Technical Note TN1061

Fundamentals of Open Firmware, Part I: The User Interface

CONTENTS

This Technote describes the Open Firmware User Interface and Forth, the Open Firmware language. This Note addresses how to connect the target machine to a host machine for two machine mode. It explains how to use the interface to go between two and one machine mode. It also provides an explanation of the basics of Forth, an introduction to the device tree, and some debugging techniques.

This Technote is targeted at the expansion device designer and the driver writer for that device. The reader should have an understanding of Open Firmware as described by the IEEE 1275-1994 Specification, the PCI Local Bus Specification 2.0+, the PCI Bus Binding Specification 1.5, and *Designing PCI Cards and Drivers for Power Macintosh Computers* .

This purpose of this Technote is to supplement the sparse documentation covering Open Firmware on the Macintosh.

> **Note:**
> A Technote is insufficient to explain all features of Open Firmware that are required by a driver writer and/or a board designer to comprehend, in order to complete an expansion device project.

Note:
This Note is written using a 9500 Macintosh. Your machine may be somewhat different with respect to display output. For instance, the 9500 PCI series machines are PCI Local Bus Specification 2.0 compliant. Newer machine may be compliant to version 2.1.

Updated: [Jul 01 2000]

About the Open Firmware User Interface

Open Firmware is the process that controls the microprocessor after hardware initialization and diagnostics are performed, but before the main Operating System is passed control. It is responsible, among other things, for building the device tree and probing the expansion slots for I/O devices. Open Firmware queries PCI devices for its address space needs and dynamically assigns this space to each device. It is during this probing process that each device and motherboard ASIC is given a node in the device tree.

Nodes, which are also called **packages**, contain **properties** and **methods**. **Properties** are attributes that describe the hardware and driver. **Methods** do work much like subroutines or procedures. The hardware and software engineer can use the Open Firmware user interface to debug their device and driver, respectively. See Technote 1044, *Understanding PCI Expansion Choices for Mac OS 8, Part III* in the Open Firmware Technote Series, for details about properties and methods for various devices. You must be able to traverse the device tree to get to your node and then to edit and debug that node.

Forth is the human interface language to Open Firmware and the device tree. If you're a board designer, you'll want to directly read and write registers on your device, and, therefore, must be able to move throughout the device tree, create and delete words, etc. The driver writer has similar needs and must also build an **FCode** representation of the driver properties and methods. If the device contains a **boot driver**, that driver must be debugged using the Open Firmware user interface.

A **boot driver** is written in Forth, then tokenized into FCode and debugged from the interface. This form of driver is used during the earliest stages of the boot process before an operating system is available. Boot drivers are typically display, keyboard, network, and block, but are not limited to these.

The Open Firmware user interface, therefore, as specified by the IEEE1275-1994 Specification, is required to allow board designers and driver writers access to their hardware and software to build and debug their expansion device project. Let's see how that works.

Back to top

Connecting for Two-Machine Mode

The Open Firmware user interface, hereafter called the interface, as implemented on Macintoshes to date, comes up in two-machine mode. You can connect the two machines together using the serial ports and cable. Start with the modem ports for simplicity. Open Firmware defaults to the modem port. Use a communication application such as Zterm, MacTerminal, or Microphone as the host. Use these settings:

- 38400 baud (typically 38.4 but some versions of OF may use 19.2 also)
- No parity
- 8 data bits

- 1 stop bit
- XON/XOFF handshake
- ANSI/VT102 terminal protocol

1. Start the host and wait for the prompt.
2. Start the target machine while holding down the O, F, Option, and Command keys. (O, F stands for Open Firmware.)
3. Wait until you see:

```
Open Firmware, 1.0.5
To continue booting the MacOS type:
BYE
To continue booting from the default boot device type:
BOOT
ok
0 >
```

(This prompt will vary depending on the version of OF running on the machine. For example, some newer machines, OF version is 2.0.x.)

The O.K. means the interpeter is waiting for keyboard input. The 0 indicates the top of the stack.

Going between one and two machine modes

To move from two machine to one machine mode during an individual session, enter the following redirection words:

```
0 > " pci2/@f" output    \ the path must point to your display node
0 > " kbd" input
```

> **Note:**
> " kbd" is standard alias supported on all the machines. But there is no standard alias for display screen. The redirection of output is specific to power surge machines. You will have to decide what is the display for the target machine and use that path name.

