REFERENCES


The P1275 “Open Boot” Standard for Boot Firmware

Mitch Bradley
Bradley Forthware
Mountain View, Calif.

Mitch Bradley is the principal architect and implementor of the Open Boot firmware. He has worked for Sun Microsystems, Inc. since 1982, and has been involved in hardware design, operating systems, and firmware. Prior to Sun, Bradley worked at ROLM as an analog circuit designer, at HP Laboratories as a programmer, and at Bell Laboratories as a circuit designer and programmer. Bradley has degrees from Vanderbilt and Stanford Universities, and studied for a year at Cambridge University. He is a member of the ANS Forth Technical Committee, and owns Bradley Forthware.
Open Boot 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 job of boot firmware is to initialize the hardware and then to "boot" (load and execute) the primary operating system. Secondary jobs include testing the hardware, managing hardware configuration information, and providing tools for hardware and software debugging.

Open Boot is portable in the sense that its design is not tied to any particular processor family nor to any particular expansion bus. Open Boot was specifically designed to support a variety of different processor Instruction Set Architectures (ISAs) and different buses.

The IEEE P1275 Open Boot working group is developing an IEEE standard for boot firmware based on Open Boot. Open Boot is already in use on over half a million machines, and is supported by several system vendors. A number of bus standards, including Futurebus+, VME-D, and SBus, include provisions for Open Boot card identification and booting.

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 Boot squarely 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. Some Open Boot features are:

**PLUG-IN DRIVERS**

One key Open Boot 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 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 from a particular vendor.

Open Boot uses the "plug-in driver" technique, but instead of storing those drivers in machine language, Open Boot encodes the drivers in a machine-independent language called "FCode". FCode is a byte-coded "intermediate language" for the Forth programming language. Forth 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 386-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, 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.

**INTERACTIVE DEBUGGERS**

Open Boot 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 Boot 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 Boot does include provisions for "canned" diagnostics too). With today's short product cycles, a new design may spend as much time in the lab as in actual production. Open Boot is an excellent bringup tool, and can shorten the time it takes to get a product to market.

**FLEXIBLE NAMING**

Open Boot 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 Boot 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...
Open Boot 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 job of boot firmware is to initialize the hardware and then to "boot" (load and execute) the primary operating system. Secondary jobs include testing the hardware, managing hardware configuration information, and providing tools for hardware and software debugging.

Open Boot is portable in the sense that its design is not tied to any particular processor family nor to any particular expansion bus. Open Boot was specifically designed to support a variety of different processor Instruction Set Architectures (ISAs) and different buses.

The IEEE P1275 Open Boot working group is developing an IEEE standard for boot firmware based on Open Boot. Open Boot is already in use on over half a million machines, and is supported by several system vendors. A number of bus standards, including Futurebus+, VME-D, and SBus, include provisions for Open Boot card identification and booting.

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 Boot squarely 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. Some Open Boot features are:

PLUG-IN DRIVERS

One key Open Boot 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 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 from a particular vendor.

Open Boot uses the "plug-in driver" technique, but instead of storing those drivers in machine language, Open Boot encodes the drivers in a machine-independent language called "FCode". FCode is a byte-coded "intermediate language" for the Forth programming language. FCode 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 386-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, 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.

INTERACTIVE DEBUGGERS

Open Boot 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 Boot 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 Boot does include provisions for "canned" diagnostics too). With today's short product cycles, a new design may spend as much time in the lab as in actual production. Open Boot is an excellent bringup tool, and can shorten the time it takes to get a product to market.

FLEXIBLE NAMING

Open Boot 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 Boot 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 Boot 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 Boot has a facility for keeping such user choices in non-volatile memory, such as battery-backed RAM, electrically erasable PROM, or FLASH memory. Open Boot configuration management uses self-describing human-readable parameter names and values. The human readable values are encoded into a “private” internal format 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.

OPERATING SYSTEM INTERFACE

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

The Open Boot services provided through the OS interface are also frequently used by secondary boot programs. In some cases, Open Boot can load the operating system directly.

In other systems, Open Boot loads and executes a “secondary boot” program, which then loads the operating system, perhaps from some special file system structure. Such secondary booters usually need drivers for the devices they use, and the Open Boot OS interface allows them to use the Open Boot device drivers.

Open Boot 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 Boot drivers, but that is rarely done because Open Boot drivers are generally optimized for simplicity and small size, whereas OS drivers are often optimized for performance and complete functionality.

