

# **IEEE Draft Std P1275.1/D14a Standard for Boot (Initialization Configuration) Firmware Supplement for IEEE 1754 ISA**

18 August, 1994

*DRAFT EDITION in response to SPONSOR BALLOTING*

Sponsored by the IEEE Bus Architecture Standards Committee

Prepared by the P1275 Working Group

This is an unapproved draft of a proposed IEEE Standard, subject to change. Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities. If this document is to be submitted to ISO or IEC, notification shall be given to the IEEE Copyright Administrator. Permission is also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position. Other entities seeking permission to reproduce this document for standardization or other activities, or to reproduce portions of this document for these or other uses must contact the IEEE Standards Department for the appropriate license. Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department  
Copyright and Permissions  
445 Hoes Lane, PO Box 1331  
Piscataway, NJ 08855-1331 USA

Copyright © 1992-94 by the Institute of Electrical and Electronics Engineers, Inc.  
345 East 47th Street, New York, NY 10017 USA  
All rights reserved.

**1 Introduction**

2 (This foreword is not a part of IEEE Draft Std P1275.1/D14a, Standard for Boot (Initialization Configuration) Firmware,  
3 Supplement for IEEE 1754 ISA)

4 Firmware is the ROM-based software that controls a computer between the time it is turned on and the time the  
5 primary operating system takes control of the machine. Firmware's responsibilities include testing and initializing  
6 the hardware, determining the hardware configuration, loading (or booting) the operating system, and providing  
7 interactive debugging facilities in case of faulty hardware or software.

8 Open Firmware is the firmware architecture defined by IEEE Std 1275-1994, *Standard for Boot (Initialization*  
9 *Configuration) Firmware, Core Requirements and Practices*. That standard is bus-independent, instruction-set-  
10 independent, and system-independent.

11 The core requirements and practices specified by IEEE Std 1275-1994 must be supplemented by system-specific  
12 requirements to form a complete specification for the firmware for a particular system. This standard establishes  
13 such additional requirements pertaining to the instruction set architecture defined by IEEE Std 1754-1994, *Open*  
14 *Microprocessor Standard*.

**1 Working Group Members**

2 The following individuals were members of the Working Group at the time this document was produced:

|   |   |
|---|---|
| 3 William M. (Mitch) Bradley, Chairman, FirmWorks | David M. Kahn, Vice Chairman, Sun Microsystems Inc. |
| 4 John Rible, Draft Editor, FORCE COMPUTERS Inc.  | Luan D. Nguyen, Secretary, IBM Corporation          |
| 5 Ron Hochsprung, Apple Computer Inc.             | Thanos Mentzelopoulos, Intel, Inc.                  |
| 6 David L. Paktor, FORCE COMPUTERS Inc.           | Yongjae Rim, IBM Corporation                        |
| 7 Paul Thomas, Sun Microsystems Inc.              | Mike Tuciarone, FirmWorks                           |
| 8 Martin Walsh, Sun Microsystems Inc.             |   |
| 9   |   |

**10 Contributors**

11 The following individuals have contributed to this document:

|  |                                       |
|--|---------------------------------------|
| 12 Shawn Morrissey, FORCE COMPUTERS Inc. | Mike Williams, SHL                    |
| 13 Paul Fischer, FORCE COMPUTERS Inc.    | Ilan Rabinowitz, FORCE COMPUTERS Inc. |
| 14                                       |                                       |

**15 Sponsor Balloting Body**

16 The following individuals were members of the Sponsor Balloting Body:

17

---

## Table of Contents

|    |  |    |
|----|--|----|
| 1  |  |    |
| 2  | Introduction.....                        | ii |
| 3  | 1. Overview.....                         | 1  |
| 4  | 2. References.....                       | 1  |
| 5  | 3. Terms.....                            | 1  |
| 6  | 4. Data formats and representations..... | 1  |
| 7  | 5. Client interface requirements.....    | 1  |
| 8  | 5.1. Client program loading.....         | 2  |
| 9  | 5.2. Initial program state.....          | 2  |
| 10 | 5.3. Client interface handler.....       | 5  |
| 11 | 6. User interface extensions.....        | 5  |
| 12 | 6.1. Machine register access.....        | 7  |
| 13 | 6.2. Debugger extensions.....            | 9  |
| 14 | 6.3. Configuration variables.....        | 9  |
| 15 | 6.4. Restrictions.....                   | 9  |
| 16 |  |    |

