Expanding Your Market with Open Firmware

Greg Hill
Director of Marketing
FirmWorks
gregh@firmworks.com

Agenda

- Why Have Open Firmware
- What is Open Firmware
- The Good News -- The Benefits of Open Firmware
- The Bad News -- The Costs of Open Firmware
- Open Firmware Development Resources
- Conclusion

Who's On First?

- In a recent quarter, which computer vendor had the largest market share?
  - Apple with about 12%
  - Compaq with about 12%
  - IBM with about 12%
  - All of the above

Who's On PCI with Open Firmware?

- Apple with the Power Macintosh™
- IBM with PR*P (coming soon)
- Do you want a piece of the action?

Power Macintosh is a trademark of Apple Computer Inc.

What Problems Does Open Firmware Solve?

- No Standards
  - Proprietary Solutions
  - Machine-dependent Interfaces
  - Inconsistent User Interfaces
  - Re-invention of the Wheel
- Ad Hoc Design
  - Cumbersome/inflexible OS Interfaces
  - Weak Naming Structure

What Problems Does Open Firmware Solve?

- No Open Systems Support
  - Single-vendor boot/diagnostic support
  - CPU dependencies
  - Weak or nonexistent auto-configuration mechanisms
- Constrained environment
  - Firmware environment can't depend on full machine capabilities
  - Meager debugging tool set
- Expensive to maintain and upgrade

The Open Firmware Response

- Unencumbered Public Specification (IEEE 1275-1994)
  - The interfaces are open and public - no fees, restrictions or "contamination" concerns
  - Companies may sell or license specific implementations
  - Buy off-the-shelf or build from the spec -- your choice
- Designed for the long term
  - Structured OS Interfaces
  - Explicit reporting of resource utilization
  - Unambiguous hierarchical naming structure
  - Architected extensibility for future growth

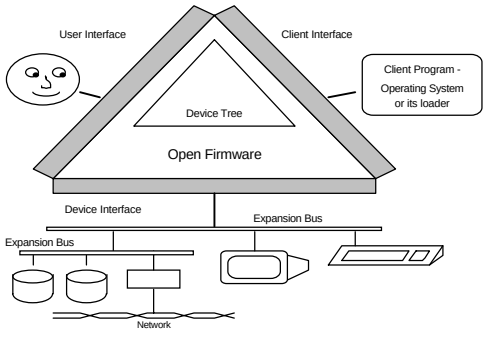
The Open Firmware Response

- Open Systems Orientation
  - CPU-independent, bus-independent design
  - Device name space supports arbitrary combinations of different buses
  - Architecture-neutral interfaces
  - Multi-vendor booting, testing
- Auto-configuration Support
  - "Plug and play" across different processors with ONE driver

The Open Firmware Response

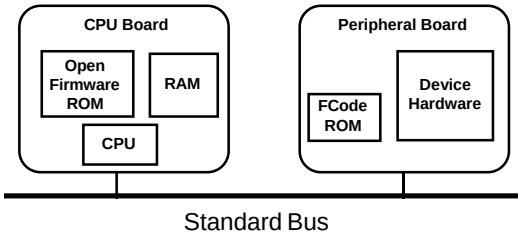
- Debugging Facilities
  - Standard Firmware-level Debug Interface
  - Built-in interactive programming language
  - Hardware, software, firmware, driver debugging tools
- Field Patches, Upgrades
  - Downloadable firmware extensions
  - Extensions and patches in non-volatile RAM

Open Firmware Interfaces



- Device Interface - plug-in “FCode” drivers
- User Interface - administration and debugging
- Client Interface - services for OS and loaders
- The 3 interfaces are separately optional

Device Interface (FCode)



- FCode interpreter/compiler runs on the main CPU
- FCode programs reside in PCI Expansion ROMs on peripheral cards (can be stored elsewhere for some buses)
- During system start-up, Open Firmware “probes” the bus and reads/interprets any FCode it finds
- Resulting driver used during start-up

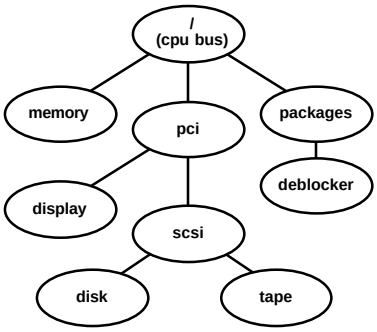
Device Interface (FCode)

- Interpreting/compiling an FCode program:
  - Creates a device node with descriptive properties
  - Creates device driver methods in RAM
  - Initializes and tests the device
- The same FCode driver works with any CPU
- FCode drivers don’t replace OS drivers
  - OS drivers are complicated; FCode drivers are simple
  - OS and loaders can use FCode drivers temporarily

