02 51

Splits an octlet into 2 quadlets. The more-significant-bits than the 32 least-significant bits of each of the two resulting values are zero.

xor

stack: (x1 x2 -- x3)
code: 25

x3 is the bit-by-bit exclusive-or of x1 with x2.

xwflip

stack: (oct1 -- oct2)
code: 02 52

Reverses doublets in an octlet. Does not reverse bytes in each doublet.

xwflips

stack: (oaddr len --)
code: 02 53

Reverses doublets in each octlet in the given region. Does not reverse the bytes in each doublet. The region begins at oaddr and spans len bytes.

xwsplit

stack: (0 -- w.lo w.2 w.3 w.hi)
code: 02 54

Splits an octlet into four doublets. The more-significant-bits than the 16 least-significant bits of each of the four resulting values are zero..

FCode Reference



FCode Primitives

This appendix contains three lists:

- FCodes sorted according to functional group
- FCodes sorted by byte value
- FCodes sorted alphabetically by name

FCodes by Function

The following tables describe FCodes currently supported by OpenBoot. Both the FCode token values and Forth names are included. A token value entry of CR indicates a tokenizer-generated sequence, while - indicates that no FCode is generated.

Table A-1 Stack Manipulation

Value	Function	Stack	Description
51	depth	(??? -- ??? u)	Number of items on stack
46	drop	(x --)	Removes the top item from the stack
52	2drop	(x1 x2 --)	Removes 2 items from stack
CR	3drop	(x1 x2 x3 --)	Removes 3 items from stack
47	dup	(x -- x x)	Duplicates x
53	2dup	(x1 x2 -- x1 x2 x1 x2)	Duplicates 2 stack items

Table A-1 Stack Manipulation (Continued)

Value	Function	Stack	Description
CR	3dup	(x1 x2 x3 -- x1 x2 x3 x1 x2 x3)	Copies top 3 stack items
50	?dup	(x -- 0 x x)	Duplicates x if it is non-zero
4D	nip	(x1 x2 -- x2)	Discards the second stack item
48	over	(x1 x2 -- x1 x2 x1)	Copies second stack item to top of stack
54	2over	(x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2)	Copies 2 stack items
4E	pick	(xu ... x1 x0 u -- xu ... x1 x0 xu)	Copies u-th stack item
30	>r	(x --) (R: -- x)	Moves a stack item to the return stack
31	r>	(-- x) (R: x --)	Moves the top item from return stack to data stack
32	r@	(-- x) (R: x -- x)	Copies the top of the return stack to the data stack
4F	roll	(xu ... x1 x0 u -- xu-1 ... x1 x0 xu)	Rotates u stack items
4A	rot	(x1 x2 x3 -- x2 x3 x1)	Rotates 3 stack items (Same as 3roll)
4B	-rot	(x1 x2 x3 -- x3 x1 x2)	Rotates top 3 stack items in reverse order of rot
56	2rot	(x1 x2 x3 x4 x5 x6 -- x3 x4 x5 x6 x1 x2)	Rotates 3 pairs of stack items
49	swap	(x1 x2 -- x2 x1)	Exchanges the top 2 stack items
55	2swap	(x1 x2 x3 x4 -- x3 x4 x1 x2)	Exchanges 2 pairs of stack items
4C	tuck	(x1 x2 -- x2 x1 x2)	Copies the top stack item below the second item

Table A-2 Arithmetic Operations

Value	Function	Stack	Description
20	*	(nu1 nu2 -- prod)	Multiplies nu1 and nu2
1E	+	(nu1 nu2 -- sum)	Adds nu1 to nu2
1F	-	(nu1 nu2 -- diff)	Subtracts nu2 from nu1
21	/	(n1 n2 -- quot)	Divides n1 by n2
CR	1+	(nu1 -- nu2)	Adds 1
CR	1-	(nu1 -- nu2)	Subtracts 1
CR	2+	(nu1 -- nu2)	Adds 2
CR	2-	(nu1 -- nu2)	Subtracts 2
59	2*	(x1 -- x2)	Multiplies by 2
57	2/	(x1 -- x2)	Divides by 2
27	lshift	(x1 u -- x2)	Left shifts x1 by u places

Table A-2 Arithmetic Operations (Continued)

Value	Function	Stack	Description
28	rshift	(x1 u -- x2)	Right shifts <i>x1</i> by <i>u</i> places
CR	<<a	(n1 u -- n2)	Arithmetic left shifts (same as lshift)
29	>>a	(x1 u -- x2)	Arithmetic right shifts <i>x1</i> by <i>u</i> places
2D	abs	(n -- u)	Absolute value
AE	aligned	(n1 -- a-addr)	Adjusts an address to a machine word boundary
23	and	(x1 x2 -- x3)	Logical and
AC	bounds	(start len -- end start)	Converts <i>start,len</i> to <i>end,start</i> for do loop
2F	max	(n1 n2 -- n3)	<i>n3</i> is maximum of <i>n1</i> and <i>n2</i>
2E	min	(n1 n2 -- n3)	<i>n3</i> is minimum of <i>n1</i> and <i>n2</i>
22	mod	(n1 n2 -- rem)	Remainder of <i>n1/n2</i>
CR	*/mod	(n1 n2 n3 -- rem quot)	Remainder, quotient of <i>n1*n2/n3</i>
2A	/mod	(n1 n2 -- rem quot)	Remainder, quotient of <i>n1/n2</i>
2C	negate	(n1 -- n2)	Changes the sign of <i>n1</i>
26	invert	(x1 -- x2)	One's complement
26	not	(x1 -- x2)	Synonym for invert
24	or	(x1 x2 -- x3)	Logical or
2B	u/mod	(u1 u2 -- urem uquot)	Unsigned single precision divide of <i>u1/u2</i>
58	u2/	(x1 -- x2)	Logical right shifts 1 bit
25	xor	(x1 x2 -- x3)	Exclusive or
D4	um*	(u1 u2 -- ud.prod)	Multiplies two unsigned quadlets, yields an unsigned double precision product.
D5	um/mod	(ud u -- urem uquot)	Divides an unsigned double precision number by an unsigned single precision number, yields a single precision remainder and quotient
D8	d+	(d1 d2 -- d.sum)	Adds two double precision numbers
D9	d-	(d1 d2 -- d.diff)	Subtracts two double precision numbers

Table A-3 Memory Operations

Value	Function	Stack	Description
72	!	(x a-addr --)	Stores a number at <i>a-addr</i>
6C	+!	(n a-addr --)	Adds <i>n</i> to the number stored at <i>a-addr</i>
77	2!	(x1 x2 a-addr --)	Stores 2 numbers at <i>a-addr</i> ; <i>x2</i> at lower address
76	2@	(a-addr -- x1 x2)	Fetches 2 numbers from <i>a-addr</i> ; <i>x2</i> from lower address
6D	@	(a-addr -- x)	Fetches a number from at <i>a-addr</i>
CR	?	(a-addr --)	Displays the number at <i>a-addr</i>
75	c!	(byte addr --)	Stores <i>byte</i> at <i>addr</i>
71	c@	(addr -- byte)	Fetches <i>byte</i> from <i>addr</i>
CR	blank	(addr len --)	Sets <i>len</i> bytes of memory to ASCII space, starting at <i>addr</i>
7A	comp	(addr1 addr2 len -- n)	Compares two byte arrays including case. <i>n=0</i> if same
CR	erase	(addr len --)	Sets <i>len</i> bytes of memory to zero, starting at <i>addr</i>
79	fill	(addr len byte --)	Sets <i>len</i> bytes of memory to value <i>byte</i> starting at <i>addr</i>
0228	lbflips	(qaddr len --)	Reverses bytes within each quadlet in given region
0237	lwflips	(qaddr len --)	Exchanges doublets within quadlets in <i>qaddr len</i>
73	l!	(quad qaddr --)	Stores the quadlet at <i>qaddr</i> , must be 32-bit aligned
6E	l@	(qaddr -- quad)	Fetches the quadlet at <i>qaddr</i> , must be 32-bit aligned
78	move	(src-addr dest-addr len --)	Copies <i>len</i> bytes from <i>src-addr</i> to <i>dest-addr</i>
6B	off	(a-addr --)	Stores <i>false</i> at <i>a-addr</i>
6A	on	(a-addr --)	Stores <i>true</i> at <i>a-addr</i>
0236	wbflips	(waddr len --)	Exchanges bytes within doublets in the specified region
74	w!	(w waddr --)	Stores doublet <i>w</i> at <i>waddr</i> , must be 16-bit aligned
6F	w@	(waddr -- w)	Fetches the unsigned doublet at <i>waddr</i> , must be 16-bit aligned
70	<w@	(waddr -- n)	Fetches the signed doublet at <i>waddr</i> , must be 16-bit aligned

Table A-4 Atomic Access

Value	Function	Stack	Description
0230	rb@	(addr -- byte)	Reads the 8-bit value at the given address, atomically
0231	rb!	(byte addr --)	Writes the 8-bit value at the given address, atomically
0232	rw@	(waddr -- w)	Reads the doublet at the given address, atomically
0233	rw!	(w waddr --)	Writes the doublet at the given address, atomically
0234	rl@	(qaddr -- quad)	Reads the quadlet at the given address, atomically
0235	rl!	(quad qaddr --)	Writes the quadlet at the given address, atomically

Table A-5 Data Exception Tests

Value	Function	Stack	Description
0220	cpeek	(addr -- false byte true)	Reads 8-bit value at <i>addr</i> ; returns <i>false</i> if unsuccessful
0221	wpeek	(waddr -- false w true)	Reads doublet at <i>addr</i> ; returns <i>false</i> if unsuccessful
0222	lpeek	(qaddr -- false quad true)	Reads quadlet at <i>addr</i> ; returns <i>false</i> if unsuccessful
0223	cpoke	(byte addr -- okay?)	Writes 8-bit value at <i>addr</i> ; returns <i>false</i> if unsuccessful
0224	wpoke	(w waddr -- okay?)	Writes doublet to <i>addr</i> ; returns <i>false</i> if unsuccessful
0225	lpoke	(quad qaddr -- okay?)	Writes quadlet to <i>addr</i> ; returns <i>false</i> if unsuccessful

Table A-6 Comparison Operations

Value	Function	Stack	Description
36	0<	(n -- less-than-0?)	True if $n < 0$
37	0<=	(n -- less-or-equal-to-0?)	True if $n \leq 0$
35	0<>	(n -- not-equal-to-0?)	True if $n \neq 0$
34	0=	(n -- equal-to-0?)	True if $n = 0$, also inverts any flag
38	0>	(n -- greater-than-0?)	True if $n > 0$
39	0>=	(n -- greater-or-equal-to-0?)	True if $n \geq 0$
3A	<	(n1 n2 -- less?)	True if $n1 < n2$
43	<=	(n1 n2 -- less-or-equal?)	True if $n1 \leq n2$
3D	<>	(n1 n2 -- not-equal?)	True if $n1 \neq n2$

Table A-6 Comparison Operations (Continued)

Value	Function	Stack	Description
3C	=	(x1 x2 -- equal?)	True if $x1 = x2$
3B	>	(n1 n2 -- greater?)	True if $n1 > n2$
42	>=	(n1 n2 -- greater-or-equal?)	True if $n1 \geq n2$
44	between	(n min max -- min<=n<=max?)	True if $min \leq n \leq max$
CR	false	(-- false)	The value false (0)
CR	true	(-- true)	The value true (1)
40	u<	(u1 u2 -- unsigned-less?)	True if $u1 < u2$, unsigned
3F	u<=	(u1 u2 -- unsigned-less-or-equal?)	True if $u1 \leq u2$, unsigned
3E	u>	(u1 u2 -- unsigned-greater?)	True if $u1 > u2$, unsigned
41	u>=	(u1 u2 -- unsigned-greater-or-equal?)	True if $u1 \geq u2$, unsigned
45	within	(n min max -- min<=n<max?)	True if $min \leq n < max$

Table A-7 Text Input

Value	Function	Stack	Description
-	(([text<]> --)	Begins a comment (All text until next close parenthesis “)” is ignored)
-	\	(--)	Ignore rest of line (comment)
CR	ascii	([text< >] -- char)	ASCII value of next character
CR	control	([text< >] -- char)	Interprets next character as ASCII control character
8E	key	(-- char)	Reads a character from the keyboard
8D	key?	(-- pressed?)	True if a key has been typed on the keyboard
CR	accept	(addr len1 -- len2)	Gets an edited input line, stores it at <i>addr</i>
8A	expect	(addr len --)	Gets a line of edited input from the keyboard; stores it at <i>addr</i>
88	span	(-- a-addr)	Variable containing the number of characters read by <i>expect</i>

Table A-8 ASCII Constants

Value	Function	Stack	Description
AB	bell	(-- 0x07)	The ASCII code for the bell character; decimal 7
A9	bl	(-- 0x20)	The ASCII code for the space character; decimal 32
AA	bs	(-- 0x08)	The ASCII code for the backspace character; decimal 8
CR	carret	(-- 0x0D)	The ASCII code for the carriage return character; decimal 13
CR	linefeed	(-- 0x0A)	The ASCII code for the linefeed character; decimal 10
CR	newline	(-- 0x0A)	The ASCII code for the newline character; decimal 10

Table A-9 Numeric Input

Value	Function	Stack	Description
A4	-1	(-- -1)	Constant -1
A5	0	(-- 0)	Constant 0
A6	1	(-- 1)	Constant 1
A7	2	(-- 2)	Constant 2
A8	3	(-- 3)	Constant 3
CR	d#	([number< >] -- n)	Interprets next number in decimal
-	decimal	(--)	If outside definition, input numbers in decimal
CR	h#	([number< >] -- n)	Interprets next number in hexadecimal
-	hex	(--)	If outside definition, input numbers in hexadecimal

Table A-10 Numeric Primitives

Value	Function	Stack	Description
99	u#	(u1 -- u2)	Converts a digit in pictured numeric output
97	u#>	(u -- str len)	Ends pictured numeric output
96	<#	(--)	Initializes pictured numeric output
C7	#	(ud1 -- ud2)	Converts a digit in pictured numeric output conversion
C9	#>	(ud -- str len)	Ends pictured numeric output conversion
A0	base	(-- a-addr)	Variable containing number base

Table A-10 Numeric Primitives (Continued)

Value	Function	Stack	Description
A3	digit	(char base -- digit true char false)	Converts a character to a digit
95	hold	(char --)	Inserts the char in the pictured numeric output string
C8	#s	(ud -- 0 0)	Converts remaining digits in pictured numeric output
9A	u#s	(u1 -- u2)	Converts rest of the digits in pictured numeric output
98	sign	(n --)	Sets sign of pictured output
A2	\$number	(addr len -- true n false)	Converts a string to a number