## 1 1. Overview

2 This standard specifies the application of IEEE Std 1275-1994, *Standard for Boot (Initialization*  
3 *Configuration) Firmware, Core Requirements and Practices* to computer systems that use the instruction  
4 set architecture (ISA) defined by IEEE Std 1754-1994, *A 32-bit Microprocessor Architecture*, including  
5 instruction-set-specific requirements and practices for debugging, client program interface, and data  
6 formats. These requirements are imposed upon firmware compliant with IEEE Std 1275-1994 when such  
7 firmware is used on a computer system that uses the above ISA. The requirements are *not* imposed on the  
8 ISA itself.

## 9 2. References

10 This standard shall be used in conjunction with the following publications. When the following standards  
11 are superseded by an approved revision, the revision shall apply.

12 [1] IEEE Std 1275-1994, *Standard for Boot (Initialization Configuration) Firmware, Core Requirements*  
13 *and Practices*

14 [2] IEEE Std 1754-1994, *A 32-bit Microprocessor Architecture*

15 NOTE—Where the notation style differs between IEEE Std 1275-1994 and IEEE Std 1754-1994, the notation of this document  
16 follows that of IEEE Std 1275-1994.

## 17 3. Terms

18 This standard uses technical terms as they are defined in the documents cited in “References,” plus the following  
19 terms:

20 **3.1 core specification.** Synonym for IEEE Std 1275-1994, *Standard for Boot (Initialization Configuration)*  
21 *Firmware, Core Requirements and Practices*, i.e., the standard that specifies the system-independent and bus-  
22 independent requirements for Open Firmware.

23 **3.2 Open Firmware.** The firmware architecture defined by IEEE Std 1275-1994 and its applicable supplements or,  
24 when used as an adjective, a software component compliant with such an architecture.

## 25 4. Data formats and representations

26 The cell size shall be 32-bits. Number ranges for  $n$ ,  $u$ , and other cell-sized items, are consistent with 32-  
27 bit two’s-complement number representation.

28 The required alignment for items accessed with  $a$ -*addr* addresses shall be two-byte alignment (i.e. any  
29 even address is an acceptable  $a$ -*addr*). (This ISA requires four-byte alignment at the hardware level.  
30 However, the Forth implementation can hide this restriction; doing so can result in worthwhile reductions  
31 in ROM size in some cases.) An implementation may allow one-byte alignment of  $a$ -*addr* addresses, but  
32 shall not require alignment more strict than two-byte.

33 Each operation involving a  $q$ -*addr* address shall be performed with a single 32-bit access to the addressed  
34 location; similarly, each  $w$ -*addr* access shall be performed with a single 16-bit access. (This, in conjunction  
35 with the alignment requirements imposed by the instruction set architecture, implies four-byte alignment  
36 of  $q$ -*addrs* and two-byte alignment of  $w$ -*addrs*.)

## 37 5. Client interface requirements

38 An Open Firmware client interface implementation for an IEEE 1754-compliant processor shall behave as  
39 described below.

## 1 5.1. Client program loading

### 2 5.1.1. Default load address

3 The default load address is the virtual address 0x4000. At least 0x80000 bytes of memory shall be available at that  
4 address. It is strongly recommended that as much memory as is practical for the particular system be available  
5 there, thus allowing the loading of large client programs.

### 6 5.1.2. Client program header

7 An Open Firmware implementation shall recognize the sequence of eight quadlets described below as a valid client  
8 program header (as used by the **load** User Interface command in the core specification) if the `bf_magic` and  
9 `bf_format` quadlets contain the specified values. If either quadlet does not contain the specified value, the  
10 behavior of the Open Firmware **load** command is implementation-dependent. The offsets given below are from  
11 the beginning of the loaded image. Each of the quadlets described below is in big-endian byte order.