OPEN BOOT AND EMBEDDED SYSTEMS

Open Boot 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 Boot is probably overkill for 16-bit machines, whose resource constraints make it both too big and also 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 Boot fits those environments quite well. Force Computers, a leading supplier of board level products used for embedded systems, has announced its intention to use Open Boot as the standard firmware for all its future products, spanning several processor families and several industry standard buses.

Embedded systems often require the integration of various hardware devices from different companies, as well as custom hardware. Open Boot’s powerful hardware debugging capabilities, based on the interactive Forth programming language, greatly assist in the task of integrating that 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 without peer for such tasks.

SIZE

A full-featured Open Boot 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 Boot in a single 128K byte ROM.

AVAILABILITY

The Pl275 draft specification is readily available. The ideas and interfaces embodied therein are unencumbered, and may be freely implemented and used. However, developing a complete Open Boot implementation “from scratch” would be a significant undertaking, as the system is rather extensive. Sun Microsystems is licensing the source code for its Open Boot implementation. That source code is structured for ease of porting - the portions of the
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 Boot 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 Boot has a facility for keeping such user choices in non-volatile memory, such as battery-backed RAM, electrically erasable PROM, or FLASH memory. Open Boot configuration management uses self-describing human-readable parameter names and values. The human readable values are encoded into a "private" internal format 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.

OPERATING SYSTEM INTERFACE

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

The Open Boot services provided through the OS interface are also frequently used by secondary boot programs. In some cases, Open Boot can load the operating system directly.

In other systems, Open Boot loads and executes a "secondary boot" program, which then loads the operating system, perhaps from some special file system structure. Such secondary booters usually need drivers for the devices they use, and the Open Boot OS interface allows them to use the Open Boot device drivers.

Open Boot 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 Boot drivers, but that is rarely done because Open Boot drivers are generally optimized for simplicity and small size, whereas OS drivers are often optimized for performance and complete functionality.

OPEN BOOT AND EMBEDDED SYSTEMS

Open Boot 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 Boot is probably overkill for 16-bit machines, whose resource constraints make it both too big and also 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 Boot fits those environments quite well. Force Computers, a leading supplier of board level products used for embedded systems, has announced its intention to use Open Boot as the standard firmware for all its future products, spanning several processor families and several industry standard buses.

Embedded systems often require the integration of various hardware devices from different companies, as well as custom hardware. Open Boot's powerful hardware debugging capabilities, based on the interactive Forth programming language, greatly assist in the task of integrating that 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 without peer for such tasks.

SIZE

A full-featured Open Boot 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 Boot in a single 128K byte ROM.

Availability

The P1275 draft specification is readily available. The ideas and interfaces embodied therein are unencumbered, and may be freely implemented and used. However, developing a complete Open Boot implementation "from scratch" would be a significant undertaking, as the system is rather extensive. Sun Microsystems is licensing the source code for its Open Boot implementation. That source code is structured for ease of porting - the portions of the
code that are SPARC-dependent and Sun-hardware-dependent are well-isolated, and every effort has been made to minimize the extent of such dependencies.

FOR MORE INFORMATION

The latest version of the P1275 boot firmware draft specification is available by anonymous FTP from the Internet host ftp.apple.com, in the directory pub/standards/p1275. You may request a paper copy by writing or sending email to the P1275 working group secretary:

Mike Williams
Interactive Systems Corp.
26635 West Agoura Rd., Suite 200
Calabasas, CA 91320
(818) 880-1200x2212
Email: gwhia@ism.isc.com

After the working group has finished and the document becomes an official IEEE standard, it will be distributed through the usual IEEE channels.

For more information about the P1275 working group, contact the working group chairman:

Mitch Bradley
2732 Katrina Way
Mountain View, CA 94040
Voice: (415) 961-1302
Fax: (415) 962-0927
Email: Mitch.Bradley@Eng.Sun.COM

Trademarks:

Open Boot is a trademark of Sun Microsystems, Inc.
386 is a registered trademark of Intel Corp.

Tasking in Ada

Richard Riehle
AdaWorks
Palo Alto, Calif.

Richard Riehle is a software engineer with more than 20 years of experience in software development and project management. He teaches at the college level and has, since 1987, provided training, consulting, and programming resources for such clients as the Department of Defense, international aerospace companies, and software developers.

Doug Bryan
Computer Systems Laboratory
Stanford, Calif.

Doug Bryan has been working with Ada since 1982, when he served as reviewer for the initial MIL/ANSI standard. For the past eight years, he has been doing research at Stanford University on advanced programming-support environments and concurrent Ada programs. He has taught object-oriented design with Ada at Foothill College and Stanford, published more than fifteen technical papers, conducted as many Ada-related tutorials, and co-authored with Geoff Mendal the two-volume text Exploring Ada.