Table A-11 Numeric Output

Value	Function	Stack	Description
9D	.	(nu --)	Displays a number in the current base
CR	.d	(n --)	Displays number in decimal
CR	decimal	(--)	If inside definition, numeric output in decimal
CR	.h	(n --)	Displays number in hexadecimal
CR	hex	(--)	If inside definition, numeric output in hexadecimal
9E	.r	(n size --)	Displays a number in a fixed width field
9F	.s	(... -- ...)	Displays the contents of the data stack
CR	s.	(n --)	Displays n as a signed number
9B	u.	(u --)	Displays an unsigned number
9C	u.r	(u size --)	Prints an unsigned number in a fixed width field

Table A-12 General-purpose Output

Value	Function	Stack	Description
CR	.(([text<>] --)	Displays a string now
91	(cr	(--)	Outputs ASCII CR character; decimal 13
92	cr	(--)	Starts a new line of display output
8F	emit	(char --)	Displays the character

Table A-12 General-purpose Output (Continued)

Value	Function	Stack	Description
CR	space	(--)	Outputs a single space character
CR	spaces	(n --)	Outputs <i>n</i> spaces
90	type	(text-addr text-len --)	Displays the text string

Table A-13 Formatted Output

Value	Function	Stack	Description
94	#line	(-- a-addr)	Variable holding the line number on the output device
93	#out	(-- a-addr)	Variable holding the column number on the output device

Table A-14 begin Loops

Value	Function	Stack	Description
CR	again	(C: dest-sys --)	Ends begin...again (infinite) loop
CR	begin	(C: -- dest-sys) (--)	Starts conditional or infinite loop
CR	repeat	(C: orig-sys dest-sys --) (--)	Returns to loop start (begin keyword)
CR	until	(C: dest-sys --) (done? --)	If not false, exits begin...until loop
CR	while	(C: dest-sys -- orig-sys dest-sys) (continue? --)	If not false, continues begin...while...repeat loop, else exits loop

Table A-15 Conditionals

Value	Function	Stack	Description
CR	if	(C: -- orig-sys) (do-next? --)	If not false, executes next FCode(s)
CR	else	(C: orig-sys1 -- orig-sys2) (--)	Executes next FCode(s) if if failed
CR	then	(C: orig-sys --) (--)	Terminates if...else...then construct

Table A-16 Case Statements

Value	Function	Stack	Description
CR	case	(C: -- case-sys) (sel -- sel)	Begins a case (multiple selection) statement
CR	endcase	(C: case-sys --) (sel <nothing> --)	Marks end of a case statement
CR	of	(C: case-sys1 -- case-sys2 of-sys) (sel of-val -- sel <nothing>)	Marks beginning of conditional execution clause based on case selector.
CR	endof	(C: case-sys1 of-sys -- case-sys2) (--)	Marks the end of an of clause

Table A-17 do Loops

Value	Function	Stack	Description
CR	do	(C: -- dodest-sys) (limit start --) (R: -- sys)	Marks beginning of loop which will execute with index value ranging from <i>start</i> to <i>limit-1</i> , inclusive
CR	?do	(C: -- dodest-sys) (limit start --) (R: -- sys)	Like do, but skips loop if <i>limit = start</i>
19	i	(-- index) (R: sys -- sys)	Returns current loop index value
1A	j	(-- index) (R: sys -- sys)	Returns value of next outer loop index
CR	leave	(--) (R: sys --)	Exits do loop immediately
CR	?leave	(exit? --) (R: sys --)	If flag is not false, exits do loop
CR	loop	(C: dodest-sys --) (--) (R: sys1 -- <nothing> sys2)	Increments index, returns to do
CR	+loop	(C: dodest-sys --) (n --) (R: sys1 -- <nothing> sys2)	Increments index by <i>n</i> , returns to do.
89	unloop	(--) (R: sys --)	Discards loop control parameters

Table A-18 Control Words

Value	Function	Stack	Description
1D	execute	(... xt -- ???)	Executes the word whose compilation address is on the stack
33	exit	(--) (R: sys --)	Returns from the current word

Table A-19 Strings

Value	Function	Stack	Description
CR	"	([text<">< >] -- text-str text-len)	Collects a string
CR	s"	([text<">] -- text-str text-len)	Gathers the immediately-following string
84	count	(pstr -- str len)	Unpacks a packed string
82	lcc	(char1 -- char2)	Converts <i>char1</i> to lower case
83	pack	(strlen addr -- pstr)	Makes a packed string from <i>addr strlen</i> , placing it at <i>pstr</i>
81	upc	(char1 -- char2)	Converts <i>char1</i> to upper case
0240	left-parse-string	(str len char -- R-str R-len L-str L-len)	Splits a string at the given delimiter (which is discarded)
011B	parse-2int	(str len -- val.lo val.hi)	Converts a string into a physical address and space

Table A-20 Defining Words

Value	Function	Stack	Description
CR	: (colon) name	(--)	Begins colon definition
CR	; (semicolon)	(--)	Ends colon definition
-	alias	(E: ... -- ???) ("new-name< >old-name< >" --)	Defines a <i>new-name</i> with behavior of <i>old-name</i>
CR	buffer:	(E: -- a-addr) (len "new-name< >" --)	Creates data array of <i>len</i> bytes
CR	constant	(E: -- x) (x "new-name< >" --)	Creates a constant
CR	create	(E: -- a-addr) ("new-name< >" --)	Generic defining word
CR	defer	(E: ... -- ???) ("new-name< >" --)	Execution vector (change with <i>to</i>)
CR	field	(E: addr -- addr+offset) (offset size "new-name< >" -- offset+size)	Creates a named offset pointer
C0	instance	(--)	Declare a data type to be local
CR	struct	(-- 0)	Initializes for <i>field</i> creation
CR	variable	(E: -- a-addr) ("new-name< >" --)	Creates a variable
CR	value	(E: -- x) (x "new-name< >" --)	Creates a value

Table A-21 Dictionary Compilation

Value	Function	Stack	Description
D3	,	(x --)	Places a number in the dictionary
D0	c,	(byte --)	Places a byte in the dictionary
AD	here	(-- addr)	Address of top of dictionary
D2	l,	(quad --)	Places a quadlet in the dictionary
D1	w,	(w --)	Places a doublet in the dictionary
CR	allot	(len --)	Allocates <i>len</i> bytes in the dictionary
CR	to	(param [old-name< >] --)	Changes value in a defer word or a value
DD	compile	(--)	Compiles following command at run time
DC	state	(-- a-addr)	Variable containing true if in compilation state

Table A-22 Dictionary Search

Value	Function	Stack	Description
CR	'	("old-name< >" -- xt)	Finds the named word (while executing)
CR	['] name	(-- xt)	Finds the named word (while compiling)
CB	\$find	(name-str name-len -- xt true name-str name-len false)	Finds the execution token corresponding to the name string in the dictionary
CD	eval	(... str len -- ???)	Executes Forth commands within a string
CD	evaluate	(... str len -- ???)	Interprets Forth text from the given string

Table A-23 Conversion Operators

Value	Function	Stack	Description
7F	bljoin	(bl.lo b2 b3 b4.hi -- quad)	Joins four bytes to form a quadlet
B0	bwjoin	(b.lo b.hi -- w)	Joins two bytes to form a doublet
5A	/c	(-- n)	Address increment for a byte; 1
-	/c*	(nu1 -- nu2)	Synonym for chars
66	chars	(nu1 -- nu2)	Multiplies by /c
5E	ca+	(addr1 index -- addr2)	Increments <i>addr1</i> by <i>index</i> times /c

Table A-23 Conversion Operators (Continued)

Value	Function	Stack	Description
CR	cal+	(addr1 -- addr2)	Synonym for char+
62	char+	(addr1 -- addr2)	Increments <i>addr1</i> by <i>/c</i>
80	wbflip	(w1 -- w2)	Swaps the bytes within a doublet
5C	/l	(-- n)	Address increment for a quadlet; 4
68	/l*	(nu1 -- nu2)	Multiplies by <i>/l</i>
60	la+	(addr1 index -- addr2)	Increments <i>addr1</i> by <i>index</i> times <i>/l</i>
64	lal+	(addr1 -- addr2)	Increments <i>addr1</i> by <i>/l</i>
0227	lbflip	(quad1 -- quad2)	Reverses the bytes within a quadlet
7E	lbsplit	(quad -- b.lo b2 b3 b4.hi)	Splits a quadlet into four bytes
7E	lwflip	(quad1 -- quad2)	Swaps the doublets within a quadlet
7C	lwsplit	(quad -- w1.lo w2.hi)	Splits a quadlet into two doublets
5D	/n	(-- n)	Address increment for a cell
CR	/n*	(nu1 -- nu2)	Synonym for cells
69	cells	(nu1 -- nu2)	Multiplies by <i>/n</i>
61	na+	(addr1 index -- addr2)	Increments <i>addr1</i> by <i>index</i> times <i>/n</i>
CR	nal+	(addr1 -- addr2)	Synonym for cell+
65	cell+	(addr1 -- addr2)	Increments <i>addr1</i> by <i>/n</i>
5B	/w	(-- n)	Address increment for a doublet; 2
67	/w*	(nu1 -- nu2)	Multiplies by <i>/w</i>
5F	wa+	(addr1 index -- addr2)	Increments <i>addr1</i> by <i>index</i> times <i>/w</i>
63	wal+	(addr1 -- addr2)	Increments <i>addr1</i> by <i>/w</i>
AF	wbsplit	(w -- b1.lo b2.hi)	Splits a doublet into two bytes
7D	wljoin	(w.lo w.hi -- quad)	Joins two doublets to form a quadlet

Table A-24 64-bit Operations

Value	Function	Stack	Description
022E	rx@	(oaddr -- o)	Reads the 64-bit value at the given address, atomically
022F	rx!	(o oaddr --)	Writes the 64-bit value at the given address, atomically
0241	bxjoin	(b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi -- o)	Joins 8 bytes to form a octlet
0242	<l@	(qaddr -- n)	Fetches a sign-extended quadlet at <i>qaddr</i>

Table A-24 64-bit Operations

Value	Function	Stack	Description
0243	lxjoin	(quad.lo quad.hi -- o)	Joins two quadlets to form an octlet
0244	wxjoin	(w.lo w.2 w.3 w.hi -- o)	Joins four doublets to form an octlet
0245	x,	(o --)	Places an octlet in the dictionary
0246	x@	(oaddr -- o)	Fetches the octlet at <i>oaddr</i> ; must be 64-bit aligned
0247	x!	(o oaddr --)	Stores an octlet at <i>oaddr</i> ; must be 64-bit aligned
0248	/x	(-- n)	Address increment for an octlet; 8
0249	/x*	(nu1 -- nu2)	Multiplies by /x
024A	xa+	(addr1 index -- addr2)	Increments <i>addr1</i> by <i>index</i> times /x
024B	xa1+	(addr1 -- addr2)	Increments <i>addr1</i> by /x
024C	xbflip	(oct1 -- oct2)	Reverse bytes within octlet
024D	xbflips	(oaddr len --)	Reverse bytes within each octlet in given region
024E	xbsplit	(o -- b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi)	Splits an octlet into 8 bytes
024F	xlflip	(oct1 -- oct2)	Reverse quadlets within octlet
0250	xlflips	(oaddr len --)	Reverse quadlets within each octlet in given region
0251	xlsplit	(o -- quad.lo quad.hi)	Splits an octlet into 2 quadlets
0252	xwflip	(oct1 -- oct2)	Reverse doublets within octlet
0253	xwflips	(oaddr len --)	Reverse doublets within each octlet in given region
0254	xwsplit	(o -- w.lo w.2 w.3 w.hi)	Splits an octlet into 4 doublets

Table A-25 Memory Buffers Allocation

Value	Function	Stack	Description
8B	alloc-mem	(n -- a-addr)	Allocates <i>n</i> bytes of memory and returns its address
8C	free-mem	(n len --)	Frees memory allocated by alloc-mem

Table A-26 Miscellaneous Operators

Value	Function	Stack	Description
86	>body	(xt -- a-addr)	Finds parameter field address from compilation address
85	body>	(a-addr -- xt)	Finds compilation address from parameter field address
DA	get-token	(FCode# -- xt immediate?)	Converts FCode Number to function execution token
DB	set-token	(xt immediate? FCode# --)	Assigns FCode Number to existing function

Table A-26 Miscellaneous Operators (Continued)

Value	Function	Stack	Description
00	end0	(--)	Marks the end of FCode
FF	end1	(--)	Alternates form for end0 (not recommended)
CR	fcode-version1	(--)	Begins FCode program
023E	byte-load	(addr xt --)	Interprets FCode beginning at location <i>addr</i>
-	fload	([filename<cr>] --)	Begins tokenizing <i>filename</i>
-	headerless	(--)	Creates new names with <i>new-token</i> (no name fields)
-	headers	(--)	Creates new names with <i>named-token</i> (default)
7B	noop	(--)	Does nothing
CC	offset16	(--)	All further branches use 16-bit offsets (instead of 8-bit)
-	tokenizer[(--)	Begins tokenizer program commands
-]tokenizer	(--)	Ends tokenizer program commands
CR	fcode-version2	(--)	Begins 2.0 FCode program, compiles <i>start1</i>
-	external	(--)	Creates new names with <i>external-token</i>
CR	fcode-version3	(--)	Begins 3.0 FCode program, compiles <i>start1</i>

Table A-27 Internal Operators (invalid for program text)

Value	Function	Stack	Description
01-0F			First byte of a two byte FCode
10	b(lit)	(-- n) (F: /FCode-num32/ --)	Followed by 32-bit#. Compiled by numeric data
11	b(')	(-- xt) (F: /FCode#/ --)	Followed by a token (1 or 2-byte code) . Compiled by ['] or ' .
12	b(")	(-- str len) (F: /FCode-string/ --)	Followed by count byte, text. Compiled by " or . "
C3	b(to)	(x --)	Compiled by to
FD	version1	(--)	Followed by reserved byte, checksum (2 bytes) , length (4 bytes). Compiled by <i>fcode-version1</i> , as the first FCode bytes
13	bbranch	(--) (F: /FCode-offset/ --)	Followed by offset. Compiled by <i>else</i> or <i>again</i>
14	b?branch	(don't-branch? --) (F: /FCode-offset/ --)	Followed by offset. Compiled by <i>if</i> or <i>until</i>

Table A-27 Internal Operators (invalid for program text) (Continued)

Value	Function	Stack	Description
15	b(loop)	(--) (F: /FCcode-offset/ --)	Followed by offset. Compiled by loop
16	b(+loop)	(delta --) (F: /FCcode-offset/ --)	Followed by offset. Compiled by +loop
17	b(do)	(limit start --) (F: /FCcode-offset/ --)	Followed by offset. Compiled by do
18	b(?do)	(limit start --) (F: /FCcode-offset/ --)	Followed by offset. Compiled by ?do
1B	b(leave)	(F: --)	Compiled by leave or ?leave
B1	b(<mark)	(F: --)	Compiled by begin
B2	b(>resolve)	(--) (F: --)	Compiled by else or then
C4	b(case)	(sel -- sel) (F: --)	Compiled by case
C5	b(endcase)	(sel <nothing> --) (F: --)	Compiled by endcase
C6	b(endof)	(--) (F: /FCcode-offset/ --)	Compiled by endof
1C	b(of)	(sel of-val -- sel <nothing>) (F: /FCcode-offset/ --)	Followed by offset. Compiled by of
B5	new-token	(--) (F: /FCcode#/ --)	Followed by table#, code#, token-type. Compiled by any defining word.
B6	named-token	(--) (F: /FCcode-string FCode#/ --)	Followed by packed string (count,text), table#, code#, token-type. Compiled by any defining word (: value constant etc.)
B7	b(:)	(E: ... -- ???) (F: -- colon-sys)	Token-type compiled by :
B8	b(value)	(E: -- x) (F: x --)	Token-type compiled by value
B9	b(variable)	(E: -- a-addr) (F: --)	Token-type compiled by variable
BA	b(constant)	(E: -- n) (F: n --)	Token-type compiled by constant
BB	b(create)	(E: -- a-addr) (F: --)	Token-type compiled by create
BC	b(defer)	(E: ... -- ???) (F: --)	Token-type compiled by defer
BD	b(buffer:)	(E: -- a-addr) (F: size --)	Token-type compiled by buffer:
BE	b(field)	(E: addr -- addr+offset) (F: offset size -- offset+size)	Token-type compiled by field
C2	b(;	(--) (F: colon-sys --)	End a colon definition. Compiled by ;
CA	external-token	(--) (F: /FCcode-string FCode#/ --)	Create a new named FCode function.