| Offset | Name                   | Contents  |
|--------|------------------------|---|
| 0      | <code>bf_magic</code>  | 0x0103.0107.  |
| 4      | <code>bf_text</code>   | Size of the client program's code.                    |
| 8      | <code>bf_data</code>   | Size of the client program's initialized data.        |
| 12     | <code>bf_bss</code>    | Size of the client program's uninitialized data area. |
| 16     | <code>bf_pad1</code>   | Undefined.  |
| 20     | <code>bf_origin</code> | Client program entry address.                         |
| 24     | <code>bf_pad2</code>   | Undefined.  |
| 28     | <code>bf_format</code> | 0xffff.ffff.  |

12 The program image immediately follows the header. After recognizing this header, **load** allocates and  
13 maps `bf_text` + `bf_data` + `bf_bss` bytes of memory beginning at the address given by  
14 `bf_origin`, moves the program image, of size `bf_text` + `bf_data`, to that address, and zeroes  
15 `bf_bss` bytes of memory beginning at `bf_origin` + `bf_text` + `bf_data`.

16 NOTE—Some existing client programs use other values in `bf_format`. An Open Firmware implementation may implement  
17 compatibility modes to handle such client programs. The details of such compatibility modes are outside the scope of this  
18 standard.

## 19 5.2. Initial program state

20 This section defines the “initial program state,” the execution environment that exists when the first machine  
21 instruction of a *client program* of the format specified above begins execution. Many aspects of the “initial  
22 program state” are established by **init-program**, which sets the *saved-program-state* so that subsequent  
23 execution of **go** will begin execution of the *client program* with the specified environment.

1 **5.2.1. Register values**

2 The CPU registers shall contain the following values:

| Register(s)     | Value  |
|-----------------|--|
| <b>%psr</b>     | S=1<br>ICC=0<br>EF=1 if floating point coprocessor present<br>EC=1 if second coprocessor present<br>EE=0<br>ET=1<br>PIL: implementation dependent value sufficiently low to allow the Open Firmware timer interrupt to occur<br>CWP: 0   |
| <b>%wim</b>     | 2  |
| <b>%tbr</b>     | See 5.2.4.   |
| <b>%y</b>       | 0  |
| <b>%i7</b>      | 0  |
| <b>%o6, %i6</b> | See 5.2.2.   |
| <b>%o1, %o2</b> | See 5.2.3.   |
| <b>%o3</b>      | Address of client interface handler. See 5.3.  |
| Other registers | Other global (%g0-%g7), local (%l0-%l7), in (%i0-%i5), and out (%o0, %o4, %o5, %o7) registers may be used for conveying information required by other client interfaces that are outside the scope of this standard. Any registers that are not used for such purposes shall contain zero. |

3 NOTE—The stipulation that unused other registers contain zero makes it possible for a firmware system to support multiple  
4 different client interfaces simultaneously. For example, a firmware system might present both an Open Firmware client  
5 interface and also a different interface for compatibility with some existing client program. A client program can determine  
6 whether or not a particular client interface is present by testing for a nonzero value in one of the registers that that client  
7 interface uses. The presence of the Open Firmware client interface is denoted by a nonzero value in %o3. An earlier firmware  
8 system that was an ancestor of Open Firmware uses %o0 to pass the (nonzero) address of its client interface data structure to  
9 the client program.

10 **5.2.2. Initial stack**

11 When the first machine instruction of a *client program* begins execution, there shall be a valid stack and the  
12 processor state shall have the following characteristics:

- 13 – %i6 shall contain zero.
- 14 – %o6 shall contain an 8-byte-aligned address referring to a location within an area of memory that the Open  
15 Firmware implementation has allocated for use as the client program's stack. That address shall be at least 96  
16 bytes below the top address of the stack memory area (providing space for saving the window registers of the  
17 current window), and at least 8000 bytes above the bottom address of stack memory area (providing room for  
18 stack growth).

19 An Open Firmware implementation shall handle window overflow traps by saving the “local” and “in” registers of  
20 the window below the trap window to the address specified by the %o6 register of that window.

21 An Open Firmware implementation shall handle window underflow traps by restoring the “local” and “in”  
22 registers of the window above the trap window from the address specified by the %o6 register of that window.

### 1 5.2.3. Client program arguments

2 Registers `%o1` and `%o2` may be used to pass to the client program an array of bytes of arbitrary content, with `%o1`  
3 containing the base address of the array and `%o2` the length. If no such array is passed, `%o1` and `%o2` shall contain  
4 zero.

