FirmWorks's Open Firmware Products

Summary

FirmWorks's Open Firmware products are implementations, for various processors, hardware architectures, buses, operating systems and devices, of software conforming to IEEE Standard 1275-1994, *Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices*. FirmWorks licenses both source code and object code versions. The source code is used by system manufacturers to develop firmware specific to their individual systems. The object code is recorded on read-only memory (ROM) chips and installed in the actual computer systems shipped by the manufacturers.

IEEE Standard 1275 is the first non-proprietary open standard for boot firmware that is usable on different processors and buses. Firmware complying with the standard (known as "Open Firmware") includes a processor-independent *device interface* that allows an add-in peripheral card to identify itself and to supply a single boot driver that can be used, unchanged, on any CPU. Open Firmware also includes a *user interface* with powerful scripting and debugging capabilities and a *client program interface* that allows operating systems and their loaders to make use of Open Firmware services to assist in the configuration and initialization process.

Open Firmware provides a device tree, a tree-structured description mechanism for system hardware configurations. The device tree supports multiple interconnected system buses, thus serving as a framework for "plug and play"-type autoconfiguration across different buses. Open Firmware is based on technology that has been in use for more than six years on over two million Sun Microsystems, Inc. computer systems around the world.

Following is a more detailed description of the general functionality of FirmWorks's products.

Description

Open Firmware is a portable boot firmware system. Boot firmware is the ROM-based software that controls a computer from the time that it is turned on until the primary operating system has taken control of the machine. The main function of boot firmware is to initialize the hardware and then to "boot" (load and execute) the primary operating system. Secondary functions include testing the hardware, managing hardware configuration information, and providing tools for debugging in case of faulty hardware or software.

Open Firmware is portable in the sense that its design is not tied to any particular processor family nor to any particular expansion bus. Open Firmware was specifically designed to support a variety of different processor Instruction Set Architectures (ISAs) and different buses. Open Firmware is already in use on over a million machines, and is supported by several system vendors. A number of bus standards, including PCI, Futurebus+, VME-D, and SBus, include provisions for Open Firmware card identification and booting. The PowerPC Platform ("PPCP" or "CHRP") and PowerPC Reference Platform ("PR*P") specifications require the use of Open Firmware.

FirmWORKS
480 San Antonio Road, Suite 115
Mountain View, California 94040-1218
U.S.A.

Telephone: +1 (650) 917-0100
FAX: +1 (650) 917-6990
Internet: info@firmworks.com

Firmware standardization can reduce system costs by eliminating “reinvention of wheels,” providing “off the shelf” sources for firmware, eliminating unnecessary relearning of different firmware systems, reducing the effort of porting operating systems to different machines, and providing a consistent and powerful base set of hardware and software debugging tools.

Open Firmware addresses numerous firmware problems. Its design was undertaken as a long-term effort to “do it right,” rather than viewing firmware as a “necessary evil” that should be done quickly and forgotten as soon as possible.

Notable Open Firmware Features

Plug-In Drivers

One key Open Firmware feature is support for self-identifying devices. Consider a computer with an “open” expansion bus, such as VMEbus or SBus. An independent board vendor (*i.e.* not a system manufacturer) of a card to plug into that bus would like for the system to recognize and be able to use that card. In the operating system environment, that may be easy - the board vendor can supply a driver on a diskette, and that driver may be loaded onto a hard disk or installed into the operating system.

In the firmware environment, acquiring drivers is more difficult; the firmware has to operate before the system is ready to read the disk. It is better to have the board driver in a ROM somewhere. Since it is difficult to merge third-party drivers into existing system ROMs, it is better to store such a driver in a ROM on the card for the “plug-in” device to which it applies. This approach has been taken before, but in most existing firmware systems, the driver is stored as ISA-dependent machine language binary code, and thus only works on computer systems compatible with a particular architecture.

Open Firmware uses the “plug-in driver” technique, but instead of storing those drivers in machine language, Open Firmware encodes the drivers in a machine-independent language called “FCode”. FCode is a byte-coded “intermediate language” for the Forth programming language. Forth, an industry-standard interactive programming language, is based on a stack-oriented “virtual machine” that may be easily and efficiently implemented on any computer. FCode drivers are “incrementally compiled” into system RAM for later execution. The same FCode driver can be used on systems with different processor types. Thus, for example, a particular Futurebus+ add-in card could be a boot device for a 680x0-based Futurebus+ system or an x86-based Futurebus+ system with no firmware changes.

In addition to its use for firmware device drivers, FCode also provides a descriptive capability. Plug-in device cards use it to report their characteristics to the firmware and system software. Such characteristics may include the device name, model, revision level, device type, register locations, interrupt levels, supported features, and any other identification information that makes sense for the particular device. System software may use this information to automatically configure itself for correct operation with particular devices. This information is stored in a processor- and architecture-independent format that may easily be retrieved decoded. Furthermore, this specification is open and extensible, and allows any arbitrary device information to be recorded, providing protection against obsolescence of the interface.