Table A-27 Internal Operators (invalid for program text) (Continued)

Value	Function	Stack	Description
F0	start0	(--)	Like <code>version1</code> , but for version 2.x and 3.x FCodes. Uses 16-bit branches. Fetches successive tokens from same address
F1	start1	(--)	Like <code>version1</code> , but for version 2.x and 3.x FCodes. Uses 16-bit branches. Fetches successive tokens from consecutive addresses. Compiled by <code>fcode-version2</code>
F2	start2	(--)	Like <code>version1</code> , but for version 2.x and 3.x FCodes. Uses 16-bit branches. Fetches successive tokens from consecutive 16-bit addresses
F3	start4	(--)	Like <code>version1</code> , but for version 2.x and 3.x FCodes. Uses 16-bit branches. Fetches successive tokens from consecutive 32-bit addresses

Table A-28 Virtual Memory Allocation

Value	Function	Stack	Description
0105	free-virtual	(virt size --)	Frees virtual memory obtained using <code>map-low</code> .
0130	map-low	(phys-low size -- virt)	Allocate virtual memory.

Table A-29 Properties

Value	Function	Stack	Description
01 10	property	(prop-addr prop-len name-str name-len --)	Declares a property with the given value structure, for the given name string.
02 1E	delete-property	(name-str name-len --)	Deletes the property with the given name
01 15	encode-bytes	(data-addr data-len -- prop-addr prop-len)	Converts a byte array into an prop-format string
01 11	encode-int	(n -- prop-addr prop-len)	Converts a number into an prop-format string
01 13	encode-phys	(phys.lo ... phys.hi -- prop-addr prop-len)	Converts physical address and space into an prop-format string

Table A-29 Properties (Continued)

Value	Function	Stack	Description
01 14	encode-string	(str len -- prop-addr prop-len)	Converts a string into an prop-format string
01 12	encode+	(prop-addr1 prop-len1 prop-addr2 prop-len2 -- prop-addr3 prop-len3)	Merges two prop-format strings. They must have been created sequentially
CR	decode-bytes	(prop-addr1 prop-len1 data-len -- prop-addr2 prop-len2 data-addr data-len)	Decodes a byte array from a prop-encoded-array
02 1B	decode-int	(prop-addr1 prop-len1 -- prop-addr2 prop-len2 n)	Converts the beginning of an prop-format string to an integer
02 1C	decode-string	(prop-addr1 prop-len1 -- prop-addr2 prop-len2 str len)	Converts the beginning of a prop-format string to a normal string
01 28	decode-phys	(prop-addr1 prop-len1 -- prop-addr2 prop-len2 phys.lo ... phys.hi)	Decode a unit-address from a prop-encoded array
02 1A	get-my-property	(name-str name-len -- true prop-addr prop-len false)	Returns the prop-format string for the given property name
02 1D	get-inherited-property	(name-str name-len -- true prop-addr prop-len false)	Returns the value string for the given property, searches parents' properties if not found
02 1F	get-package-property	(name-str name-len phandle -- true prop-addr prop-len false)	Returns the prop-format string for the given property name in the package phandle

Table A-30 Commonly-used Properties

Value	Function	Stack	Description
0116	reg	(phys.lo ... phys.hi size --)	Declares location and size of device registers
0119	model	(str len --)	Declares model# for this device, such as " SUNW,501-1623-01"
011A	device-type	(str len --)	Declares type of device, e.g. " display", " block", " network", or " byte"
CR	name	(addr len --)	Declares driver name, as in " SUNW,zebra"
0201	device-name	(str len --)	Creates the "name" property with the given value

Table A-31 System Version Information

Value	Function	Stack	Description
87	fcode-revision	(-- n)	Returns major/minor FCode interface version

Table A-32 Device Activation Vector Setup

Value	Function	Stack	Description
01 1C	is-install	(xt --)	Identifies "install" routine to allocate a frame buffer
01 1D	is-remove	(xt --)	Identifies "remove" routine, to deallocate a frame buffer
01 1E	is-selftest	(xt --)	Identifies "selftest" routine for this frame buffer
01 1F	new-device	(--)	Creates a new device node
01 27	finish-device	(--)	Completes current device

Table A-33 Self-test Utility Routines

Value	Function	Stack	Description
01 20	diagnostic-mode?	(-- diag?)	Returns <code>true</code> if extended diagnostics are desired
01 21	display-status	(n --)	Obsolete
01 22	memory-test-suite	(addr len -- fail?)	Calls memory tester for given region
01 24	mask	(-- a-addr)	Variable, holds "mask" used by memory-test-suite

Table A-34 Time Utilities

Value	Function	Stack	Description
01 25	get-msecs	(-- n)	Returns the current number of milliseconds
01 26	ms	(n --)	Delays for n milliseconds. Resolution is 1 millisecond
02 13	alarm	(xt n --)	Periodically execute <code>xt</code> . If <code>n=0</code> , stop.

Table A-35 Machine-specific Support

Value	Function	Stack	Description
01 30	map-low	(phys.lo ... size -- virt)	Maps a region of memory in device's physical address space
01 31	sbus-intr>cpu	(sbus-intr# -- cpu-intr#)	Translates SBus interrupt# into CPU interrupt#

Note – Table A-36 through Table A-42 apply only to *display* device-types.

Table A-36 User-set Terminal Emulation Values

Value	Function	Stack	Description
01 50	#lines	(-- rows)	Number of lines of text being used for display. This word must be initialized (using to). fbx-install does this automatically
01 51	#columns	(-- columns)	Number of columns (chars/line) used for display. This word must be initialized (using to). fbx-install does this automatically

Table A-37 Terminal-set Terminal Emulation Values

Value	Function	Stack	Description
01 52	line#	(-- line#)	Current cursor position (row). 0 is top line
01 53	column#	(-- column#)	Current cursor position (column). 0 is left char.
01 54	inverse?	(-- white-on-black?)	True if output is inverted (white-on-black)
01 55	inverse-screen?	(-- black?)	True if screen has been inverted (black background)

Table A-38 Terminal Emulation Routines*

Value	Function	Stack	Description
01 57	draw-character	(char --)	Paints the given character and advances the cursor
01 58	reset-screen	(--)	Initializes the display device
01 59	toggle-cursor	(--)	Draws or erases the cursor
01 5A	erase-screen	(--)	Clears all pixels on the display
01 5B	blink-screen	(--)	Flashes the display momentarily

Table A-38 Terminal Emulation Routines* (Continued)

Value	Function	Stack	Description
01 5C	invert-screen	(--)	Changes all pixels to the opposite color
01 5D	insert-characters	(n --)	Inserts n blanks just before the cursor
01 5E	delete-characters	(n --)	Deletes n characters to the right of the cursor Remaining chars slide left
01 5F	insert-lines	(n --)	Inserts n blank lines just before the current line, lower lines are scrolled downward
01 60	delete-lines	(n --)	Deletes n lines starting with the current line, lower lines are scrolled upward
01 61	draw-logo	(line# addr width height --)	Draws the logo

*defer-type loadable routines.

Table A-39 Frame Buffer Parameter Values*

Value	Function	Stack	Description
016C	char-height	(-- height)	Height (in pixels) of a character (usually 22)
016D	char-width	(-- width)	Width (in pixels) of a character (usually 12)
016F	fontbytes	(-- bytes)	Number of bytes/scan line for font entries (usually 2)
0162	frame-buffer- adr	(-- addr)	Address of frame buffer memory
0163	screen-height	(-- height)	Total height of the display (in pixels)
0164	screen-width	(-- width)	Total width of the display (in pixels)
0165	window-top	(-- border-height)	Distance (in pixels) between display top and text window
0166	window-left	(-- border-width)	Distance (in pixels) between display left edge and text window left edge

*These must all be initialized before using any fbx- routines.

Table A-40 Font Operators

Value	Function	Stack	Description
016A	default-font	(-- addr width height advance min-char #glyphs)	Returns default font values, plugs directly into set-font
016B	set-font	(addr width height advance min-char #glyphs --)	Sets the character font for text output
016E	>font	(char -- addr)	Returns font address for given ASCII character

Table A-41 One-bit Frame Buffer Utilities

Value	Function	Stack	Description
0170	fb1-draw-character	(char --)	Paints the character and advance the cursor
0171	fb1-reset-screen	(--)	Initializes the display device (noop)
0172	fb1-toggle-cursor	(--)	Draws or erases the cursor
0173	fb1-erase-screen	(--)	Clears all pixels on the display
0174	fb1-blink-screen	(--)	Inverts the screen, twice (slow)
0175	fb1-invert-screen	(--)	Changes all pixels to the opposite color
0176	fb1-insert-characters	(n --)	Inserts n blanks just before the cursor
0177	fb1-delete-characters	(n --)	Deletes n characters, starting at with cursor character, rightward. Remaining chars slide left
0178	fb1-insert-lines	(n --)	Inserts n blank lines just before the current line, lower lines are scrolled downward
0179	fb1-delete-lines	(n --)	Deletes n lines starting with the current line, lower lines are scrolled upward
017A	fb1-draw-logo	(line# addr width height --)	Draws the logo
017B	fb1-install	(width height #columns #lines --)	Installs the one-bit built-in routines
017C	fb1-slide-up	(n --)	Like fb1-delete-lines, but doesn't clear lines at bottom

Table A-42 eight-bit Frame Buffer Utilities

Value	Function	Stack	Description
0180	fb8-draw-character	(char --)	Paints the character and advance the cursor
0181	fb8-reset-screen	(--)	Initializes the display device (noop)
0182	fb8-toggle-cursor	(--)	Draws or erases the cursor
0183	fb8-erase-screen	(--)	Clears all pixels on the display
0184	fb8-blink-screen	(--)	Inverts the screen, twice (slow)
0185	fb8-invert-screen	(--)	Changes all pixels to the opposite color
0186	fb8-insert-characters	(n --)	Inserts n blanks just before the cursor
0187	fb8-delete-characters	(n --)	Deletes n characters starting with cursor char, rightward. Remaining chars slide left
0188	fb8-insert-lines	(n --)	Inserts n blank lines just before the current line, lower lines are scrolled downward
0189	fb8-delete-lines	(n --)	Deletes n lines starting with the current line, lower lines are scrolled upward
018A	fb8-draw-logo	(line# addr width height --)	Draws the logo
018B	fb8-install	(width height #columns #lines --)	Installs the eight-bit built-in routines

Table A-43 Package Support

Value	Function	Stack	Description
0129	push-package	(phandle --)	Make <i>phandle</i> the active package
012A	pop-package	(--)	Undo the effect of the most recent <i>push-package</i> relative to each execution of <i>pop-package</i>
012B	interpose	(adr len phandle --)	Can add more capabilities on top of unmodified OpenBoot device drivers
023C	peer	(phandle -- phandle.sibling)	Returns <i>phandle</i> of package that is the next child of the the parent of the package
023B	child	(phandle.parent -- phandle.child)	Returns <i>phandle</i> of the package that is the first child of the package parent <i>phandle</i>

Table A-43 Package Support (Continued)

Value	Function	Stack	Description
0204	find-package	(name-str name-len -- false phandle true)	Finds a package named <i>name-str</i>
0205	open-package	(arg-str arg-len phandle -- ihandle 0)	Opens an instance of the package <i>phandle</i> , passes arguments <i>arg-str arg-len</i>
020F	\$open-package	(arg-str arg-len name-str name-len -- ihandle 0)	Finds a package <i>name-str name-len</i> then opens it with arguments <i>arg-str arg-len</i>
020A	my-parent	(-- ihandle)	Returns the <i>ihandle</i> of the parent of the current package instance
0203	my-self	(-- ihandle)	Returns the <i>ihandle</i> of currently-executing package instance
020B	ihandle>phandle	(ihandle -- phandle)	Converts an <i>ihandle</i> to a <i>phandle</i>
0206	close-package	(ihandle --)	Closes an instance of a package
0207	find-method	(method-str method-len phandle -- false xt true)	Finds the method (command) named <i>method-str</i> in the package <i>phandle</i>
0208	call-package	(... xt ihandle -- ???)	Executes the method <i>xt</i> in the instance <i>ihandle</i>
020E	\$call-method	(... method-str method-len ihandle -- ???)	Executes the method named <i>method-str</i> in the instance <i>ihandle</i>
0209	\$call-parent	(... method-str method-len -- ???)	Executes the method <i>method-str</i> in the parent's package
0202	my-args	(-- arg-str arg-len)	Returns the argument string passed when this package was opened
020D	my-unit	(-- phys.lo ... phys.hi)	Returns the physical unit number pair for this package
0102	my-address	(-- phys.lo ...)	Returns the physical addr of this plug-in device. <i>Phys</i> is a "magic" number, usable by other routines
0103	my-space	(-- phys.hi)	Returns address space of plug-in device. <i>Space</i> is a "magic" number, usable by other routines

Table A-44 Asynchronous Support

Value	Function	Stack	Description
0213	alarm	(xt n --)	Executes method (command) indicated by <i>xt</i> every <i>n</i> milliseconds
0219	user-abort	(... --) (R: ... --)	Abort after alarm routine finishes execution

Table A-45 Miscellaneous Operations

Value	Function	Stack	Description
0214	(is-user-word)	(E: ... -- ???) (name-str name-len xt --)	Creates a new word called <i>name-str</i> which executes <i>xt</i>
01A4	mac-address	(-- mac-str mac-len)	Returns the MAC address

Table A-46 Interpretation

Value	Function	Stack	Description
0215	suspend-fcode	(--)	Suspends execution of FCode, resumes later if an undefined command is required

Table A-47 Error Handling

Value	Function	Stack	Description
0216	abort	(... --) (R:... --)	Aborts FCode execution, returns to the "ok" prompt
0217	catch	(... xt -- ??? error-code ??? false)	Executes <i>xt</i> , returns throw error code or 0 if throw not encountered
0218	throw	(... error-code -- ??? error-code ...)	Returns given error code to <i>catch</i>
FC	ferror	(--)	Displays "Unimplemented FCode" and stops FCode interpretation

FCodes by Byte Value

The following table lists, in hexadecimal order, currently-assigned FCode byte values.