5 NOTE—The Open Firmware standard makes no provision for specifying such an array or its contents. Therefore, in the  
6 absence of implementation-dependent extensions, a client program executed directly from an Open Firmware implementation  
7 will not be passed such an array. However, intermediate boot programs that simulate or propagate the Open Firmware client  
8 interface to the programs that they load can provide such an array for their clients.

9 NOTE—Boot command line arguments, typically consisting of the name of a file to be loaded by a secondary boot program  
10 followed by flags selecting various secondary boot and operating system options, are provided to client programs via the  
11 “`bootargs`” and “`bootpath`” properties of the “`/chosen`” node.

### 12 5.2.4. Trap table

13 In this subclause, *save-state-and-interact* means to save the CPU state to the extent possible, display (if possible) a  
14 message indicating that the trap occurred, and return control to the Open Firmware user interface if it is present.

15 `%tbr` shall refer to a trap table which handles traps as described in this subclause.

16 Open Firmware trap table entries shall not contain PC-relative branch offsets, in order that client programs can  
17 copy trap table entries without modifying them.

| Hardware Trap # | Name              | Behavior   |
|-----------------|-------------------|--|
| 5               | Window Overflow   | See 5.2.2.   |
| 6               | Window Underflow  | See 5.2.2.   |
| 0x11-0x1f       | Interrupt Vectors | An Open Firmware implementation may use certain interrupt vectors for internal functions, for example managing the timer used to implement the ALARM mechanism. Vectors that are not used for such purposes shall <i>save-state-and-interact</i> . |
| 0xff            | Software trap 127 | This trap vector is used for Open Firmware breakpoints. The Open Firmware handler for this trap vector shall <i>save-state-and-interact</i> .  |

18 A client program that installs its own trap table but wishes to continue using Open Firmware services should  
19 preserve the Open Firmware trap table entries for any traps that the client program does not explicitly need to  
20 handle. A good technique for doing this is to copy the contents of the Open Firmware trap table into the client  
21 program’s trap table, and then to replace only those entries to be serviced directly by the client program.

22 A client program shall not alter entries within the Open Firmware trap table.

23 When an Open Firmware command interpreter is entered after a client program has begun execution (e.g., via a  
24 user abort sequence, the `enter` client interface service, or a trap), the Open Firmware implementation shall  
25 restore the trap table register to point to its own trap table. If execution of the client program is subsequently  
26 resumed (e.g., with the `go` command), the Open Firmware implementation shall restore the trap table register to  
27 the value previously established by the client program.

### 28 5.2.5. MMU

29 The memory management unit (MMU), if present, shall be enabled.

30 NOTE—Many client programs require no knowledge of the details, or even the existence, of the MMU.



### 1 5.2.6. Virtual address space and memory allocation

2 When a client program begins execution, an Open Firmware implementation's use of any virtual address space  
3 outside of the ranges 0xffd0.0000-0xffef.ffff and 0xfe00.0000-0xfeff.ffff shall have ceased, except for the virtual  
4 address space and associated memory that is allocated for the client program's code and data, as specified in the  
5 client program header. Subsequently, the Open Firmware implementation shall not allocate virtual address space  
6 outside those ranges, except as needed for the execution of subsequent client programs or as explicitly requested by  
7 a client program.

8 An Open Firmware implementation should use the virtual address space range 0xffd0.0000-0xffef.ffff in preference  
9 to the range 0xfe00.0000-0xfeff.ffff, to the extent that is possible. Furthermore, allocation within the range  
10 0xfe00.0000-0xfeff.ffff should allocate higher addresses before lower addresses.

11 Client programs shall not depend on the ability to be loaded (as specified by its client program header) within  
12 either of those address ranges.

13 NOTE—By inspecting the value of “**available**” and “**existing**” properties in an MMU package, if such a package  
14 exists, a client program can determine precisely which ranges of virtual address space the firmware is using. For maximum  
15 portability, a client program ought not depend on the availability of any particular “hardcoded” virtual address.

### 16 5.2.7. Memory cache(s)

17 IEEE Std 1754 does not specify the cache organization, thus cache details depend on the system architecture. As a  
18 general guideline, it is recommended that the initial program state should have caches enabled.