Interactive Debuggers

Open Firmware has an interactive Forth language interpreter that uses the same run-time system that executes FCode drivers. The Forth interpreter provides a set of programmable debugging features to allow developers, users, and service personnel to isolate system problems in the event of a failure.

Open Firmware can debug hardware, operating system software, plug-in drivers, and even the firmware itself. The emphasis is on interactive tools for exploring problems, rather than "canned" diagnostics (although Open Firmware includes provisions for "canned" diagnostics as well). With today's short product cycles, a new design may spend as much time in the lab as in actual production. Open Firmware is an excellent bringup tool, and has been shown to shorten the time it takes to get a product to market.

Flexible Naming

Open Firmware was designed for adaptability. Its notation and structure for naming particular devices is based on a hierarchical "device tree" that mimics the bus configuration and physical addressing of the machine on which it is being used. This structure applies equally well to simple single-bus desktop machines and to "back room" servers with multiple processors and complicated hierarchies of interconnected buses. The "name space" for individual device names was designed so that no central authority is needed for "allocating" names - companies can design their products without appealing to a "master name arbiter".

The Open Firmware command language is open-ended. In addition to the standard commands that are present on all implementations, an arbitrary number of new commands may be added at any time, even by the user. Such additional commands may provide access to system-specific features, or may simply be customizations for the needs and tastes of individual users.

Maintainability

Field ROM upgrades are expensive. Open Firmware provides a "self-patching" facility that allows many types of firmware bugs to be fixed without changing the system ROM. The same facility allows additional firmware capabilities to be added to systems in the field, without changing the ROMs.

Configuration Maintenance

As mentioned, plug-in devices describe their own characteristics with FCode. Such descriptions are stored in the device tree. Each device tree node represents a particular device, and the description of that device is stored in its device node. Buses are considered to be devices in this sense, and are represented by "interior" nodes in the device tree. The "children" of a bus node represent the devices attached to that bus. Permanently-installed "built-in" devices also have device tree nodes with associated descriptions. The set of descriptive information about a particular device is open-ended, so new types of devices and new characteristics are handled easily.

An operating system may use the device tree, with its device descriptions, to configure itself, locate particular devices, attach device drivers, etc. This supports the growing requirement for "plug and play" installation of new devices.

Another configuration issue is storage and maintenance of user choices, such as the preferred boot device and the amount of memory to test. Open Firmware has a facility for keeping such user choices in non-volatile memory, such as battery-backed RAM, electrically erasable PROM, or FLASH memory. Open Firmware configuration management uses self-describing human-readable parameter names and values. The human readable values are encoded for efficient storage in the non-volatile memory device, where space is often at a premium. All access to these parameters is by name; new parameters may be added and old ones deleted at will, allowing for easy evolution of product families.

Client Program (Operating System) Interface

After an operating system has been loaded, while the OS is configuring and initializing itself, it may need to use Open Firmware services. The Open Firmware client program interface allows the OS to examine the device tree, temporarily use Open Firmware device drivers, display progress messages on the console device, allocate memory, and utilize other Open Firmware services. Usually, after the OS is fully initialized, it assumes responsibility for most of these tasks, and Open Firmware is no longer needed until the machine is rebooted.

The Open Firmware services provided through the client program interface are also frequently used by secondary boot programs. In some cases, Open Firmware can load the operating system directly. In other systems, Open Firmware loads and executes an “OS loader” program, which then loads the operating system, perhaps from some special file system structure. Such OS loaders usually need drivers for the devices they use, and the Open Firmware client program interface allows them to use the Open Firmware device drivers.