Table A-48 FCodes by Byte Value

Value	Function	Stack
00	end0	(--)
10	b(lit)	(-- n) (F: /FCode-num32/ --)
11	b(')	(-- xt) (F: /FCode#/ --)
12	b(")	(-- str len) (F: /FCode-string/ --)
13	bbranch	(--) (F: /FCode-offset/ --)
14	b?branch	(don't-branch? --) (F: /FCode-offset/ --)
15	b(loop)	(--) (F: /FCode-offset/ --)
16	b(+loop)	(delta --) (F: /FCode-offset/ --)
17	b(do)	(limit start --) (F: /FCode-offset/ --)
18	b(?do)	(limit start --) (F: /FCode-offset/ --)
19	i	(-- index) (R: sys -- sys)
1A	j	(-- index) (R: sys -- sys)
1B	b(leave)	(F: --)
1C	b(of)	(sel of-val -- sel <nothing>) (F: /FCode-offset/ --)
1D	execute	(... xt -- ???)
1E	+	(nu1 nu2 -- sum)
1F	-	(nu1 nu2 -- diff)
20	*	(nu1 nu2 -- prod)
21	/	(n1 n2 -- quot)
22	mod	(n1 n2 -- rem)
23	and	(x1 x2 -- x3)
24	or	(x1 x2 -- x3)
25	xor	(x1 x2 -- x3)
26	invert	(x1 -- x2)
27	lshift	(x1 u -- x2)
28	rshift	(x1 u -- x2)
29	>>a	(x1 u -- x2)

Table A-48 FCodes by Byte Value (Continued)

Value	Function	Stack
2A	/mod	(n1 n2 -- rem quot)
2B	u/mod	(u1 u2 -- urem uquot)
2C	negate	(n1 -- n2)
2D	abs	(n -- u)
2E	min	(n1 n2 -- n1 n2)
2F	max	(n1 n2 -- n1 n2)
30	>r	(x --) (R: -- x)
31	r>	(-- x) (R: x --)
32	r@	(-- x) (R: x -- x)
33	exit	(--) (R: sys --)
34	0=	(nulflag -- equal-to-0?)
35	0<>	(n -- not-equal-to-0?)
36	0<	(n -- less-than-0?)
37	0<=	(n -- less-or-equal-to-0?)
38	0>	(n -- greater-than-0?)
39	0>=	(n -- greater-or-equal-to-0?)
3A	<	(n1 n2 -- less?)
3B	>	(n1 n2 -- greater?)
3C	=	(x1 x2 -- equal?)
3D	<>	(x1 x2 -- not-equal?)
3E	u>	(u1 u2 -- unsigned-greater?)
3F	u<=	(u1 u2 -- unsigned-less-or-equal?)
40	u<	(u1 u2 -- unsigned-less?)
41	u>=	(u1 u2 -- unsigned-greater-or-equal?)
42	>=	(n1 n2 -- greater-or-equal?)
43	<=	(n1 n2 -- less-or-equal?)
44	between	(n min max -- min<=n<=max?)
45	within	(n min max -- min<=n<max?)
46	drop	(x --)
47	dup	(x -- x x)
48	over	(x1 x2 -- x1 x2 x1)

Table A-48 FCodes by Byte Value (Continued)

Value	Function	Stack
49	swap	(x1 x2 -- x2 x1)
4A	rot	(x1 x2 x3 -- x2 x3 x1)
4B	-rot	(x1 x2 x3 -- x3 x1 x2)
4C	tuck	(x1 x2 -- x2 x1 x2)
4D	nip	(x1 x2 -- x2)
4E	pick	(xu ... x1 x0 u -- xu ... x1 x0 xu)
4F	roll	(xu ... x1 x0 u -- xu-1 ... x1 x0 xu)
50	?dup	(x -- 0 x x)
51	depth	(-- u)
52	2drop	(x1 x2 --)
53	2dup	(x1 x2 -- x1 x2 x1 x2)
54	2over	(x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2)
55	2swap	(x1 x2 x3 x4 -- x3 x4 x1 x2)
56	2rot	(x1 x2 x3 x4 x5 x6 -- x3 x4 x5 x6 x1 x2)
57	2/	(x1 -- x2)
58	u2/	(x1 -- x2)
59	2*	(x1 -- x2)
5A	/c	(-- n)
5B	/w	(-- n)
5C	/l	(-- n)
5D	/n	(-- n)
5E	ca+	(addr1 index -- addr2)
5F	wa+	(addr1 index -- addr2)
60	la+	(addr1 index -- addr2)
61	na+	(addr1 index -- addr2)
62	char+	(addr1 -- addr2)
63	wal+	(addr1 -- addr2)
64	lal+	(addr1 -- addr2)
65	cell+	(addr1 -- addr2)
66	chars	(nu1 -- nu2)
67	/w*	(nu1 -- nu2)

Table A-48 FCodes by Byte Value (Continued)

Value	Function	Stack
68	/l*	(nu1 -- nu2)
69	cells	(nu1 -- nu2)
6A	on	(a-addr --)
6B	off	(a-addr --)
6C	+	(nu a-addr --)
6D	@	(a-addr -- x)
6E	l@	(quad -- quad)
6F	w@	(waddr -- w)
70	<w@	(waddr -- n)
71	c@	(addr -- byte)
72	!	(x a-addr --)
73	l!	(quad quad --)
74	w!	(w waddr --)
75	c!	(byte addr --)
76	2@	(a-addr -- x1 x2)
77	2!	(x1 x2 a-addr --)
78	move	(src-addr dest-addr len --)
79	fill	(addr len byte --)
7A	comp	(addr1 addr2 len -- n)
7B	noop	(--)
7C	lwsplit	(quad -- w1.lo w2.hi)
7D	wljoin	(w.lo w.hi -- quad)
7E	lbsplit	(quad -- b.lo b2 b3 b4.hi)
7F	bljoin	(bl.lo b2 b3 b4.hi -- quad)
80	wbflip	(w1 -- w2)
81	upc	(char1 -- char2)
82	lcc	(char1 -- char2)
83	pack	(str len addr -- pstr)
84	count	(pstr -- str len)
85	body>	(a-addr -- xt)
86	>body	(xt -- a-addr)

Table A-48 FCodes by Byte Value (Continued)

Value	Function	Stack
87	fcode-revision	(-- n)
88	span	(-- a-addr)
89	unloop	(--) (R: sys --)
8A	expect	(addr len --)
8B	alloc-mem	(len -- a-addr)
8C	free-mem	(a-addr len --)
8D	key?	(-- pressed?)
8E	key	(-- char)
8F	emit	(char --)
90	type	(text-str text-len --)
91	(cr	(--)
92	cr	(--)
93	#out	(-- a-addr)
94	#line	(-- a-addr)
95	hold	(char --)
96	<#	(--)
97	u#>	(u -- str len)
98	sign	(n --)
99	u#	(u1 -- u2)
9A	u#s	(u1 -- u2)
9B	u.	(u --)
9C	u.r	(u size --)
9D	.	(nu --)
9E	.r	(n size --)
9F	.s	(... -- ...)
A0	base	(-- a-addr)
A2	\$number	(addr len -- true n false)
A3	digit	(char base -- digit true char false)
A4	-1	(-- -1)
A5	0	(-- 0)
A6	1	(-- 1)

Table A-48 FCodes by Byte Value (Continued)

Value	Function	Stack
A7	2	(-- 2)
A8	3	(-- 3)
A9	b1	(-- 0x20)
AA	bs	(-- 0x08)
AB	bell	(-- 0x07)
AC	bounds	(n cnt -- n+cnt n)
AD	here	(-- addr)
AE	aligned	(n1 -- n1 a-addr)
AF	wbsplit	(w -- b1.lo b2.hi)
B0	bwjoin	(b.lo b.hi -- w)
B1	b(<mark)	(F: --)
B2	b(>resolve)	(--) (F: --)
B5	new-token	(--) (F: /FCCode#/ --)
B6	named-token	(--) (F: /FCCode-string FCCode#/ --)
B7	b(:)	(E: ... -- ???) (F: -- colon-sys)
B8	b(value)	(E: -- x) (F: x --)
B9	b(variable)	(E: -- a-addr) (F: --)
BA	b(constant)	(E: -- n) (F: n --)
BB	b(create)	(E: -- a-addr) (F: --)
BC	b(defer)	(E: ... -- ???) (F: --)
BD	b(buffer:)	(E: -- a-addr) (F: size --)
BE	b(field)	(E: addr -- addr+offset) (F: offset size -- offset+size)
C0	instance	(--)
C2	b(;)	(--) (F: colon-sys --)
C3	b(to)	(x --)
C4	b(case)	(sel -- sel) (F: --)
C5	b(endcase)	(sel <nothing> --) (F: --)
C6	b(endof)	(--) (F: /FCCode-offset/ --)
C7	#	(ud1 -- ud2)
C8	#s	(ud -- 0 0)
C9	#>	(ud -- str len)

Table A-48 FCodes by Byte Value (Continued)

Value	Function	Stack
CA	external-token	(--) (F: /FCode-string FCode#/ --)
CB	\$find	(name-str name-len -- xt true name-str name-len false)
CC	offset16	(--)
CD	evaluate	(... str len -- ???)
D0	c,	(byte --)
D1	w,	(w --)
D2	l,	(quad --)
D3	,	(x --)
D4	um*	(u1 u2 -- ud.prod)
D5	um/mod	(ud u -- urem uquot)
D8	d+	(d1 d2 --d.sum)
D9	d-	(d1 d2 -- d.diff)
DA	get-token	(fcode# -- xt immediate?)
DB	set-token	(xt immediate? fcode# --)
DC	state	(-- a-addr)
DD	compile,	(xt --)
DE	behavior	(defer-xt -- contents-xt)
F0	start0	(--)
F1	start1	(--)
F2	start2	(--)
F3	start4	(--)
FC	ferror	(--)
FD	version1	(--)
FF	endl	(--)
0102	my-address	(-- phys.lo ...)
0103	my-space	(-- phys.hi)
0105	free-virtual	(virt size --)
0110	property	(prop-addr prop-len name-str name-len --)
0111	encode-int	(n -- prop-addr prop-len)
0112	encode+	(prop-addr1 prop-len1 prop-addr2 prop-len2 -- prop-addr3 prop-len3)
0113	encode-phys	(phys.lo ... phys.hi -- prop-addr prop-len)

Table A-48 FCodes by Byte Value (Continued)

Value	Function	Stack
0114	encode-string	(str len -- prop-addr prop-len)
0115	encode-bytes	(data-addr data-len -- prop-addr prop-len)
0116	reg	(phys.lo ... phys.hi size --)
0119	model	(str len --)
011A	device-type	(str len --)
011B	parse-2int	(str len -- val.lo val.hi)
011C	is-install	(xt --)
011D	is-remove	(xt --)
011E	is-selftest	(xt --)
011F	new-device	(--)
0120	diagnostic-mode?	(-- diag?)
0121	display-status	(n --)
0122	memory-test-suite	(addr len -- fail?)
0124	mask	(-- a-addr)
0125	get-msecs	(-- n)
0126	ms	(n --)
0127	finish-device	(--)
0128	decode-phys	(prop-addr1 prop-len1 -- prop-addr2 prop-len2 phys.lo ... phys.hi)
0129	push-package	(phandle --)
012A	pop-package	(--)
012B	interpose	(adr len phandle --)
0130	map-low	(phys.lo ... size -- virt)
0131	sbus-intr>cpu	(sbus-intr# -- cpu-intr#)
0150	#lines	(-- rows)
0151	#columns	(-- columns)
0152	line#	(-- line#)
0153	column#	(-- column#)
0154	inverse?	(-- white-on-black?)
0155	inverse-screen?	(-- black?)
0157	draw-character	(char --)
0158	reset-screen	(--)

Table A-48 FCodes by Byte Value (Continued)

Value	Function	Stack
0159	toggle-cursor	(--)
015A	erase-screen	(--)
015B	blink-screen	(--)
015C	invert-screen	(--)
015D	insert-characters	(n --)
015E	delete-characters	(n --)
015F	insert-lines	(n --)
0160	delete-lines	(n --)
0161	draw-logo	(line# addr width height --)
0162	frame-buffer-adr	(-- addr)
0163	screen-height	(-- height)
0164	screen-width	(-- width)
0165	window-top	(-- border-height)
0166	window-left	(-- border-width)
016A	default-font	(-- addr width height advance min-char #glyphs)
016B	set-font	(addr width height advance min-char #glyphs --)
016C	char-height	(-- height)
016D	char-width	(-- width)
016E	>font	(char -- addr)
016F	fontbytes	(-- bytes)
0170	fbl-draw-character	(char --)
0171	fbl-reset-screen	(--)
0172	fbl-toggle-cursor	(--)
0173	fbl-erase-screen	(--)
0174	fbl-blink-screen	(--)
0175	fbl-invert-screen	(--)
0176	fbl-insert-characters	(n --)
0177	fbl-delete-characters	(n --)
0178	fbl-insert-lines	(n --)
0179	fbl-delete-lines	(n --)
017A	fbl-draw-logo	(line# addr width height --)

Table A-48 FCodes by Byte Value (Continued)

Value	Function	Stack
017B	fb1-install	(width height #columns #lines --)
017C	fb1-slide-up	(n --)
0180	fb8-draw-character	(char --)
0181	fb8-reset-screen	(--)
0182	fb8-toggle-cursor	(--)
0183	fb8-erase-screen	(--)
0184	fb8-blink-screen	(--)
0185	fb8-invert-screen	(--)
0186	fb8-insert-characters	(n --)
0187	fb8-delete-characters	(n --)
0188	fb8-insert-lines	(n --)
0189	fb8-delete-lines	(n --)
018A	fb8-draw-logo	(line# addr width height --)
018B	fb8-install	(width height #columns #lines --)
01A4	mac-address	(-- mac-str mac-len)
0201	device-name	(str len --)
0202	my-args	(-- arg-str arg-len)
0203	my-self	(-- ihandle)
0204	find-package	(name-str name-len -- false phandle true)
0205	open-package	(arg-str arg-len phandle -- ihandle 0)
0206	close-package	(ihandle --)
0207	find-method	(method-str method-len phandle -- false xt true)
0208	call-package	(... xt ihandle -- ???)
0209	\$call-parent	(... method-str method-len -- ???)
020A	my-parent	(-- ihandle)
020B	ihandle>phandle	(ihandle -- phandle)
020D	my-unit	(-- phys.lo ... phys.hi)
020E	\$call-method	(... method-str method-len ihandle -- ???)
020F	\$open-package	(arg-str arg-len name-str name-len -- ihandle 0)
0213	alarm	(xt n --)
0214	(is-user-word)	(E: ... -- ???) (name-str name-len xt --)