Building the Device Tree

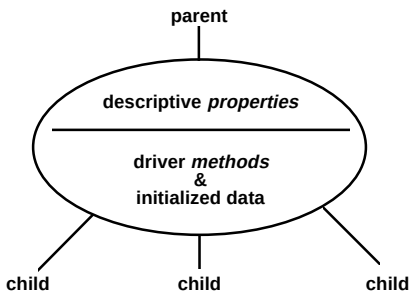
- Start with tree for built-in devices
  For each occupied slot:
  - Create an empty device node
  - Interpret FCode for that board
  - (if the device is a bridge), recurse
    - For each occupied slot:
      - Create an empty device node
      - Interpret FCode for that board
      - (if the device is a bridge), recurse
        - (etc.)

Device Tree



- The Device Tree maps the hardware's physical addressing
- The tree structure is the key to portability and extensibility
- Nodes with children are usually buses
- Nodes without children are usually individual devices
- Siblings are distinguished by name and by physical address

Device Node



- Properties - name,value pairs describing the device
- Methods - driver procedures for the device

Properties

- Name,value pairs describing device characteristics
- Some standard properties
  - "name" Human-readable device name used in paths
  - "reg" List of address ranges for registers
  - "interrupts" List of interrupt levels and/or vectors
- Some properties are class-dependent, e.g. "width"
- Other device-dependent properties can be created
- PCI "binding" defines some PCI-specific properties

Methods

- Forth procedures for driving the device
- Called by name - run-time binding
- Some standard methods:
  - "open" , "close" — Start/stop using device
  - " read" , "write" — Input/Output
  - "load" — Load a program from the device
- Bus nodes provide mapping and DMA methods their children can use

Client Program Interface

- Allows the operating system (or OS loader) to use Open Firmware services
- Device tree access (for OS auto-configuration)
  - Select a device node
  - Get and set property values
- FCode driver access
  - Console I/O
  - Disk, tape, network (for secondary booters)
- Memory allocation and mapping

User Interface

- Provides casual user access to:
  - Booting commands
  - Configuration variables
- Provides expert user access to:
  - Device tree browsing
  - Canned hardware diagnostics
  - Patch scripting for bug fixes/workarounds
  - Debugging tools

Configuration Variables

- Used to specify boot process options
  - Default boot device and default OS to boot
  - Default console device
  - Control level of diagnostics to run
  - Specify patches (if any) and control whether patches should be applied
- User can define new configuration variables
- Stored in the system's NVRAM

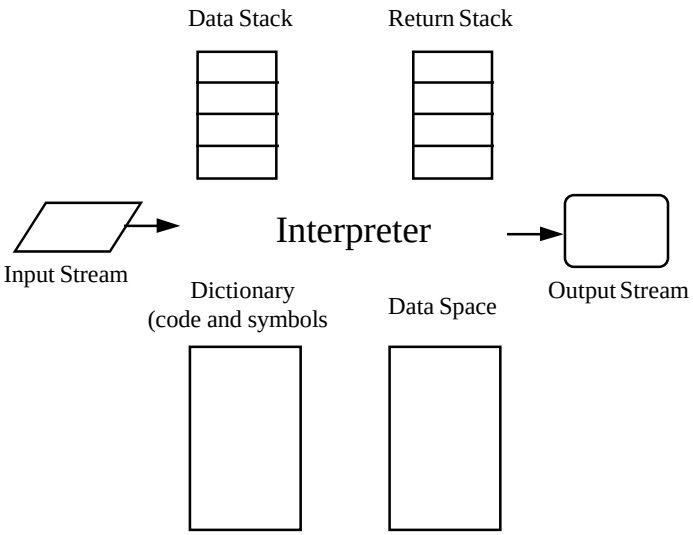
The “script”

- Used to store commands as though typed at User Interface
- Automatically run during startup if use-nvramrc? configuration variable is true
- Part of system’s NVRAM
- Stores custom extensions and/or bug workarounds (avoids emergency ROM upgrades!)

Underlying Technology -- Why Forth?!

- Interactive environment on a constrained system
- Obvious machine-independent binary format
- Built-in debugger
- Extensible User Interface "for free"
- Very easy to port to new machines
- The entire language is always available

Forth Virtual Machine



Forth Language

- Lexical: Blank-delimited “words”
- Syntax: Very little. The basic operation of the interpreter:
  - Read a token
  - Search the dictionary for the token
  - If found, execute the associated code
  - If not found, parse token as a number and push onto stack
  - If not a number, indicate error
- Compiling
  - “: <newname>” switches to “compile state” and begins defining <newname>
  - Code is incrementally compiled instead of executed
  - When compile state is ended with “;”, <newname> is entered in the dictionary
- Data Types:
  - Forth is basically untyped
  - Fundamental type: integer on the stack
  - Can represent a number, a character, an address, etc.