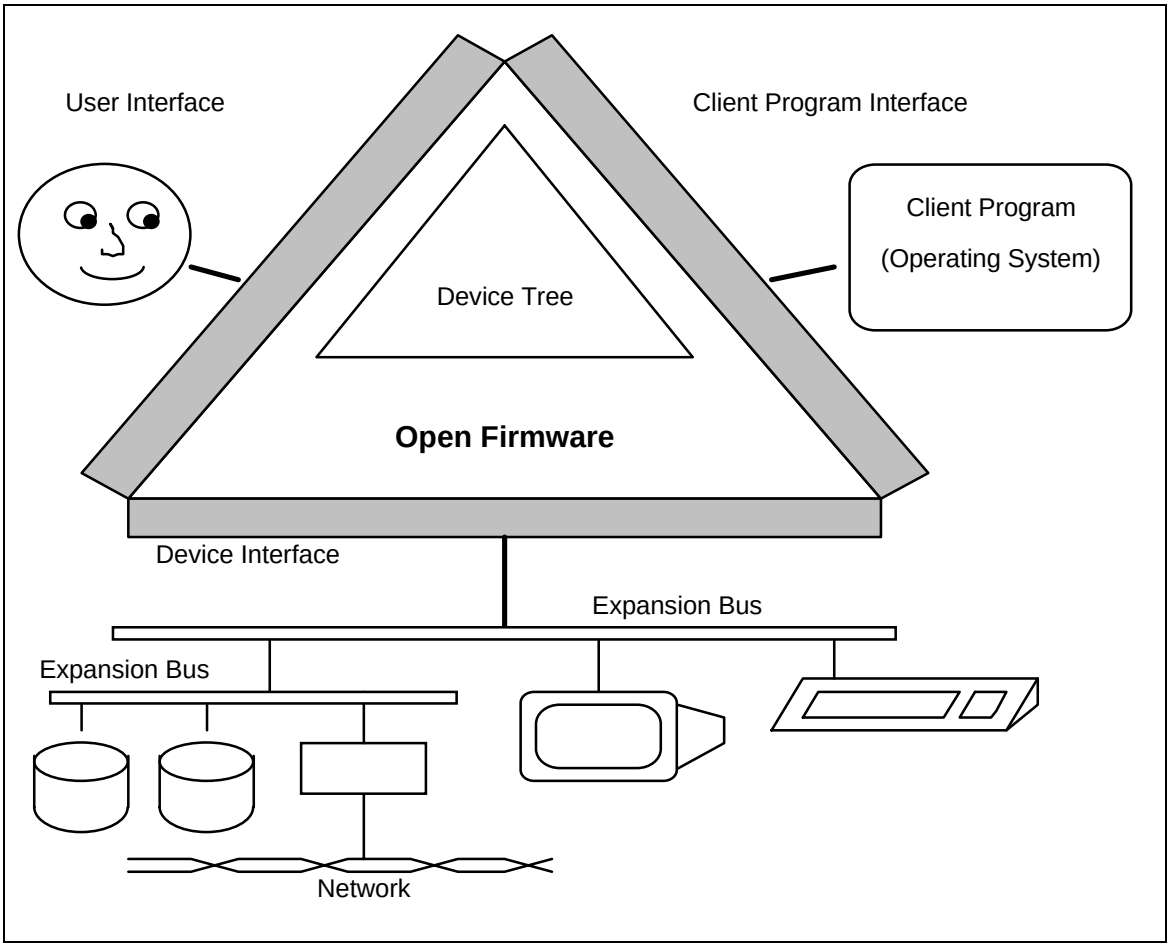
Open Firmware device drivers are rarely used by the primary operating system, except temporarily during OS initialization. In principle, it would be possible for an OS to continue using Open Firmware drivers, but that is rarely done because Open Firmware drivers are generally optimized for simplicity and small size, whereas OS drivers are often optimized for performance and complete functionality.

Open Firmware and Embedded Systems

Open Firmware was originally designed for “workstation-class” machines. (In particular, it was developed at Sun Microsystems, and is in the ROMs of all current Sun machines). Open Firmware is probably excessive for 16-bit machines, whose resource constraints make it both too big and unnecessary. (Such machines are often not very expandable). However, many embedded systems use powerful 32-bit processors, often built around industry standard buses and running off-the-shelf real-time operating systems. Such systems share many characteristics with general purpose workstations, and Open Firmware fits those environments quite well.

Embedded systems often require the integration of various hardware devices from different companies, as well as custom hardware. Open Firmware's powerful hardware debugging capabilities, based on the interactive Forth programming language, greatly assist in the task of integrating the various hardware devices. Dealing with a new device usually requires a fair amount of “exploration” (to find out how the device really behaves, as opposed to what the manual says), and the Forth interactive environment is ideal for such tasks.

Basic Structure



System Core

The core of FirmWorks's Open Firmware implementation is an ANS Forth-compliant kernel. This kernel provides the set of language primitives and operators used to implement the drivers and interfaces of the Open Firmware system. Compatibility with the ANS Forth Standard ensures source code portability and stability for future development. The FirmWorks kernel is a small, efficient Forth engine implemented in native machine code for speed and size. The kernel contains an integrated machine language assembler/disassembler and a rich set of extensions for development and debugging at the hardware level, including breakpoints, stepping, tracing, disassembly, and memory and I/O operations.

Device Interface

FCode drivers are not written in a vacuum: they may take advantage of Open Firmware operations supplied by the system firmware. The device interface specifies the full set of FCode primitives guaranteed to be available to an FCode driver. This set effectively constitutes an Application Binary Interface (ABI) for FCode which is required to be consistent and dependable across platforms and processors, providing a powerful framework for writing machine-independent drivers. The device interface also specifies probe and configuration practices on a bus-by-bus basis, affording processor independence to third-party developers.

Client Program Interface

As described previously, the client program interface specifies the services available to operating systems or other software loaded and invoked by the firmware. The client program interface also specifies the calling convention required by any processor family, providing client programs a platform- and operating system-neutral interface to firmware services. For most processor families, the client program interface is implemented as a subroutine call with a simple argument-passing convention.

User Interface

Nearly all the kernel primitives are accessible interactively. The user interface provides access to the system for booting and debugging, and includes features such as command-line editing, ANSI terminal emulation, and system security controls.

Size

A full-featured Open Firmware implementation, including debuggers, network protocols, selftest diagnostics, drivers for on-board devices, keyboard maps, graphics device support libraries, fonts, and on-line help, usually requires between 128K and 256K bytes of ROM space. In some system environments, unnecessary features may be omitted, making it possible to fit Open Firmware in a single 128K byte ROM.