Table A-48 FCodes by Byte Value (Continued)

Value	Function	Stack
0215	suspend-fcode	(--)
0216	abort	(... --) (R: ... --)
0217	catch	(... xt -- ??? error-code ??? false)
0218	throw	(... error-code -- ??? error-code ...)
0219	user-abort	(... --) (R: ... --)
021A	get-my-property	(nam-str nam-len -- true prop-addr prop-len false)
021B	decode-int	(prop-addr1 prop-len1 -- prop-addr2 prop-len2 n)
021C	decode-string	(prop-addr1 prop-len1 -- prop-addr2 prop-len2 str len)
021D	get-inherited-property	(nam-str nam-len -- true prop-addr prop-len false)
021E	delete-property	(nam-str nam-len --)
021F	get-package-property	(name-str name-len phandle -- true prop-addr prop-len false)
0220	cpeek	(addr -- false byte true)
0221	wpeek	(waddr -- false w true)
0222	lpeek	(qaddr -- false quad true)
0223	cpoke	(byte addr -- okay?)
0224	wpoke	(w waddr -- okay?)
0225	lpoke	(quad qaddr -- okay?)
0226	lwflip	(quad1 -- quad2)
0227	lbflip	(quad1 -- quad2)
0228	lbflips	(qaddr len --)
0230	rb@	(addr -- byte)
0231	rb!	(byte addr --)
0232	rw@	(waddr -- w)
0233	rw!	(w waddr --)
0234	rl@	(qaddr -- quad)
0235	rl!	(quad qaddr --)
0236	wbflips	(waddr len --)
0237	lwflips	(qaddr len --)
023B	child	(phandle.parent -- phandle.child)
023C	peer	(phandle -- phandle.sibling)
023D	next-property	(previous-str previous-len phandle -- false name-str name-len true)

Table A-48 FCodes by Byte Value (Continued)

Value	Function	Stack
023E	byte-load	(addr xt --)
023F	set-args	(arg-str arg-len unit-str unit-len --)
0240	left-parse-string	(str len char -- R-str R-len L-str L-len)
022E	rx@	(xaddr -- o)
022F	rx!	(o xaddr --)
0241	bxjoin	(b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi -- o)
0242	<l@	(qaddr -- n)
0243	lxjoin	(quad.lo quad.hi -- o)
0244	wxjoin	(w.lo w.2 w.3 w.hi -- o)
0245	x,	(o --)
0246	x@	(xaddr -- o)
0247	x!	(o xaddr --)
0248	/x	(-- n)
0249	/x*	(nu1 -- nu2)
024A	xa+	(addr1 index -- addr2)
024B	xal+	(addr1 -- addr2)
024C	xbflip	(oct1 -- oct2)
024D	xbflips	(xaddr len --)
024E	xbsplit	(o -- b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi)
024F	xlflip	(oct1 -- oct2)
0250	xlflips	(xaddr len --)
0251	xlsplit	(o -- quad.lo quad.hi)
0252	xwflip	(oct1 -- oct2)
0253	xwflips	(xaddr len --)
0254	xwsplit	(o -- w.lo w.2 w.3 w.hi)

Table A-49 Tokenizer Directives

Value	Function	Stack
-	(([text<]> --)
-]tokenizer	(--)
-	\	(--)
-	alias	(E: ... -- ???) ("new-name< >old-name< >" --)
-	decimal	(--)
-	external	(--)
-	fload	([filename<cr>] --)
-	headerless	(--)
-	headers	(--)
-	hex	(--)
-	octal	(--)
-	tokenizer[(--)
CR	"	([text<">< >] -- text-str text-len)
CR	'	([old-name< >" -- xt)
CR	(.)	(n -- str len)
CR	."	([text<.>] --)
CR	.(([text<.>] --)
CR	:(colon)	("new-name< >" -- colon-sys) (E: ... -- ???)
CR	;(semicolon)	(--)
CR	<<	(x1 u -- x2)
CR	>>	(x1 u -- x2)
CR	?	(addr --)
CR	[']	([old-name< >] -- xt)
CR	1+	(nu1 -- nu2)
CR	1-	(nu1 -- nu2)

Table A-49 Tokenizer Directives (Continued)

Value	Function	Stack
CR	2+	(nu1 -- nu2)
CR	2-	(nu1 -- nu2)
CR	accept	(addr len1 -- len2)
CR	again	(C: dest-sys --)
CR	allot	(len --)
CR	ascii	([text< >] -- char)
CR	begin	(C: -- dest-sys) (--)
CR	blank	(addr len --)
CR	buffer:	(E: -- a-addr) (len "new-name< >" --)
CR	/c*	(nu1 -- nu2)
CR	cal+	(addr1 -- addr2)
CR	carret	(-- 0x0D)
CR	case	(C: -- case-sys) (sel -- sel)
CR	constant	(E: -- x) (x "new-name< >" --)
CR	control	([text< >] -- char)
CR	create	(E: -- a-addr) ("new-name< >" --)
CR	d#	([number< >] -- n)
CR	.d	(n --)
CR	decimal	(--)
CR	decode-bytes	(prop-addr1 prop-len1 data-len -- prop-addr2 prop-len2 data-addr data-len)
CR	defer	(E: ... -- ???) ("new-name< >" --)
CR	do	(C: -- dodest-sys) (limit start --) (R: -- sys)
CR	?do	(C: -- dodest-sys) (limit start --) (R: -- sys)
CR	3drop	(x1 x2 x3 --)
CR	3dup	(x1 x2 x3 -- x1 x2 x3 x1 x2 x3)
CR	else	(C: orig-sys1 -- orig-sys2) (--)
CR	endcase	(C: case-sys --) (sel <nothing> --)

Table A-49 Tokenizer Directives (Continued)

Value	Function	Stack
CR	eof	(C: case-sys1 of-sys -- case-sys2) (--)
CR	erase	(addr len --)
CR	eval	(... str len -- ???)
CR	false	(-- false)
CR	fcode-version2	(--)
CR	fcode-version3	(--)
CR	field	(E: addr -- addr+offset) (offset size "new-name< >" -- offset+size)
CR	h#	([number< >] -- n)
CR	.h	(n --)
CR	hex	(--)
CR	if	(C: -- orig-sys) (do-next? --)
CR	leave	(--) (R: sys --)
CR	?leave	(exit? --) (R: sys --)
CR	linefeed	(-- 0x0A)
CR	loop	(C: dodest-sys --) (--) (R: sys1 -- <nothing> sys2)
CR	+loop	(C: dodest-sys --) (delta --) (R: sys1 -- <nothing> sys2)
CR	/n*	(nu1 -- nu2)
CR	nal+	(addr1 -- addr2)
CR	not	(x1 -- x2)
CR	of	(C: case-sys1 -- case-sys2 of-sys) (sel of-val -- sel <nothing>)
CR	repeat	(C: orig-sys dest-sys --) (--)
CR	s"	([text<">] -- test-str text-len)
CR	s.	(n --)
CR	space	(--)
CR	spaces	(cnt --)
CR	struct	(-- 0)
CR	then	(C: orig-sys --) (--)

Table A-49 Tokenizer Directives (*Continued*)

Value	Function	Stack
CR	to	(param [old-name< >] --)
CR	true	(-- true)
CR	(u.)	(u -- str len)
CR	until	(C: dest-sys --) (done? --)
CR	value	(E: -- x) (x "new-name< >"--)
CR	variable	(E: -- a-addr) ("new-name< >"--)
CR	while	(C: dest-sys -- orig-sys dest-sys) (continue? --)

FCodes by Name

The following table lists, in alphabetic order, currently-assigned FCodes.