## 19 5.3. Client interface handler

20 The *client interface handler* shall perform the following sequence of operations:

- 21 – invoke the Open Firmware client interface service specified by the argument array whose address was in %00  
22 when the code sequence was invoked.
- 23 – place the return value (indicating success or failure of the attempt to invoke the service) back in %00.
- 24 – return control to the client program at the address given by the value that was in %07 when the code sequence  
25 was invoked, plus eight.

26 The execution of the *client interface handler*, including the invocation of the client interface service, shall preserve  
27 the contents of all CPU registers other than %00, %01, %02, %03, %04, and %05.

28 NOTE—This implies that in order to invoke a client interface service, a client program first constructs a client interface  
29 argument array and puts its address in %00, puts the return address minus eight in %07, and then jumps to the client interface  
30 handler (typically this is done by using a JMPL instruction with %07 as the destination register).

31 The *client interface handler* may assume that it is invoked with a valid stack with enough space free to store 24  
32 quadlets (enough for the active register window and the global registers), and that operational Window Overflow  
33 and Window Underflow trap handlers are installed in the current trap table.

34 A *client program* calling a client interface service must ensure that when the client interface handler is invoked,  
35 the stack is valid and has enough space free to store 24 quadlets (enough for the active register window and the  
36 global registers), and that operational Window Overflow and Window Underflow trap handlers are installed in the  
37 current trap table.

38 NOTE—These conditions are met when a client program first begins execution under the control of an Open Firmware  
39 implementation. In order to call client interface services, the *client program* must not destroy the integrity of the stack.

## 40 6. User interface extensions

41 An Open Firmware user interface implementation for an IEEE 1754-compliant processor should  
42 implement the following additional commands.



## 1 **6.1. Machine register access**

2 The following commands represent registers within the *saved program state*. Executing the command  
3 returns the saved value of the corresponding register. The saved value can be changed by preceding the  
4 command with the new value and **to**. The actual registers are restored to the saved values when **go** is  
5 executed.

### 6 *Window Register Save Area*

7 Processors compliant with IEEE Std 1754 contain multiple register “windows.” The processor hardware  
8 physically implements a single set of the “global” registers (registers 0-7) and multiple (typically seven or  
9 eight) integer register “windows.” From the point of view of a typical program, the number of logical  
10 windows is limited only by the amount of memory available for the program stack. The hardware, through  
11 “Window Overflow” and “Window Underflow” traps, manages the hardware register windows  
12 transparently to the program, saving them to and restoring them from memory as necessary.

13 The *saved program state* shall contain the values of the set of 16 windowed registers that was active when  
14 the state was saved. The process of saving the program state shall include flushing the other hardware  
15 register window sets to the locations reserved for them on the program’s stack. If the registers that specify  
16 those locations are invalid, an Open Firmware implementation may omit the flushing of the  
17 corresponding register window sets.

18 The window register access commands (**%o0-%o7**, **%i0-%i7**, **%i0-%i7**) refer to one set of window  
19 registers at any given time; that set is known as the *displayed register set*. The various *displayed register*  
20 *sets* are denoted by small integers. Set zero is the set that was active when the program state was saved.  
21 Set one is the set that would be active if a RESTORE instruction were executed from set zero, and so on.  
22 The maximum set number is determined not by the number of hardware register window sets, but instead  
23 by the number of “logical” register window sets in use by the program at the time the state was saved.  
24 That maximum number can be less than, equal to or greater than the number of register windows  
25 implemented by the hardware.

26 When the Forth Interpreter (User Interface) is invoked, after the program state is saved, the *displayed*  
27 *register set* shall be set to zero, and may be changed with the **w** command. If the *displayed register set* is  
28 zero, the register values shall be accessed from the *saved program state*. Otherwise, the values shall be  
29 accessed from the appropriate save area in the program stack.

30 **%f0 through %f31** ( -- n )

31 Access floating point registers.

32 Return (or set, if preceded by **to**) the value corresponding to the contents of the register with the same name as the  
33 command.

34 **%fsr** ( -- n )

35 Access floating-point state register.

36 Return (or set, if preceded by **to**) the value corresponding to the contents of the register with the same name as the  
37 command.