FCode is Encoded Forth Source Code

- Use the Source, Luke, for CPU Independence
- Binary-encoded to Save Space
- Byte-encoded to Eliminate Endianess
- Each byte code means "do something now"
- Normally, functions are executed immediately
- “Words” (i.e. functions) can be defined for later use
- Each function is very simple
- Typically uses 200-5K bytes of PCI Expansion ROM

FCode is the General Solution

- The list of properties can be extended arbitrarily
  - Properties identified by name, not by “magic number”
  - No central “registration service” is needed
  - Property values can encode arbitrary data
- Handles device hierarchies and complex configurations
  - Multiple devices on one card
  - Devices with hierarchical relationships
  - Bus bridges
- Complete programming language power
  - Initialize devices
  - Calculate property values
  - Report dynamic characteristics
  - Boot drivers
- Object-oriented, general-purpose, extensible framework
  - Not a “hit list” of individual features
  - New features don’t require interpreter changes or CPU firmware upgrades

FCode Interpreter

- Interpreter Loop:
  - Read byte code from the device ROM
  - Index into jump table to get function address
  - Call function
- Compiling:
  - Detect function that switches from interpreting to compiling
  - Read byte code from the device ROM
  - Index into jump table to get function address
  - Add function address to definition of new function
  - Detect function that completes definition and switches back to interpreting
  - New function is immediately available for use either by interpreter or compiler
- The set of predefined functions forms a general-purpose programming language (based on ANSI Forth)
- There are library functions for creating properties and other identification and booting requirements

System ROM Support Simplifies FCode Drivers

- "Support packages" provide common functions for use by FCode drivers
- Standard support packages include:
  - terminal emulator (bit-mapped frame buffer)
  - disk label (disk)
  - deblocker (tape, disk)
  - obp-tftp (Ethernet, FDDI)

What's in it for You?

- For Your Company
  - Opens New Markets with the Same Hardware
  - Product differentiation
  - Adds Productivity to the Development Lab
  - Powerful Framework for Manufacturing and Field Service Diagnostics

What's in it for the Firmware/Software Developer

- FCode versus BIOS drivers
- Top-down Design/Bottom-up Test Made Easy
- Forth / Assembly Language Debugger
- Open Firmware and Plug and Play
- Open Firmware and ARC
- Open Firmware and PCI

FCode Drivers
and
BIOS Drivers

- FCode drivers *don't* replace x86 BIOS drivers (They could in theory; in practice, it won't happen)
- FCode driver and x86 driver co-reside in the same PCI Expansion ROM
- FCode driver handles non-x86 platforms

Open Firmware
and
Plug and Play

- Plug and Play - a collection of bus-dependent, x86-centric point solutions
- Open Firmware - a unifying framework for different auto-configuration technologies
- Open Firmware can use, augment and enhance ISA Plug and Play
- Open Firmware is extensible to future systems and complicated bus topologies

Open Firmware
and
ARC

- ARC
  - Scope is limited to “client interface”
  - No facilities for CPU-independent plug-in drivers
  - Specification appears to be encumbered
- ARC “veneer” can be built above Open Firmware services (FirmWorks has created one)
- Open Firmware specification is unencumbered
- Open Firmware is a complete solution

Open Firmware
and
PCI

- PCI Open Firmware “Binding” Spec
  - Builds upon IEEE “core” standard
  - Defines address representations, property names, ROM Image format for FCode, handling of cards without FCode
- x86 driver and FCode driver share Expansion ROM
- Generic descriptive “properties” created from Configuration Space header
- FCode driver can create additional properties
- FCode contains diagnostics and non-x86 boot code

What's in it for the Hardware Developer

- Open Firmware as a Bring-up Tool
  - User Interface permits rapid experimentation ("begin 4000 c@ drop key? until" = 'scope loop)
  - Demands only CPU, memory and UART be functional to get started
  - Register display and modification commands
  - Breakpoints

What's in it . . .

- For the Casual User
  - Auto-configuration
  - Easy customization
  - Text-based or Graphical Interface
- For the Expert User/Service Person
  - Consistent Interface Across Systems
  - Patch High-level Language with High-level Language

Casual User Interface (Optional)

- Booting
  - boot
  - boot disk
  - boot /pci/scsi/disk@3,0:1
- Setting configuration options
  - setenv input-device keyboard
  - setenv selftest-#megs 4

Expert User Interface (Optional)

- Device tree browsing
- Canned hardware diagnostics
- Complete Forth language interpreter
- Write custom macros for hardware debugging
- Store Forth scripts in non-volatile RAM
  - Custom extensions
  - Bug work-arounds (avoid emergency upgrades!)