Table A-50 FCodes and Tokenizer Directives by Name

Value	Function	Stack
72	!	(x a-addr --)
CR	"	([text<">< >] -- text-str text-len)
C7	#	(ud1 -- ud2)
C9	#>	(ud -- str len)
CR	'	("old-name< >" -- xt)
-	(([text<)> --)
CR	(.)	(n -- str len)
20	*	(nu1 nu2 -- prod)
1E	+	(nu1 nu2 -- sum)
6C	+!	(nu a-addr --)
D3	,	(x --)
1F	-	(nu1 nu2 -- diff)
9D	.	(nu --)
CR	.">	([text<)>] --)
CR	."<	([text<)>] --)
21	/	(n1 n2 -- quot)
CR	: (colon)	("new-name< >" -- colon-sys) (E: ... -- ???)
CR	; (semicolon)	(--)
3A	<	(n1 n2 -- less?)
96	<#	(--)
CR	<<	(x1 u -- x2)
43	<=	(n1 n2 -- less-or-equal?)
3D	<>	(n1 n2 -- not-equal?)
3C	=	(n1 n2 -- equal?)
0B	>	(n1 n2 -- greater?)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
42	>=	(n1 n2 -- greater-or-equal?)
CR	>>	(x1 u -- x2)
CR	?	(addr --)
6D	@	(a-addr -- x)
CR	[']	([old-name< >] -- xt)
-	\	(--)
-]tokenizer	(--)
A5	0	(-- 0)
36	0<	(n -- less-than-0?)
37	0<=	(n -- less-or-equal-to-0?)
35	0<>	(n -- not-equal-to-0?)
34	0=	(nulflag -- equal-to-0?)
38	0>	(n -- greater-than-0?)
39	0>=	(n -- greater-or-equal-to-0?)
A6	1	(-- 1)
CR	1+	(nu1 -- nu2)
CR	1-	(nu1 -- nu2)
A4	-1	(-- -1)
A7	2	(-- 2)
77	2!	(x1 x2 a-addr --)
59	2*	(x1 -- x2)
CR	2+	(nu1 -- nu2)
CR	2-	(nu1 -- nu2)
57	2/	(x1 -- x2)
76	2@	(a-addr -- x1 x2)
A8	3	(-- 3)
29	>>a	(x1 u -- x2)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
0216	abort	(... --) (R: ... --)
2D	abs	(n -- u)
CR	accept	(addr len1 -- len2)
CR	again	(C: dest-sys --)
0213	alarm	(xt n --)
-	alias	(E: ... -- ???) ("new-name< >old-name< >" --)
AE	aligned	(n1 -- n1 a-addr)
8B	alloc-mem	(len -- a-addr)
CR	allot	(len --)
23	and	(x1 x2 -- x3)
CR	ascii	([text< >] -- char)
12	b(")	(-- str len) (F: /FCODE-string/ --)
11	b(')	(-- xt) (F: /FCODE#/ --)
B7	b(:)	(E: ... -- ???) (F: -- colon-sys)
C2	b(;)	(--) (F: colon-sys --)
A0	base	(-- a-addr)
13	bbranch	(--) (F: /FCODE-offset/ --)
14	b?branch	(don't-branch? --) (F: /FCODE-offset/ --)
BD	b(buffer:)	(E: -- a-addr) (F: size --)
C4	b(case)	(sel -- sel) (F: --)
BA	b(constant)	(E: -- n) (F: n --)
BB	b(create)	(E: -- a-addr) (F: --)
BC	b(defer)	(E: ... -- ???) (F: --)
17	b(do)	(limit start --) (F: /FCODE-offset/ --)
18	b(?do)	(limit start --) (F: /FCODE-offset/ --)
CR	begin	(C: -- dest-sys) (--)
DE	behavior	(defer-xt -- contents-xt)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
AB	bell	(-- 0x07)
C5	b(endcase)	(sel <nothing> --) (F: --)
C6	b(endof)	(--) (F: /FCCode-offset/ --)
44	between	(n min max -- min<=n<=max?)
BE	b(field)	(E: addr -- addr+offset) (F: offset size -- offset+size)
A9	b1	(-- 0x20)
CR	blank	(addr len --)
1B	b(leave)	(F: --)
015B	blink-screen	(--)
10	b(lit)	(-- n) (F: /FCCode-num32/ --)
7F	bljoin	(bl.lo b2 b3 b4.hi -- quad)
15	b(loop)	(--) (F: /FCCode-offset/ --)
16	b(+loop)	(delta --) (F: /FCCode-offset/ --)
B1	b(<mark)	(F: --)
85	body>	(a-addr -- xt)
86	>body	(xt -- a-addr)
1C	b(of)	(sel of-val -- sel <nothing>) (F: /FCCode-offset/ --)
AC	bounds	(n cnt -- n+cnt n)
B2	b(>resolve)	(--) (F: --)
AA	bs	(-- 0x08)
C3	b(to)	(x --)
CR	buffer:	(E: -- a-addr) (len "new-name< >" --)
B8	b(value)	(E: -- x) (F: x --)
B9	b(variable)	(E: -- a-addr) (F: --)
B0	bwjoin	(b.lo b.hi -- w)
02 41	bxjoin	(b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi -- o)
023E	byte-load	(addr xt --)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
75	c!	(byte addr --)
D0	c,	(byte --)
5A	/c	(-- n)
-	/c*	(nu1 -- nu2)
71	c@	(addr -- byte)
5E	ca+	(addr1 index -- addr2)
CR	cal+	(addr1 -- addr2)
62	char+	(addr1 -- addr2)
020E	\$call-method	(... method-str method-len ihandle -- ???)
0208	call-package	(... xt ihandle -- ???)
0209	\$call-parent	(... method-str method-len -- ???)
CR	carret	(-- 0x0D)
CR	case	(C: -- case-sys) (sel -- sel)
0217	catch	(... xt -- ??? error-code ??? false)
65	cell+	(addr1 -- addr2)
69	cells	(nu1 -- nu2)
62	char+	(addr1 -- addr2)
016C	char-height	(-- height)
66	chars	(nu1 -- nu2)
016D	char-width	(-- width)
0236	child	(phandle.parent -- phandle.child)
0206	close-package	(ihandle --)
0153	column#	(-- column#)
0151	#columns	(-- columns)
7A	comp	(addr1 addr2 len -- n)
DD	compile,	(xt --)
CR	constant	(E: -- x) (x "new-name< >" --)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
CR	control	([text< >] -- char)
84	count	(pstr -- str len)
0220	cpeek	(addr -- false byte true)
0223	cpoke	(byte addr -- okay?)
92	cr	(--)
91	(cr	(--)
CR	create	(E: -- a-addr) ("new-name< >" --)
CR	d#	([number< >] -- n)
D8	d+	(d1 d2 --d.sum)
D9	d-	(d1 d2 -- d.diff)
CR	.d	(n --)
-	decimal	(--)
CR	decimal	(--)
021B	decode-int	(prop-addr1 prop-len1 -- prop-addr2 prop-len2 n)
0128	decode-phys	(prop-addr1 prop-len1 -- prop-addr2 prop-len2 phys.lo ... phys.hi)
021C	decode-string	(prop-addr1 prop-len1 -- prop-addr2 prop-len2 str len)
016A	default-font	(-- addr width height advance min-char #glyphs)
CR	defer	(E: ... -- ???) ("new-name< >" --)
015E	delete-characters	(n --)
0160	delete-lines	(n --)
021E	delete-property	(nam-str nam-len --)
51	depth	(-- u)
0201	device-name	(str len --)
011A	device-type	(str len --)
0120	diagnostic-mode?	(-- diag?)
A3	digit	(char base -- digit true char false)
0121	display-status	(n --)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
CR	do	(C: -- dodest-sys) (limit start --) (R: -- sys)
CR	?do	(C: -- dodest-sys) (limit start --) (R: -- sys)
0157	draw-character	(char --)
0161	draw-logo	(line# addr width height --)
46	drop	(x --)
52	2drop	(x1 x2 --)
CR	3drop	(x1 x2 x3 --)
47	dup	(x -- x x)
53	2dup	(x1 x2 -- x1 x2 x1 x2)
CR	3dup	(x1 x2 x3 -- x1 x2 x3 x1 x2 x3)
50	?dup	(x -- 0 x x)
CR	else	(C: orig-sys1 -- orig-sys2) (--)
8F	emit	(char --)
0112	encode+	(prop-addr1 prop-len1 prop-addr2 prop-len2 -- prop-addr3 prop-len3)
0115	encode-bytes	(data-addr data-len -- prop-addr prop-len)
0111	encode-int	(n -- prop-addr prop-len)
0113	encode-phys	(phys.lo ... phys.hi -- prop-addr prop-len)
0114	encode-string	(str len -- prop-addr prop-len)
00	end0	(--)
FF	end1	(--)
CR	endcase	(C: case-sys --) (sel <nothing> --)
CR	endof	(C: case-sys1 of-sys -- case-sys2) (--)
CR	erase	(addr len --)
015A	erase-screen	(--)
CR	eval	(... str len -- ???)
CD	evaluate	(... str len -- ???)
1D	execute	(... xt -- ???)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
33	exit	(--) (R: sys --)
8A	expect	(addr len --)
-	external	(--)
CA	external-token	(--) (F: /FCODE-string FCODE#/ --)
CR	false	(-- false)
0174	fb1-blink-screen	(--)
0177	fb1-delete-characters	(n --)
0179	fb1-delete-lines	(n --)
0170	fb1-draw-character	(char --)
017A	fb1-draw-logo	(line# addr width height --)
0173	fb1-erase-screen	(--)
0176	fb1-insert-characters	(n --)
0178	fb1-insert-lines	(n --)
017B	fb1-install	(width height #columns #lines --)
0175	fb1-invert-screen	(--)
0171	fb1-reset-screen	(--)
017C	fb1-slide-up	(n --)
0172	fb1-toggle-cursor	(--)
0184	fb8-blink-screen	(--)
0187	fb8-delete-characters	(n --)
0189	fb8-delete-lines	(n --)
0180	fb8-draw-character	(char --)
018A	fb8-draw-logo	(line# addr width height --)
0183	fb8-erase-screen	(--)
0186	fb8-insert-characters	(n --)
0188	fb8-insert-lines	(n --)
018B	fb8-install	(width height #columns #lines --)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
0185	fb8-invert-screen	(--)
0181	fb8-reset-screen	(--)
0182	fb8-toggle-cursor	(--)
87	fcode-revision	(-- n)
CR	fcode-version2	(--)
FC	ferror	(--)
CR	field	(E: addr -- addr+offset) (offset size "new-name< >" -- offset+size)
79	fill	(addr len byte --)
CB	\$find	(name-str name-len -- xt true name-str name-len false)
0207	find-method	(method-str method-len phandle -- false xt true)
0204	find-package	(name-str name-len -- false phandle true)
0127	finish-device	(--)
016E	>font	(char -- addr)
-	fload	([filename<cr>] --)
016F	fontbytes	(-- bytes)
0162	frame-buffer-adr	(-- addr)
8C	free-mem	(a-addr len --)
0105	free-virtual	(virt size --)
021d	get-inherited-property	(nam-str nam-len -- true prop-addr prop-len false)
0125	get-msecs	(-- n)
021A	get-my-property	(nam-str nam-len -- true prop-addr prop-len false)
021F	get-package-property	(name-str name-len phandle -- true prop-addr prop-len false)
DA	get-token	(fcode# -- xt immediate?)
CR	h#	([number< >] -- n)
CR	.h	(n --)
-	headerless	(--)
-	headers	(--)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
AD	here	(-- addr)
-	hex	(--)
CR	hex	(--)
95	hold	(char --)
19	i	(-- index) (R: sys -- sys)
CR	if	(C: -- orig-sys) (do-next? --)
020B	ihandle>phandle	(ihandle -- phandle)
015D	insert-characters	(n --)
015F	insert-lines	(n --)
C0	instance	(--)
01 2B	interpose	(adr len phandle --)
0154	inverse?	(-- white-on-black?)
0155	inverse-screen?	(-- black?)
26	invert	(x1 -- x2)
015C	invert-screen	(--)
011C	is-install	(xt --)
011D	is-remove	(xt --)
011E	is-selftest	(xt --)
0214	(is-user-word)	(E: ... -- ???) (name-str name-len xt --)
1A	j	(-- index) (R: sys -- sys)
8E	key	(-- char)
8D	key?	(-- pressed?)
73	l!	(quad qaddr --)
D2	l,	(quad --)
6E	l@	(qaddr -- quad)
02 42	<l@	(qaddr -- n)
5C	/l	(-- n)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
68	/l*	(nu1 -- nu2)
60	la+	(addr1 index -- addr2)
64	la1+	(addr1 -- addr2)
0227	lbflip	(quad1 -- quad2)
0228	lbflips	(qaddr len --)
7E	lbsplit	(quad -- b.lo b2 b3 b4.hi)
82	lcc	(char1 -- char2)
CR	leave	(--) (R: sys --)
CR	?leave	(exit? --) (R: sys --)
0240	left-parse-string	(str len char -- R-str R-len L-str L-len)
0152	line#	(-- line#)
94	#line	(-- a-addr)
CR	linefeed	(-- 0x0A)
0150	#lines	(-- rows)
CR	loop	(C: dodest-sys --) (--) (R: sys1 -- <nothing> sys2)
CR	+loop	(C: dodest-sys --) (delta --) (R: sys1 -- <nothing> sys2)
0222	lpeek	(qaddr -- false quad true)
0225	lpoke	(quad qaddr -- okay?)
27	lshift	(x1 u -- x2)
0226	lwflip	(quad1 -- quad2)
0237	lwflips	(qaddr len --)
7C	lwsplit	(quad -- w1.lo w2.hi)
02 43	lxjoin	(quad.lo quad.hi -- o)
01A4	mac-address	(-- mac-str mac-len)
0130	map-low	(phys.lo ... size -- virt)
0124	mask	(-- a-addr)
2F	max	(n1 n2 -- n1 n2)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
0122	memory-test-suite	(addr len -- fail?)
2E	min	(n1 n2 -- n1 n2)
22	mod	(n1 n2 -- rem)
2A	/mod	(n1 n2 -- rem quot)
0119	model	(str len --)
78	move	(src-addr dest-addr len --)
0126	ms	(n --)
0102	my-address	(-- phys.lo ...)
0202	my-args	(-- arg-str arg-len)
020A	my-parent	(-- ihandle)
0203	my-self	(-- ihandle)
0103	my-space	(-- phys.hi)
020D	my-unit	(-- phys.lo ... phys.hi)
5D	/n	(-- n)
CR	/n*	(nu1 -- nu2)
61	na+	(addr1 index -- addr2)
CR	na1+	(addr1 -- addr2)
B6	named-token	(--) (F: /FCCode-string FCode#/ --)
2C	negate	(n1 -- n2)
011F	new-device	(--)
B5	new-token	(--) (F: /FCCode#/ --)
023D	next-property	(previous-str previous-len phandle -- false name-str name-len true)
4D	nip	(x1 x2 -- x2)
7B	noop	(--)
CR	not	(x1 -- x2)
A2	\$number	(addr len -- true n false)
CR	of	(C: case-sys1 -- case-sys2 of-sys) (sel of-val -- sel <nothing>)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
6B	off	(a-addr --)
CC	offset16	(--)
6A	on	(a-addr --)
0205	open-package	(arg-str arg-len phandle -- ihandle 0)
020F	\$open-package	(arg-str arg-len name-str name-len -- ihandle 0)
24	or	(x1 x2 -- x3)
93	#out	(-- a-addr)
48	over	(x1 x2 -- x1 x2 x1)
54	2over	(x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2)
83	pack	(str len addr -- pstr)
011B	parse-2int	(str len -- val.lo val.hi)
023C	peer	(phandle -- phandle.sibling)
4E	pick	(xu ... x1 x0 u -- xu ... x1 x0 xu)
0110	property	(prop-addr prop-len name-str name-len --)
012A	pop-package	(--)
0129	push-package	(phandle --)
31	r>	(-- x) (R: x --)
32	r@	(-- x) (R: x -- x)
9E	.r	(n size --)
30	>r	(x --) (R: -- x)
0231	rb!	(byte addr --)
0230	rb@	(addr -- byte)
0116	reg	(phys.lo ... phys.hi size --)
CR	repeat	(C: orig-sys dest-sys --) (--)
0158	reset-screen	(--)
0235	rl!	(quad qaddr --)
0234	rl@	(qaddr -- quad)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
4F	roll	(xu ... x1 x0 u -- xu-1 ... x1 x0 xu)
4A	rot	(x1 x2 x3 -- x2 x3 x1)
4B	-rot	(x1 x2 x3 -- x3 x1 x2)
56	2rot	(x1 x2 x3 x4 x5 x6 -- x3 x4 x5 x6 x1 x2)
28	rshift	(x1 u -- x2)
0233	rw!	(w waddr --)
0232	rw@	(waddr -- w)
022E	rx@	(xaddr -- o)
022F	rx!	(o xaddr --)
CR	s"	([text<">] -- test-str text-len)
CR	s.	(n --)
C8	#s	(ud -- 0 0)
9F	.s	(... -- ...)
0131	sbus-intr>cpu	(sbus-intr# -- cpu-intr#)
0163	screen-height	(-- height)
0164	screen-width	(-- width)
023F	set-args	(arg-str arg-len unit-str unit-len --)
016B	set-font	(addr width height advance min-char #glyphs --)
DB	set-token	(xt immediate? fcode# --)
98	sign	(n --)
CR	space	(--)
CR	spaces	(cnt --)
88	span	(-- a-addr)
F0	start0	(--)
F1	start1	(--)
F2	start2	(--)
F3	start4	(--)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
DC	state	(-- a-addr)
CR	struct	(-- 0)
0215	suspend-fcode	(--)
49	swap	(x1 x2 -- x2 x1)
55	2swap	(x1 x2 x3 x4 -- x3 x4 x1 x2)
CR	then	(C: orig-sys --) (--)
0218	throw	(... error-code -- ??? error-code ...)
CR	to	(param [old-name< >] --)
0159	toggle-cursor	(--)
-	tokenizer[(--)
CR	true	(-- true)
4C	tuck	(x1 x2 -- x2 x1 x2)
90	type	(text-str text-len --)
99	u#	(u1 -- u2)
97	u#>	(u -- str len)
9A	u#s	(u1 -- u2)
9B	u.	(u --)
40	u<	(u1 u2 -- unsigned-less?)
3F	u<=	(u1 u2 -- unsigned-less-or-equal?)
3E	u>	(u1 u2 -- unsigned-greater?)
41	u>=	(u1 u2 -- unsigned-greater-or-equal?)
CR	(u.)	(n -- addr len)
58	u2/	(x1 -- x2)
D4	um*	(u1 u2 -- ud.prod)
D5	um/mod	(ud u -- urem uquot)
2B	u/mod	(u1 u2 -- urem uquot)
89	unloop	(--) (R: sys --)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
CR	until	(C: dest-sys --) (done? --)
81	upc	(char1 -- char2)
9C	u.r	(u size --)
0219	user-abort	(... --) (R: ... --)
CR	value	(E: -- x) (x "new-name<>"--)
CR	variable	(E: -- a-addr) ("new-name<>"--)
FD	version1	(--)
74	w!	(w waddr --)
D1	w,	(w --)
6F	w@	(waddr -- w)
5B	/w	(-- n)
67	/w*	(nu1 -- nu2)
70	<w@	(waddr -- n)
5F	wa+	(addr1 index -- addr2)
63	wal+	(addr1 -- addr2)
80	wbflip	(w1 -- w2)
0236	wbflips	(waddr len --)
AF	wbsplit	(w -- b1.lo b2.hi)
CR	while	(C: dest-sys -- orig-sys dest-sys) (continue? --)
0166	window-left	(-- border-width)
0165	window-top	(-- border-height)
45	within	(n min max -- min<=n<max?)
7D	wljoin	(w.lo w.hi -- quad)
0221	wpeek	(waddr -- false w true)
0224	wpoke	(w waddr -- okay?)
0244	wxjoin	(w.lo w.2 w.3 w.hi -- o)
0245	x,	(o --)

Table A-50 FCodes and Tokenizer Directives by Name (Continued)

Value	Function	Stack
0246	x@	(xaddr -- o)
0247	x!	(o xaddr --)
0248	/x	(-- n)
0249	/x*	(nu1 -- nu2)
024a	xa+	(addr1 index -- addr2)
024b	xal+	(addr1 -- addr2)
024c	xbflip	(oct1 -- oct2)
024d	xbflips	(xaddr len --)
024e	xsplrit	(o -- b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi)
024f	xlflip	(oct1 -- oct2)
0250	xlflips	(xaddr len --)
0251	xlsplrit	(o -- quad.lo quad.hi)
25	xor	(x1 x2 -- x3)
0252	xwflip	(oct1 -- oct2)
0253	xwflips	(xaddr len --)
0254	xwsplrit	(o -- w.lo w.2 w.3 w.hi)

FCode Memory Allocation



To get general-purpose memory, use `buffer:` or `alloc-mem`. Use `free-mem` to de-allocate memory obtained with `alloc-mem`.

To map in portions of your SBus device for ordinary access, use "`map-in`" `$call-parent` as in:

```
my-address offset + my-space size " map-in" $call-parent ( virt )
```

To later map out those portions of your device, use "`map-out`" `$call-parent` as in:

```
( virt ) size " map-out" $call-parent
```

To use a region of system memory for DMA (for example, for both direct CPU access and DMA access from a device), first define the following mapping and allocation routines, then follow the steps below to ensure data coherency.

```
: dma-alloc ( size -- virt ) " dma-alloc" $call-parent ;
: dma-free  ( virt size -- ) " dma-free" $call-parent ;
: dma-map-in ( virt size cache? -- devaddr )
  " dma-map-in" $call-parent
;
: dma-map-out ( virt devaddr size -- ) " dma-map-out" $call-parent ;
: dma-sync ( virt devaddr size -- ) \ Correct even if "dma-sync" missing
  " dma-sync" [' ] $call-parent catch if
    2drop 3drop
  then
;
;
```

1. Allocate the DMA region with:

- a. `dma-alloc`
- b. `dma-map-in`

2. CPU accesses the region using `virt` from `dma-alloc`, then perform `dma-sync`.

3. Start DMA operation, using `devaddr` from `dma-map-in`.

- a. **Wait for DMA complete status.**
- b. **Repeat DMA as needed, then perform `dma-sync`**

4. Repeat Steps 2 and 3 as needed

5. Deallocate the region when completed, with:

- a. `dma-map-out`
- b. `dma-free`

Coding Style



This appendix describes the coding style used in some OpenBoot implementations. These guidelines are a living document that first came into existence in 1985. By following these guidelines in your own code development, you will produce code that is similar in style to a large body of existing OpenBoot work. This will make your code more easily understood by others in the OpenBoot community.