Now, I must point out that after the output was directed to the target machine what you enter at the host for the second word (i.e., input) will not appear on your host display but on your target display. You can now enter you session on the target machine until you restart your target. Once you restart, your input and output capabilities will again be at the host. To make the change permanent, used the printenv and setenv words. What follows is what the printenv word displays. Note that there are two untitled columns. The left column displays current setting and the right diplays the default. You must change the environmental variables called input-device and output-device to contain the path name to your keyboard and display, respectively. Then, when you restart your target, you'll always be in one machine mode. Of course, since these variables are stored in NVRAM, you can reset them using the Option-Command-P-R keys upon restart. I'll leave this procedure to you, but let's look at one more variable.

Auto-boot? is its name and if you set auto-boot? to false, you no longer have to hold down the Option-Command-O-F keys upon restart.

```
0 > printenv

little-endian?      false                   false
real-mode?          false                   false
auto-boot?          true                    true
diag-switch?        false                   false
fcode-debug?        false                   false
oem-banner?         false                   false
oem-logo?           false                   false
use-nvramrc?        true                    false
real-base           -1                      -1
real-size           100000                  100000
virt-base           -1                      -1
virt-size           100000                  100000
load-base           4000                    4000
pci-probe-list      -1                      -1
screen-#columns     64                      64
screen-#rows        28                      28
selftest-#megs      0                       0
boot-device         /AAPL,ROM               /AAPL,ROM
boot-file
diag-device         fd: diags                fd: diags
diag-file
input-device        ttya                    ttya
output-device       ttya                    ttya
oem-banner
oem-logo
boot-command        boot                    boot
ok
0 >
```

You are ready to use the interface and at this point and a few words about Forth are needed.

Back to top

An Introduction to Forth

Forth is a stack-based interpretive language that uses reverse Polish notation. You place operands onto the stack and then operate on the operands by entering words. Words are delimited by white spaces.

Your First Forth Operation

Let's look at a simple example using four function arithmitic. We will add 5 and 7 to get 12.

1. Enter 5 CR (carriage return),
2. enter 7 CR
3. enter + CR
4. enter . CR

Your display should now look as follows:

```
0 > 5   ok
1 > 7   ok
2 > +   ok
1 > . C   ok
```

. . . But where is the correct answer, which is 12?

Setting Forth's Numeric Base

The C means OxC for hexidecimal. The Open Firmware user interface defaults to a hexidecimal numeric-base. The radix can be hexidecimal, decimal, or octal and can be changed as follows:

hex	means interpet all input and output in hexidecimal.
decimal	means interpet all input and output in decimal.
octal	means interpet all input and output in octal.

If you want to change only the next input or output character and then return to the existing numeric-base, use these words.

h#	means interpet the following number as hexadecimal.
d#	means interpet the following number as decimal.
o#	means interpet the following number as octal.

Also, there is the .d and .h equivalent words for displaying.

To display the current base, output in decimal, enter:

```
0 > base @ .d 16k
0 >
```

At this point, all the following numbers in this Technote will be decimal for convienence.

Try the above example again, but this time begin by changing the default to decimal, and your results will look like the following:

```
0 > decimal   ok
0 > 5   ok
1 > 7   ok
2 > +   ok
1 > . 12   ok
0 >
```

Since Forth interpets a word using white space delimiters, we can type more than one word on a line as follows:

```
0 > 5 7 + . 12  ok
0 >
```

The .s Word

Now try the same commands, substituting the <.> with a <.s>. Actually, you enter a period for <.>.

```
0 > 5 7 + .s 12
ok
1 >
```

This shows us two things:

1. Forth words can be punctuation marks, or even numbers; from now on in this Technote, we will surround Forth words with <>, for clarity.
2. <.s> displays the entire stack without removing items from the stack, while <.> displays the top item on the stack and removes it.