38 **%g0 through %g7** ( -- n )

39 Access saved copies of “global” registers.

40 Return (or set, if preceded by **to**) the value, within the *saved program state*, corresponding to the contents of the register  
41 with the same name as the command.

42 **%i0 through %i7** ( -- n )

43 Access saved copies of “in” registers.

44 Return (or set, if preceded by **to**) the value, within the window register save area for the current display window,  
45 corresponding to the contents of the register with the same name as the command.

---

|    |  |                |
|----|--|----------------|
| 1  | <b>%l0 through %l7</b>   | ( -- n )       |
| 2  | Access saved copies of “local” registers.  |                |
| 3  | Return (or set, if preceded by <b>to</b> ) the value, within the window register save area for the current display window,         |                |
| 4  | corresponding to the contents of the register with the same name as the command.   |                |
| 5  | <b>%o0 through %o7</b>   | ( -- n )       |
| 6  | Access saved copies of “out” registers.  |                |
| 7  | Return (or set, if preceded by <b>to</b> ) the value, within the window register save area for the current display window,         |                |
| 8  | corresponding to the contents of the register with the same name as the command.   |                |
| 9  | <b>%pc and %npc</b>  | ( -- n )       |
| 10 | Saved program counter and next program counter.  |                |
| 11 | Return (or set, if preceded by <b>to</b> ) the value, within the <i>saved program state</i> , corresponding to the contents of the |                |
| 12 | register with the same name as the command.  |                |
| 13 | <b>%psr</b>  | ( -- n )       |
| 14 | Saved processor state register.  |                |
| 15 | Return (or set, if preceded by <b>to</b> ) the value, within the <i>saved program state</i> , corresponding to the contents of the |                |
| 16 | register with the same name as the command.  |                |
| 17 | <b>%tbr</b>  | ( -- n )       |
| 18 | Saved trap base register.  |                |
| 19 | Return (or set, if preceded by <b>to</b> ) the value, within the <i>saved program state</i> , corresponding to the contents of the |                |
| 20 | register with the same name as the command.  |                |
| 21 | <b>w</b>   | ( window# -- ) |
| 22 | Set the current window for display of the <b>%i?</b> , <b>%o?</b> , and <b>%l?</b> registers.                                      |                |
| 23 | <b>See also: .window.</b>  |                |
| 24 | <b>%wim</b>  | ( -- n )       |
| 25 | Window invalid mask register.  |                |
| 26 | Return (or set, if preceded by <b>to</b> ) the value, within the <i>saved program state</i> , corresponding to the contents of the |                |
| 27 | register with the same name as the command.  |                |
| 28 | <b>%y</b>  | ( -- n )       |
| 29 | y register.  |                |
| 30 | Return (or set, if preceded by <b>to</b> ) the value, within the <i>saved program state</i> , corresponding to the contents of the |                |
| 31 | register with the same name as the command.  |                |

1 The following commands display the *saved program state*.

2 **.locals** ( -- )

3 Display all the integer registers in the current window.

4 **.window** ( window# -- )

5 Display all the integer registers in the window specified by *window#*.

6 **Equivalent to: w .locals**

7 NOTE—the current window will be changed to the value given by *window#*.

8 **.psr** ( -- )

9 Formatted display of the saved processor state register.

10 This command sets the values in the program-counter registers.

11 **set-pc** ( a-addr -- )

12 Set **%pc** to *a-addr* and **%npc** to *a-addr+4*.

### 13 6.2. Debugger extensions

14 The commands **dis**, **+dis**, **.instruction**, and **.adr** shall display addresses and symbol name  
15 offsets in hexadecimal.

16 **return** ( -- )

17 Execute until a return from subroutine is reached.

18 Set a breakpoint at the address given by register **%i7** + 8 and then execute **go**.

19 **returnl** ( -- )

20 Execute until a return from subroutine is reached.

21 Same as **return** except uses **%o7** instead of **%i7**.

### 22 6.3. Configuration variables

23 **watchdog-reboot?** ( -- reboot? )

N

24 If **true**, reboot automatically after watchdog reset.

25 Configuration Variable Type: *boolean*. Suggested Default Value: **false**.

### 26 6.4. Restrictions

27 None.