Typographic Conventions

The following typographic conventions are used in this document:

- The symbol `⌘` is used to represent space characters (i.e. ASCII 0x20).
- The symbol `...` is used to represent an arbitrary amount of Forth code.
- Within prose descriptions, Forth words are show in `this font`.

Use of Spaces

Since Forth code can be very terse, use spaces to increase readability.

Two consecutive spaces are used to separate a definition's name from the beginning of the stack diagram, another two consecutive spaces (or a newline) are used to separate the stack diagram from the word's definition, and two consecutive spaces (or a newline) separate the last word of a definition from the closing semi-colon. For example:

```
: new-name ( stack-before -- stack-after ) foo bar ;
: new-name ( stack-before -- stack-after )
  foo bar framus dup widget fozzle ribbit grindle
;
```

Forth words are usually separated by one space. If a phrase consisting of several words performs some function, that phrase should be separated from other words/phrases by two consecutive spaces or a newline.

```
: name ( stack before -- stack after ) qq qqq yyy ggg ppp ;
```

When creating multiple line definitions, all lines except the first and last should be indented by three (3) spaces. If additional indentation is needed with control structures, the left margin of each additional level of indentation should start three (3) spaces to the right of the preceding level.

```
: name ( stack before -- stack after )
  qpq...
  qpqp...
  qpqpqp...
  qpqp...
;
```

if...then...else

In if...then or if...else...then control structures that occupy no more than one line, two spaces should be used both before and after each if, else or then.

```
if qq then
if qq else ppp then
```

Longer constructs should be structured like this:

```
<code to generate flag> if
  <>true clause>
then
<code to generate flag> if
  <>true clause>
else
  <>false clause>
then
```

do...loop

In do...loop constructs that occupy no more than one line, two spaces should be used both before and after each do or loop.

```
<code to calculate limits> do qq loop
```

Longer constructs should be structured like this:

```
<code to calculate limits> do
  <body>
loop
```

The longer +loop construct should be structured like this:

```
<code to calculate limits> do
  <body>
  <incremental value> +loop
```

begin...while...repeat

In begin...while...repeat constructs that occupy no more than one line, two spaces should be used both before and after each begin, while or repeat).

```
begin while <body> repeat
```

Longer constructs:

```
begin <short flag code> while
  <body>
repeat
```

```
begin
  <long flag code>
while
  <body>
repeat
```

begin...until...again

In `begin...until` and `begin...again` constructs that occupy no more than one line, two spaces should be used both before and after each `begin`, `until` or `again`.

```
❖❖begin❖❖<body>❖❖until
❖❖begin❖❖<body>❖❖again
```

Longer constructs:

```
begin
❖❖❖<body>
until
```

```
begin
❖❖❖<body>
again
```

Block Comments

Block comments begin with `\.`. All text following the space is ignored until after the next newline. While it would be possible to delimit block comments with parentheses, the use of parentheses is reserved by convention for stack comments.

Precede each non-trivial definition with a block comment giving a clear and concise explanation of what the word does. Put more comments at the very beginning of the file to describe external words which could be used from the User Interface.

Stack Comments

Stack comments begin with `(` and end with `)`. Use stack comments liberally in definitions. Try to structure each definition so that, when you put stack comments at the end of each line, the stack picture makes a nice pattern.

```
: name ( stack before -- stack after )
❖❖❖qqq ppp bar ( stack condition after the execution of bar )
❖❖❖qqq ppp foo ( stack condition after the execution of foo )
```



```
❖❖❖qqq ppp dup ( stack condition after the execution of dup )  
;
```

Return Stack Comments

Return stack comments are also delimited with parentheses. In addition, the notation `r:` is used at the beginning of the return stack comment to differentiate it from a parameter stack comment.

Place return stack comments on any line that contains one or more words that cause the return stack to change. (This limitation is a practical one; it is often difficult to do otherwise due to lack of space.) The words `>r` and `r>` must be paired inside colon definitions and inside `do...loop` constructs.

```
: name ( stack before -- stack after )  
❖❖❖qqq >r ( r: addr )  
❖❖❖qqq r>( r: )  
;
```

Numbers

Hexadecimal numbers should be typed in lower case. If a given number contains more than 4 digits, the number may be broken into groups of four digits with periods. For example:

```
dead.beef
```

Since the default number base is hexadecimal, the convention is not to precede hexadecimal numbers with `h#`.

Differences Between FCode 2.x and 3.x



This appendix discusses the FCodes and macros that have changed between FCode 2.x and FCode 3.x. The 3.x tokenizer will still tokenize code correctly using FCode 2.x names (excepting old #>, # and #s). The function of each of the equivalent FCodes is unchanged. The existing tokenized FCode programs using 2.x FCodes will not be affected on 3.x OpenBoot PROMs. The only functional exception is in the FCode 2.x names #>, #, and #s. FCode 3.x has the same names associated with functionally different FCodes and different byte values. If you have tokenized FCode using the 2.x tokenizer with these FCodes (for instance, #), you will get the same response (since the operation of old # is equivalent to new u# and tokenized code has 0x99 as byte value for your old#) .

Table D-1 FCode Names Changed in Version 3.x

FCode 2.x	FCode 3.x (equivalent)	Byte Value
not	invert	26
<<	lshift	27
>>	rshift	28
cal+	char+	62
nal+	cell+	65
/c*	chars	66
/n*	cells	69
flip	wbflip	80
version	FCode-revision	37

Table D-1 FCode Names Changed in Version 3.x (Continued)

FCode 2.x	FCode 3.x (equivalent)	Byte Value
b(is)	b(to)	C3
eval	evaluate	CD
u*x	um*	D4
xu/mod	um/mod	D5
x+	d+	D8
x-	d-	D9
attribute	property	0110
xdrint	encode-int	0111
xdr+	encode+	0112
xdrphys	encode-phys	0113
xdrstring	encode-string	0114
xdrbytes	encode-bytes	0115
decode-2int	parse-2int	011B
map-sbus	map-low	0130 (stack diag. enhanced)
get-my-attribute	get-my-property	021A
xdrtoint	decode-int	021B
xdrtostring	decode-string	021C
get-inherited-attribute	get-inherited-property	021D
delete-attribute	delete-property	021E
get-package-attribute	get-package-property	02 1F
wflips	wbflips	0236
lflips	lwflips	0237
is	to	

Note – The following 2.x FCodes have changed names. The new 3.x FCodes with the same names function differently.

Table D-2 FCode 2.x Changed Names and Equivalent FCode 3.x Names

FCode 2.x	FCode 3.x (equivalent)	Byte Value
#>	u#>	97
#	u#	99
#s	u#s	9A

So if you are using the 3.x tokenizer to elicit the old response from #>, #, and #s, the source code must be changed so that the commands are replaced by u#>, u# and u#s respectively. For code previously tokenized using the 2.x tokenizer, the result is the same on both OpenBoot 2.x and 3.x PROMs .

Table D-3 FCode 2.x Commands Deleted in FCode 3.x

FCode 2.x	Byte Value
4-byte-id	FE
dma-alloc	0101
memmap	0104
>physical	0106
my-params	010F
intr	0117
driver	0118
group-code	0123
processor-type	0210
firmware-version	0211
fcode-version	0212
probe	0238
probe-virtual	0239

To access the functionality provided by `dma-alloc`, do:

```
: my-dma-alloc ( size -- addr ) " dma-alloc" $call-parent ;
```

and use `my-dma-alloc`.

To access the functionality provided by `memmap`, use `map-low` appropriately.

To replace `intr`, create “`intr`” properties using `property`.

To access the functionality provided by `firmware-version` or `fcode-version`, use `firmware-revision`.

Table D-4 New FCodes Added in 3.x

FCode 3.x	ByteValue
<code>unloop</code>	89
<code>get-token</code>	DA
<code>set-token</code>	DB
<code>state</code>	DC
<code>compile,</code>	DD
<code>behavior</code>	DE
<code>decode-phys</code>	0128
<code>push-package</code>	0129
<code>pop-package</code>	012A
<code>interpose</code>	012B
<code>lwflip</code>	0226
<code>lbflip</code>	0227
<code>lbflips</code>	0228
<code>next-property</code>	023D
<code>byte-load</code>	023E
<code>set-args</code>	023F

Table D-5 Differently Functioning 3.x FCodes With Changed Byte Values

FCode 3.x	Byte Value
<code>#</code>	C7
<code>#s</code>	C8
<code>#></code>	C9

Table D-6 3.x FCodes Related to 64-bit Operations

3.x FCodes	Stack Diagrams	Byte Value
rx@	(oaddr -- o)	022E
rx!	(o oaddr --)	022F
bxjoin	(b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi -- o)	0241
<l@	(qaddr -- n)	0242
lxjoin	(quad.lo quad.hi -- o)	0243
wxjoin	(w.lo w.2 w.3 w.hi -- o)	0244
x,	(o --)	0245
x@	(oaddr -- o)	0246
x!	(o oaddr --)	0247
/x	(-- n)	0248
/x*	(nu1 -- nu2)	0249
xa+	(addr1 index -- addr2)	024A
xal+	(addr1 -- addr2)	024B
xbflip	(oct1 -- oct2)	024C
xbflips	(oaddr len --)	024D
xbsplit	(o -- b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi)	024E
xlflip	(oct1 -- oct2)	024F
xlflips	(oaddr len --)	0250
xlsplit	(o -- quad.lo quad.hi)	0251
xwflip	(oct1 -- oct2)	0252
xwflips	(oaddr len --)	0253
xwsplit	(o -- w.lo w.2 w.3 w.hi)	0254

The following device-handling-related user interface commands have changed between OpenBoot 2.x and OpenBoot 3.x. Their functional behavior is the same. Determine your system's OpenBoot PROM version by entering `.version` at the `ok>` prompt, then using the appropriate commands from the following table.

Table D-7 Device-related User Interface Commands Changed in 3.x

OpenBoot 2.x Command	OpenBoot 3.x Command
<code>.attributes</code>	<code>.properties</code>
<code>cd</code>	<code>dev</code>
<code>reset</code> (to reset full system)	<code>reset-all</code>

Index

Symbols

"alternate-reg", 92
"assigned-addresses", 92
"big-endian-aperture", 93
"fast-back-to-back", 97
"has-fcode", 97
:, 12
;, 13
['], 63

Numerics

0xfd, 2
3.x tokenizer, 50
66Mhz-capable, 9
68, 57

A

a.out header, 5
accessing
 packages, 74
accessing a PCI device's configuration
 space registers, 45
active package, 58
adding a PCI header to a PROM, 45
address, 90

address-bits, 90
#address-cells, 89
addressing
 packages, 76
 SBus, 161
ANS Forth
 and FCode, 11
apply, 69
assigned-addresses property, 9
auto-boot?, 28
available, 92

B

begin-package, 34, 75, 77
begin-select-dev, 40
binary executable programs, 32
binary format
 FCode, 11
boot scenario, Solaris, 51
boot software roles, 51
booting FCode image, 5
buffer:, 62
byte-load, 35

C

- cache-line-size, 9
- \$call-method, 60, 64, ?? to 69
- call-package, 64
- \$call-parent, ?? to 69
- map-in \$call-parent, 44
- character-set, 93
- class-code, 9
- code examples
 - \$open-package, 65
 - begin-package, 77
- data
 - instance-specific, 74
 - static, 73
- find-package, 64
- open-dev, 76
- reg, 19
- colon definition, 12 to 13
- command line editor, 30
- compatible, 94
- compile state, 12
- configuration space base address, 51
- configuration space command register, 52
- configuration variables
 - auto-boot?, 28
 - fcode-debug?, 28
- CPU PROM-generated properties, 8
- current instance, 58

D

- data
 - initialized, 62
 - instance-specific, 62
 - package, 61
 - packages, 58
 - static, 62
 - zero-filled, 62
- data definition
 - packages, 73
- deblocker support, 81
- decode-unit, 63, 157

- defer, 62
- defining
 - Forth words, 12
- depth, 94
- dev, 36, 58
- device
 - drivers, plug-in, 70
 - identification, 2
 - node, 3
 - tree, 3
- device addressing
 - SBus, 161
- device methods
 - block-size, 122
 - decode-unit, 157
 - dma-alloc, 158
 - dma-free, 158, 237
 - dma-map-in, 159
 - dma-map-out, 159
 - dma-sync, 160
 - load, 122, 192
 - map-in, 161
 - map-out, 161
 - max-transfer, 123
 - probe-self, 160
 - read, 123, 193, 237
 - read-blocks, 123
 - seek, 123
 - write, 124, 193
 - write-blocks, 124
- device node
 - browsing, 36
 - creating with begin-package, 34
 - creating with end-package, 34
- device property generation, 51
- device_type, 95
- device-end, 36
- device-id, 9
- devsel-speed, 9, 96
- dloading FCode image, 5
- dma-alloc, 158
- dma-free, 158, 237
- dma-map-in, 159