For instance, look at the following sequence of numbers and words.

```
0 > 1 2 3 4 . 4  ok
3 > .s 1 2 3
ok
3 > . . .  3 2 1  ok
0 >
```

1 then 2 then 3 then 4 were put onto the stack, and then the top item, 4, was displayed. <.> removed the 4 and displayed it. This made the stack depth go to 3. When <.s> was entered, the entire stack was displayed, but no item was removed. <.s> is very useful when debugging, since it can let you see what you actually entered onto the stack before executing a word.

Unknown Words

The Forth Interpeter is not forgiving about clearing the entire stack when you type a word the interpeter doesn't understand. This can be very frustrating, as shown in the next example.

```
0 > 1 2 3 4   ok
4 > bad-word bad-word, unknown word
ok
0 >
```

Since bad-word is not a word in the Forth dictionary, the interpeter cleared the entire stack. Usually, there are ways to return to a command when this occurs. Open Firmwares terminal emulator allows the use of the arrow keys to return to previous lines so that you may edit a line before trying the offending or unknown word.

More Forth Words

You can determine the depth of the stack, clear an item, clear the entire stack, rotate an item, etc. All of these commands are detailed in Section 7, User Interface, of the IEEE 1275 Specification. Look at the following:

```
0 > 1 2 3 4 depth .s 1 2 3 4 4
ok
5 > clear .s Empty
ok
0 >
```

Four numbers were placed on the stack and then the < depth > word was executed. As you might expect, this word determined how deep the stack was and then placed that number on the top of the stack. <.s> displayed the entire stack, including the result of < depth >. < clear > returned the stack depth to zero, as can be seen by its output (i.e., Empty) when displayed with the < .s > word.

Stack Notation

But how do you know what a word does before executing it? Stack notation is the answer. Forth stack notation is a formal statement of what is on the stack before and after the word is executed. It does not report the stack contents below what is needed by the word. Each word has a notation that attempts to clarify what the word does. The stack notations for < + > and < depth > are shown next.

+ (nu1 nu2 -- sum)
depth (-- u)

Now, let's look at the format of a stack notation. It begins with the < (> led and followed by a white space, so it's a word. It informs the Forth enterpeter to ignore all characters until it sees the <) > word, which does not need to be delimited by white spaces at the beginning and end of itself. The -- is a separator of the item on the stack before and after execution of a word. Item(s) to the left are on the stack before the word is executed and item(s) to the right are placed on the stack by the word.

So < + > then takes the two top most values off the stack, adds them, and returns the sum to the stack. Note that there can be many more values on the stack below the two topmost in this example. < depth > determines the number of values on the stack and places an unsigned number on the top of the stack.

Duplication, Rearrangement, and Removal of Stack Items

So far we have looked at items or values on the stack. A single stack value may not be sufficient to describe an item. Some items have two or more values. Let's look at stack duplication, rearrangement, and removal of single and double stack items next. Enter the following command line:

```
0 > 1 2 3 4  ok
4 > dup  ok
5 > .s 1 2 3 4 4
ok
5 >
```

So < dup > duplicated the top item on the stack, which can be seen from entering the < .s > command.

Now enter the following:

```
5 > clear 1 2 3 2dup .s 1 2 3 2 3
ok
5 > 3dup .s 1 2 3 2 3 3 2 3
ok
8 >
```

< clear > emptied the stack, 1 2 3 were entered, and then the 2 and 3 were duplicated using the < 2dup > word. The stack

then produced 1 2 3 2 3 when < 3dup > was entered. At this point there were eight items on the stack.

Next, remove those items:

```
8 > drop .s 1 2 3 2 3 3 2
ok
7 > 2drop .s 1 2 3 2 3
ok
5 > 3drop .s 1 2
ok
2 > nip .s 2
ok
1 >
```

You can drop one, two, or three items from the stack and you can nip it also as shown above. Here are the stack notations for your analysis of the previous example:

drop	(x --)
2drop	(x1 x2 --)
3drop	(x1 x2 x3 --)
nip	(x1 x2 -- x2)

< nip > removed the second item. Here is the stack notation for clear.

clear	(... --) that is, < clear > removes all items as can be seen above. So the elipse means " all items ".