More Expert User Interface

- FCode Debugger
  - Forth Decompiler
  - Forth source-level debugger
- Assembly language debugger for OS software
  - Symbolic disassembler
  - Single-stepping and breakpoints
  - Conditional macros

What's it Going to Cost You?

- PCI non-boot devices
  - Maybe nothing
- PCI boot devices
  - FCode Spoken Here
  - Only One FCode Driver Required
  - System ROM Support Minimizes the Job
- FCode Doesn't Replace the OS Driver

What Does it Take?

- To comply with the PCI Open Firmware Spec
  - A plug-in card needs an FCode driver in Expansion ROM
    Typical size: 200 - 5000 bytes, depending of device type
  - A CPU board needs an FCode interpreter in main ROM
    Typical size: 64K - 256K bytes, depending on the number of optional features (debugging tools, etc.) included

Simple FCode Program

```
fcode-version2
“ MYCO,tty” name
“ MYCO,123456-01” model
my-address my-space 8 reg
“ serial” device_type
6 encode-int “ interrupts” property
<method definitions go here>
end0
```

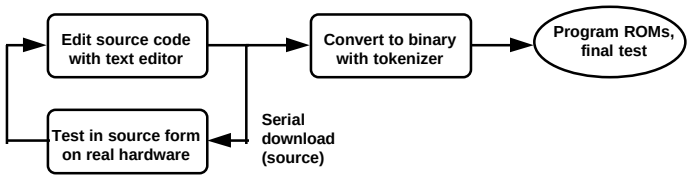
Simple FCode Methods

```
0 instance value chipaddr
: open  ( -- flag )
   my-address  my-space  8  “  map-in”  $call-parent   to
   chipaddr     true
;
: close  ( -- )   chipaddr  8  “ map-out” $call-parent   ;
: read  ( addr len -- actual )   drop  chipaddr   rb@  swap  c!   1
;
: write  ( addr len -- actual )
  tuck   bounds  ?do   i  c@   chipaddr  rb!   loop
;
```

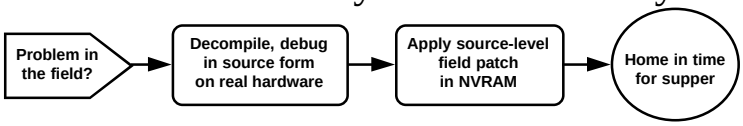

Developing FCode Drivers - Tools

- Open Firmware debugger
  - Source-level debugger - interactive execution, incremental compiler/decompiler, source-level tracing
  - Some Open Firmware ROMs have it built-in
  - Versions that run under an OS are available
- Tokenizer
  - Converts Forth source to FCode binary format
  - Inexpensive
- Detokenizer
  - Converts FCode binary programs back to source form
- Cross Platform Developer’s Kit
  - Provides development environment before hosts are generally available

Developing FCode Drivers - Process



- Develop and debug at source level
- Tokenize to binary for final delivery



ρ **Decompile, override, and patch for field support**

FCode Development Resources

- IEEE 1275-1994 Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices
- PCI Open Firmware “Binding” Spec.
- "Writing FCode Drivers for PCI"
- "Open Firmware Command Reference"
- "Open Firmware Quick Reference Card"
- Cross-Platform Developer’s Kit
- Class: “Writing FCode Drivers” (includes Forth language)
- Off-the-shelf drivers
- Third party services - drivers, training, porting, compliance testing (e.g. FirmWorks)

Standard Documents

- Core document:
  - Standard 1275-1994: IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements an Practices. US$87. Order Number SH17327.
  - IEEE Customer Service
    From the US and Canada: 1-800-678-4333
    From elsewhere: +1-908-981-1393
    Fax: +1-908-981-9667

Standard Documents

- Binding documents:
  - ~ 10 pages each
  - Define specifics for particular CPUs, buses
  - Some published by IEEE (1275.1, ...)
  - Others published by consortia (e.g. PCI SIG)
  - PCI, SPARC, 680x0, VME, Futurebus+, ...
  - Latest PowerPC and PCI bus bindings available via anonymous ftp from playground.sun.com in the directory /pub/p1275/bindings/postscript

Summary

- The Last Piece of the Open Systems Puzzle
- Powerful, Extensible Technology
- Speeds Development
- Increases User Friendliness

For More Information

- Contact:

Greg Hill
FirmWorks
480 San Antonio Road, Suite 230
Mountain View, CA 94040-1218

Phone: 415-917-6985 / 415-917-0100
FAX: 415-917-6990
Email: gregh@firmworks.com / info@firmworks.com