`dma-map-out`, 159

`dma-sync`, 160

`driver`

and boot PROM, 1

function, 1

E

`end0`, 2

`end1`, 2

`end-package`, 35

`execute-device-method`, 41, 69

`executing`

methods, 63

`execution token`, 63

obtaining, 63

`expansion FCode PROM access`, 53

F

`fakeboot`, 5

`fast-back-to-back`, 9

`FCode`

`/w`, 391

and ANS Forth, 11

binary format, 11

characteristics, 11

compile state, 12

defining words, 12

device identification, 2

in PROM, 1

interpret state, 12

interpretation, 2

interpreting, 34

one-byte, 20

programming style, 14 to 16

property-specific FCodes, 118

source format, 11

stack, 12

tokenizing, 12

two-byte, 20

valid program, 2

`w!`, 390

`w,`, 390

`w@`, 391

words, 11

`FCode programs`, 32

testing in source form, 42

`FCode PROM`

body, 2

end token, 2

header, 2

magic number, 2

organization, 2

size, 2

`FCode source`, 25 to 26

`FCode types`

interface, 21

local, 22

primitives, 20

`fcode-debug?`, 24, 28

`FCodes`

`-`, 260

`#`, 257

`#>`, 237, 238, 257

`#columns`, 290

`#line`, 348

`#lines`, 348

`#out`, 363

`#s`, 374

`$call-method`, 58, 60, 64, ?? to 69,
284

`$call-parent`, ?? to 69, 285

`$find`, 325

`$number`, 361

`$open-package`, 60, 65, 75, 362

`'`, 258

`(`, 258

`(cr`, 293

`(is-user-word)`, 342

`*`, 259

`+`, 259

`+!`, 259

`+loop`, 349

`,`, 259

`..`, 260

`.r`, 367

`.s`, 374

`/`, 261

`/c`, 283

/l, 345
/l*, 345
/mod, 354
/n, 358
/w*, 391
<, 262
<#, 262
<=, 262
<>, 263
<w@, 391
=, 263
>, 263
>=, 263
>>a, 268
>body, 280
>font, 328
>r, 368
?dup, 304
@, 264
0, 265
0<, 265
0<=, 265
0<>, 265
0=, 266
0>, 266
0>=, 266
-1, 266
1, 266
2, 267
2!, 267
2*, 267
2/, 267
2@, 268
2drop, 303
2dup, 304
2over, 363
2rot, 372
2swap, 380
3, 268
abort, 268
abs, 269
alarm, 269
aligned, 271
alloc-mem, 271
and, 272
b("), 272
b('), 272
b(+loop), 279
b(:), 272
b(;), 273
b(<mark), 279
b(>resolve), 280
b(?do), 276
b(case), 275
b(constant), 275
b(create), 275
b(defer), 275
b(do), 276
b(endcase), 277
b(endof), 277
b(field), 277
b(leave), 278
b(lit), 279
b(loop), 279
b(of), 280
b(to), 281
b(value), 281
b(variable), 281
b?branch, 275
base, 273
bbranch, 274
behavior, 276
bell, 277
between, 277
bl, 277
blink-screen, 278
bljoin, 279
body>, 279
bounds, 280
bs, 280
bwjoin, 281, 282
byte-load, 282
c!, 282
c., 282
c@, 283
ca+, 283
call-package, 64, 284
catch, 286
cell+, 288
cells, 288
char-height, 288
chars, 288

char-width, 288
child, 289
close-package, 289
column#, 290
comp, 291
compile,, 291
count, 292
cpeek, 292
cpoke, 293
cr, 293
d-, 295
d+, 294
decode-int, 296
decode-phys, 296
decode-string, 296
default-font, 297
delete-characters, 299
delete-lines, 299
delete-property, 300
depth, 300
device-name, 300
diagnostic-mode?, 301
digit, 301
draw-character, 302
draw-logo, 303
drop, 303
dup, 304
emit, 304
encode+, 305
encode-bytes, 306
encode-int, 306
encode-phys, 306
encode-string, 307
end0, 307
endl, 307
erase-screen, 308
evaluate, 308
execute, 309
exit, 309
expect, 310
external-token, 311
fb1-blink-screen, 311
fb1-delete-lines, 312
fb1-draw-character, 312
fb1-draw-logo, 313
fb1-erase-screen, 313
fb1-insert-characters, 313
fb1-insert-lines, 314
fb1-install, 314
fb1-invert-screen, 315
fb1-reset-screen, 316
fb1-slide-up, 316
fb1-toggle-cursor, 316
fb8-blink-screen, 317
fb8-delete-characters, 317
fb8-delete-lines, 318
fb8-draw-character, 318
fb8-draw-logo, 318
fb8-erase-screen, 319
fb8-insert-characters, 319
fb8-insert-lines, 319
fb8-install, 320
fb8-invert-screen, 321
fb8-reset-screen, 321
fb8-toggle-cursor, 321
fcode-revision, 322
ferror, 323
fill, 325
find-method, 63, ?? to 69, 326
find-package, 64, 326
finish-device, 36, 74, 327
fontbytes, 328
frame-buffer-adr, 329
free-mem, 330
free-virtual, 330
get-inherited-property, 330
get-msecs, 331
get-my-property, 331
get-package-property, 332
get-token, 332
here, 334
hold, 334
i, 335
ihandle>phandle, 336
insert-characters, 336
insert-lines, 336
instance, 62, 337
inverse?, 337
inverse-screen?, 338
invert, 338
invert-screen, 339
is-install, 339

is-remove, 341
is-selftest, 342
j, 343
key, 343
key?, 344
l!, 344
l,, 344
l@, 344
la+, 345
lal+, 345
lbflip, 345
lbflips, 345
lbsplit, 346
lcc, 346
left-parse-string, 75, 347
line#, 348
lpeek, 350
lpoke, 350
lshift, 350
lwflip, 351
lwflips, 351
lwsplit, 351
mac-address, 351
map-low, 352
mask, 352
max, 353
memory-test-suite, 353
min, 353
mod, 353
model, 115, 354
move, 355
ms, 355
my-address, 356
my-args, 75, 356
my-parent, 357
my-self, 64, 357
my-space, 357
my-unit, 77, 358
na+, 358
named-token, 359
negate, 359
new-device, 59, 360
new-token, 360
next-property, 360
nip, 360
noop, 360
off, 361
on, 362
open-package, 64, 75, 362
or, 363
over, 363
pack, 363
parse-2int, 364
peer, 364
pick, 365
property, 115, 365
r>, 367
r@, 367
rb!, 368
rb@, 369
reg, 369
reset-screen, 370
rl!, 371
rl@, 371
roll, 371
-rot, 372
rot, 372
rshift, 372
rw!, 372, 373
rw@, 373
sbus-intr>cpu, 374
screen-height, 375
screen-width, 375
set-args, 75, 375
set-font, 376
set-token, 376
sign, 377
span, 377
start0, 378
start1, 378
start2, 378
start4, 379
state, 379
suspend-fcode, 379
swap, 380
throw, 380
toggle-cursor, 381
tuck, 383
type, 383
u#, 383
u#>, 384
u#s, 384

u., 384
u.r, 387
u/mod, 386
u<, 384
u<=, 385
u>, 385
u>=, 385
u2/, 385
um*, 386
um/mod, 386
unloop, 386
upc, 387
user-abort, 388
version1, 389
w!, 391
wa+, 391
wal+, 392
wbflip, 392
wbflips, 392
wbsplit, 392
window-left, 393
window-top, 394
within, 394
wljoin, 394
wpeek, 395
wpoke, 395
xor, 395, 397
find-device, 58
find-method, 63, ?? to 69, 326
find-package, 64, 326
finish-device, 36, 74, 327
fload, 26, 327
Forth
 compile state, 12
 interpret state, 12
 programs, 33
 stack, 12
 tokenizing, 12
 words, 11

G
get-inherited-property, 36
get-my-property, 36

H

height, 97

I

ihandle, 64
 avoiding confusion with
 phandle, 66
initialized data, 62
instance
 arguments, 74
 creation, 58
 package, 58, 58
 parameters, 74
instance, 62
instance chain, 59
instance-specific
 data, 62
 methods, 63
interpret, 57
interpret state, 12
interpreting FCode, 2, 34 to 36

L

left-parse-string, 75
linebytes, 98
loading/executing files
 Forth over serial port A, 33
local-mac-address, 100
ls, 36

M

mac-address, 100
map?, 44
map-in, 48, 161
map-out, 161
mapping
 packages, 77
max-frame-size, 101
max-latency, 9
methods

- calling other package methods, 66
- executing, 63
- instance-specific, 63
- package, 58

min-grant, 9

model, 102, 115

my-args, 75

my-self, 64

my-unit, 77

N

name, 102

new-device, 59

node

- machine, 107
- SBus, 107
- SCSI, 107

NVRAM parameters

- setting, 28

NVRAM variable fcode-debug?, 24

nvrामrc, 77

O

open-dev, 60

\$open-package, 60, 65, 66, 75, 362

open-package, 64, 75, 362

organizationally unique identifier, 19

OUI, 19

P

package, 57

- deblocker, 81
- TFTP, 80

package method

- reset, 71, 72
- selftest, 72

package methods

- reset, 71, 72

/packages, 64, 65, 78

packages

- accessing, 74

- active, 58
- addressing, 76
- and linking, 57
- data, 58, 61
- data definition, 73
- instance, 58
- instances, 58
- interface, 58
- mapping, 77
- methods, 58, 71
- plug-in, 57
- properties, 58

packaging PCI FCode, 23

PCI

- data structure, 5
- device configuration register
 - access, 8
- FCode PROM header format, 5

PCI expansion PROM

- data structure, 6
- header, 5
- header format, 6

pcia, 46

pcia-probe-list, 49

pcib, 46, 49

pcib-probe-list, 49

pcimsg?, 24

phandle, 332

- avoiding confusion with
 - ihandle, 66

phandles, 64

phys.hi, 7, 48

phys.hi cell, 51

phys.lo, 7, 48

phys.mid, 7, 48

physical addresses, finding and using, 47

plug-in

- device drivers, 70
- package, 57
- PCI device physical address, 49

probe, 57

probemsg?, 24

probe-self, 160

probing sequence, 78
 expansion bus, 29
 modifying with NVRAM script, 29
 programming style
 FCode, 14 to 16
 PROM
 contents, 1
 `properties, 36
 properties
 "alternate-reg", 92
 "assigned-addresses", 92
 "big-endian-aperture", 93
 "fast-back-to-back", 97
 "has-fcode", 97
 #address-cells, 89
 #size-cells, 89
 address, 90
 address-bits, 90
 available, 92
 block or byte device, 125
 character-set, 93
 compatible, 94
 depth, 94
 device_type, 95
 devsel-speed, 96
 display device, 86, 146
 height, 97
 linebytes, 98
 list, 3
 local-mac-address, 100
 mac-address, 100
 max-frame-size, 101
 memory device, 87
 model, 102
 modifying from User Interface, 78
 name, 3, 102, 359
 network device, 87, 193
 packages, 58
 parent node, 87
 ranges, 105
 reg, 110, 358
 serial device, 238
 vendor-id, 115
 width, 115
 property
 creation, 85, 115
 decoding, 117
 encoding, 116
 modification, 115
 name, 83
 property value, 115
 ranges, 107
 reg, 107
 retrieval, 116
 standard names, 86
 value, 3, 83
 value array formats, 84
 property, 115
 pwd, 36

R
 ranges, 105, 107
 rb!, 368
 rb@, 369
 reg, 107, 110, 358, 369
 reg property physical address format, 7, 8
 reset, 71, 72
 restricting system use, 25
 reverse polish notation, 12
 revision-id, 9
 rl!, 371
 rl@, 371
 rw!, 372, 373
 rw@, 373

S
 SBus
 addressing, 161
 node, 107
 SBus addressing, 161
 SCSI
 node, 107
 see, 36, 39
 select-dev, 38 to 40, 60
 selftest, 72

set fcode-verbose? from
 NVRAMRC, 24
set-args, 75
show-devs, 36
size
 FCode PROM, 2
size.hi, 7
size.lo, 7
#size-cells, 89
source format
 FCode, 11
stack, 12
 comments, 13, 16
 operation, 13
standard methods
 decode-unit, 63
standard support packages, 78
state, 379
static data, 62
subsystem-id, 9
subsystem-vendor-id, 9
Sun Ultra-1 UPA/PCI-related nodes, 46
system cache line size, 45
system flags and FCode debugging, 24

T

TFTP
 support, 80
tokenizer, 33
 description, 25
 directives, 21
 macros, 20
tokenizer directives
 .(, 260
 \, 264
]tokenizer, 382
 alias, 270
 decimal, 295
 emit-byte, 305
 external, 310
 false, 311
 fload, 26, 327

headerless, 333
headers, 333
hex, 334
offset16, 361
tokenizer[, 382
tokenizer macros
 ", 256
 (.), 258
 (u.), 385
 .", 260
 .d, 295
 .h, 333
 /c*, 283
 /n*, 358
 :, 261
 i, 261
 <<, 262
 >>, 263
 ?, 263
 ?do, 302
 ?leave, 347
 ['], 63, 264
 1-, 266
 1+, 266
 2-, 267
 2+, 267
 3drop, 304
 3dup, 304
 accept, 269
 again, 269
 allot, 271
 ascii, 272
 begin, 276
 blank, 278
 buffer:, 62, 281
 cal+, 283
 carret, 285
 case, 285
 constant, 291
 control, 292
 create, 293
 d#, 294
 decimal, 295
 decode-bytes, 295
 defer, 62, 297

do, 302
else, 304
endcase, 307
endof, 308
erase, 308
eval, 308
fcode-version1, 322
fcode-version2, 323
field, 324
h#, 332
if, 335
leave, 346
linefeed, 348
loop, 349
nal+, 359
not, 360
of, 361
repeat, 370
s", 373
s., 374
space, 377
spaces, 377
struct, 379
then, 380
to, 381
true, 383
until, 387
value, 62, 388
variable, 62, 389
while, 393
tokenizing, 12

U

udf-supported, 9
unit-address, 358
unselect-dev, 40
unselect-device, 36
User Interface
 */, 259
 */mod, 354
 [, 264
 [compile], 291
], 265
 apply, 69

begin-package, 75
browsing device nodes, 36
 `properties, 36
 dev, 36
 device-end, 36
 get-inherited-
 property, 36
 get-my-property, 36
 ls, 36
 pwd, 36
 see, 36
 show-devs, 36
 words, 36
byte-load, 35
command line editor, 30
dev, 58
end-package, 35
execute-device-method, 69
find-device, 58
interpreting FCode, 34 to 36
modifying properties, 78
nvramrc, 77
open-dev, 60
s", 373
testing a device driver, 38 to 42
 begin-select-dev, 40
 execute-device-method, 41
 patch, 40
 see, 39
 select-dev, 38
 unselect-dev, 40
unselect-device, 36

V

value
 of property, 115
 property, 3
value, 62
variable, 62
variable fcode-debug?, 24
vendor-id, 8, 115
Vital Product Data, 7, 53

W

width, 115

words

 FCode, 11

 Forth, 11

words, 36

X

xxx clip-num, 50

Copyright 1997 Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, Californie 94043-1100 U.S.A.

Tous droits réservés. Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie et la décompilation. Aucune partie de ce produit ou de sa documentation associée ne peut être reproduits 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.

Des parties de ce produit pourront être dérivées du système UNIX® et du système Berkeley 4.3 BSD licencié par l'Université de Californie. UNIX est une marque enregistrée aux Etats-Unis et dans d'autres pays, et licenciée exclusivement par X/Open Company Ltd. 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.

Sun, Sun Microsystems, le logo Sun, et Solaris sont des marques déposées ou enregistrées par Sun Microsystems, Inc. aux Etats-Unis et dans certains d'autres pays. Toutes les marques SPARC, utilisées sous license, sont des marques déposées ou enregistrées 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.

Les utilisateurs d'interfaces graphiques OPEN LOOK® et Sun™ ont été développés 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, cette licence couvrant aussi les licenciés de Sun qui mettent en place OPEN LOOK GUIs et qui en outre se conforment aux licences écrites de Sun.

Le système X Window est un produit du X Consortium, Inc.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" SANS GARANTIE D'AUCUNE SORTE, NI EXPRESSE NI IMPLICITE, Y COMPRIS, ET SANS QUE CETTE LISTE NE SOIT LIMITATIVE, DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DES PRODUITS A REpondre A UNE UTILISATION PARTICULIERE OU LE FAIT QU'ILS NE SOIENT PAS CONTREFAISANTS DE PRODUITS DE TIERS.