The stack can also be rearranged, as follows:

```
0 > 1 2 3 4 .s 1 2 3 4
ok
4 > rot .s 1 3 4 2
ok
4 > -rot .s 1 2 3 4
ok
4 > swap .s 1 2 4 3
ok
4 > 2swap .s 4 3 1 2
ok
4 > clear  ok
0 >

rot          ( x1 x2 x3 -- x2 x3 x1 ) rotated top three items
-rot          ( x1 x2 x3 -- x3 x1 x2 ) the other way
swap         ( x1 x2 -- x2 x1 )
2swap         ( x1 x2 x3 x4 -- x3 x4 x1 x2 )
```

Note:
For a complete list of user interface words, see the IEEE 1275 Specification, the PCI Binding for words specific to PCI, and Designing PCI Cards and Drivers for Power Macintosh Computers for Apple defined words.

There are two more words that you will find useful when debugging. One is for comments and the other is to extend the dictionary with a new or redefined word.

Here is the display string word < ." >. When the interpeter sees this word, it inputs the following text string. When it sees < " > delimiter, it exits this mode and displays the string contained between the < ." > and the delimeter ". Here's an example.

```
0 > ." Hello world, I'm an example " Hello world, I'm an example  ok
0 >
```

What was entered was echoed. Here is the first word that did not take its operand from the stack. It took all characters until a < " > was seen. Here is the stack notation.

." ([text<">] --)

The [ ] pair says to take all text until a ". Can be confusing huh?

Here is the new or redefine word. It is called a **colon definition** and looks like this:

: + ." I don't do addition " ; Look at ." I don't do addition ". This is just a string. The < : > started the definition of a word called < + > that echoes the string and the < ; > ended the definition. Look at this example.

```
0 > 1 2 3 + .s 1 5
ok
2 > : + ( -- ) ." I don't do addition " ;  ok
2 > + I don't do addition  ok
2 > forget +  ok
2 > +  ok
1 > . 6  ok
0 >
```

Can you tell what happened?

1. 1 2 3 were entered, and then the contents of the stack were displayed.
2. The + operator was redefined to do nothing but type a string to the display and leave the contents of the stack as it was. That's the reason for the stack notation (--) which stands for " stack contents unaffected".
3. After the first addition the stack had a 5 on the top with a 1 below it. The new definition of < + > did nothing to the stack.
4. Here's a new word called < forget > which forgets the most recent definition and all words defined since that word.

Note:
The old definition of + was not replaced, just superceded. This implies that the dictionary places new definitions at the top and also searches for words starting at the top.

That's enough general Forth language and stack information to get you started with the device tree, which we turn to next.

Back to top

The Device Tree

The **device tree** is constructed by the Open Firmware probing process before the main Operating System is given control.

It contains nodes for each ASIC on the motherboard, nodes for expansion devices such as yours, and a set of utilities nodes. For example /packages and /aliases. The root of the tree is /. To get to the root node and list the entire tree, enter the following:

```
dev / ls
```

< dev > is a word which opens a node in the tree and the particular node in this example is the root or /. This is a bus node for the AR bus (Apple RISC). < ls > lists all nodes, if any, under the present node. Let's see how that works:

```
0 > dev / ls
86746496: /PowerPC,604@0
86747184:    /l2-cache@0,0
86749168: /chosen@0
86749472: /memory@0
86749800: /openprom@0
86749992: /AAPL,ROM@FFC00000
86750528: /options@0
86752280: /aliases@0
86752856: /packages@0
86752992:    /deblocker@0,0
86755040:    /disk-label@0,0
86756384:    /obp-tftp@0,0
86765664:    /mac-files@0,0
86767704:    /mac-parts@0,0
86769592:    /aix-boot@0,0
86770736:    /fat-files@0,0
86776320:    /iso-9660-files@0,0
86778696:    /xcoff-loader@0,0
86781192:    /terminal-emulator@0,0
86781344: /bandit@F2000000
86785936:    /gc@10
86787016:       /53c94@10000
86793296:          /sd@0,0
86796416:          /st@0,0
86799608:       /mace@11000
86803312:       /escc@13000
86803656:          /ch-a@13020
86805368:          /ch-b@13000
86807080:       /awacs@14000
86807312:       /swim3@15000
86811672:       /via-cuda@16000
86814632:          /adb@0,0
86814872:             /keyboard@0,0
86816744:             /mouse@1,0
86816920:          /pram@0,0
86817096:          /rtc@0,0
86818320:          /power-mgt@0,0
86818608:       /mesh@18000
86825624:          /sd@0,0
86828744:          /st@0,0
86832080:       /nvram@1D000
86839664:    /pci106b,1@B
86840136:    /ATY,XCLAIM@D
```

```
86876008:    /wayne.device@E
86878936:    /wayne.device@F
86832488: /bandit@F4000000
86882192:    /pci106b,1@B
86882664:    /pci1234,5678@D
86883440:    /TRUV,TARGA2000PCI@F
86837184: /hammerhead@F8000000
ok
0 >
```

Take a close look at the tree and then go into your Developer Notes for the 9500 Macintosh. Notice that this resembles the block diagram on the 9500. It is the block diagram and more. Note that there are two nodes called bandit. One is /bandit@F2000000 and the other is /bandit@F4000000. These are full path names. One can deduce that full path names can contain address information, such as @F40000000, when the name is not unique, such as bandit. Bandit is the PCI bridge chip that has the AR bus as a parent and the PCI bus as a child. The the device tree listing is from my 9500 and contains debugging nodes called /wayne.device@E and /wayne.device@F.

Since we are at the root node, let's look at the pwd, .properties, and words words. Enter the following.

```
0 > pwd / ok
0 > .properties
name                    device-tree
model                   Power Macintosh
compatible              AAPL,9500
MacRISC
AAPL,cpu-id             3900A69D
#address-cells          00000001
#size-cells             00000001
clock-frequency         02FAF080

ok
0 > words
dma-sync        dma-map-out     dma-map-in      dma-free
dma-alloc       map-out map-in              decode-unit
close           open
ok
0 >
```

< pwd > displays the full path name for the current node. In this case pwd displays the root node.

< .properties > listed all properties under this node (which is the AR bus). Note that although the < pwd > word listed the node name as/, its proper node name is device-tree. Look at the clock-frequency property, which is 0x02FAF080 (or 50MHz).

< words > listed the words (methods) implemented by this node. But what do these words do? Well, enter < see > and then the word you want to see, such as < open >.

```
0 > see open
: open
true
; ok
0 >
```

So < open > just places the logical value of < true > onto the top of the stack.

Now let's look at one last maneuver - traversing the tree to another node. For this example, we'll look at the node called wayne at the top location in the tree.

```
0 > dev /bandit@F2000000/wayne.device  ok
0 > pwd /bandit@F2000000/wayne.device@E ok
0 >
```

Note:
The node chosen was the first of two.

What if we had wanted the second node? Replacing the @E with @F would have chosen that node. Also note that the < pwd > word showed the name of the node. Typing long path names is subject to mistakes, so let's look at an alternative method. Enter the word < devalias > to see the following:

```
0 > devalias
vci0                /chaos@F0000000
pci1                /bandit@F2000000
pci2                /bandit@F4000000
fd                  /bandit/gc/swim3
kbd                 /bandit/gc/via-cuda/adb/keyboard
ttya                /bandit/gc/escc/ch-a
ttyb                /bandit/gc/escc/ch-b
enet                /bandit/gc/mace
scsi                /bandit/gc/53c94
scsi-int            /bandit/gc/mesh
ok
```

Look at the second entry called pci1.

Now enter:

```
0 > dev pci1/@e   ok
0 > pwd /bandit@F2000000/wayne.device@E ok
0 >
```

Cool huh? pci1 is an alias of /bandit@F2000000.

Note:
@e or @E (not case sensitive) is also a valid name for a node.

Back to top

Summary

There is much more yet to be described about using Open Firmware to build and debug your drivers and devices. The next Technote in this series, *Technote 1062 - Fundamentals of Open Firmware, Part II: The Device Tree,* which will be available soon, addresses details of working with the device tree.

Back to top

References

IEEE 1275-1994 Specification

PCI Local Bus Specification 2.0

PCI Bus Binding Specification 1.5

Designing PCI Cards and Drivers for Power Macintosh Computers

Technote 1044 - Understanding PCI Expansion ROM Choices for Mac OS8 , Part III in the Open Firmware Technote Series

Back to top

Downloadables

       Acrobat version of this Note (K).   Download

Back to top