IEEE Draft Std P1275.2/D14a Standard for Boot (Initialization Configuration) Firmware Supplement for IEEE 1496 (SBus) Bus

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.

Introduction

(This foreword is not a part of IEEE Draft Std P1275.2/D14a, Standard for Boot (Initialization Configuration) Firmware, Supplement for IEEE 1496 (SBus) Bus)

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

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

The core requirements and practices specified by IEEE Std 1275-1994 must be supplemented by system-specific requirements to form a complete specification for the firmware for a particular system. This standard establishes such additional requirements pertaining to the bus architecture defined by IEEE Std 1496-1993, *Standard for a Chip and Module Interconnect Bus: SBus*, also known as the SBus.

Copyright © 1992-1994 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Working Group Members

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

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

Contributors

The following individuals have contributed to this document:

Shawn Morrissey, FORCE COMPUTERS Inc.
Mike Williams, SHL
Paul Fischer, FORCE COMPUTERS Inc.
Ilan Rabinowitz, FORCE COMPUTERS Inc.

Sponsor Balloting Body

The following individuals were members of the Sponsor Balloting Body:

Copyright © 1992-1994 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table of Contents

Introduction ii
1. Overview 1
2. References 1
3. Terms 1
4. Bus characteristics 1
4.1. Physical address formats and representations 1
4.2. Bus-specific configuration variables 1
4.3. Format of a probe list 2
4.4. Interrupt-specification format 2
4.5. FCode interpretation semantics 2
5. Bus nodes 2
5.1. Properties 2
5.2. Methods 4
6. Child nodes 5
6.1. Properties 5
6.2. Methods 5
7. User interface extensions 5
7.1. Bus-specific interpretations of standard commands 6
7.2. Bus-specific FCodes 6
7.3. Bus-specific user interface commands 6

Copyright © 1992-1994 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1. Overview

This standard describes the application of IEEE Std 1275-1994, *Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices*, to the computer expansion bus defined by IEEE Std 1496-1993, *Standard for a Chip and Module Interconnect Bus: SBus*. This standard addresses both SBus controllers and devices intended to plug into an SBus.

2. References

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

[2] IEEE Std 1496-1993, *Standard for a Chip and Module Interconnect Bus: SBus*

3. Terms

This standard uses technical terms as they are defined in the documents cited in 2. plus the following terms:

3.1 bus node. In the device tree, a descendant node that represents the interface, or "bridge," between an SBus and its parent (which could be another bus).

3.2 child node. In the device tree, a descendant node that represents a device plugged into an SBus.

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

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

4. Bus characteristics

4.1. Physical address formats and representations

SBus uses geographical addressing with numbered slots. Slot numbers are small integers (e.g. 0, 1, 2, ...). Each slot has 28 (some earlier SBus implementations have only 25) address bits. The SBus standard imposes no "hard" limitation on the number of slots, but loading and capacitance considerations effectively limit the number of expansion slots to about four.

In addition to plug-in expansion slots, some SBus controllers have built-in, hardwired devices. Those built-in devices do not use a connector, but they are addressed as if they occupied an SBus slot. Some such devices use "virtual" slot numbers (i.e. not corresponding to a physical connector) as high as 15 (0xF). This specification accommodates slot numbers in the range 0 .. 0xF.

The numerical representation of an SBus physical address consists of the SBus slot number in the *phys.hi* number and the offset from the base of that slot in the *phys.lo* number. The text string representation is "`slot#,offset`", where both *slot#* and *offset* are the ASCII representations of hexadecimal numbers.

4.2. Bus-specific configuration variables

An IEEE 1275-compliant User Interface on a system with a single built-in SBus should implement the following SBus-specific Configuration Variable.

Copyright © 1992-1994 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

sbus-probe-list (-- list-str list-len) N

Holds list of slots to probe with **probe-sbus**.

A configuration variable containing a string whose maximum length is sufficient to contain a number of characters equal to the number of slots in the SBus implementation. The string is an SBus probe list as specified in 3.3.

Used as:

```
ok  setenv  sbus-probe-list  0123 <cr>
```

Configuration Variable Type: string.

Default value: a system-dependent value that includes all available slots.

4.3. Format of a probe list

A probe list for an SBus is a text string consisting of hexadecimal digits, each corresponding to an SBus slot number. The first character in the string specifies the first slot to be probed, and so on.

4.4. Interrupt-specification format

SBus has seven interrupt levels numbered 1 (lowest priority) through 7 (highest priority). SBus interrupts are not vectored at the SBus level. For the standard **interrupts** property, each interrupt level used by an SBus card is represented as a number from 1 to 7, encoded as with **encode-int**.

4.5. FCode interpretation semantics

Each SBus card has an FCode Program beginning at offset zero within its slot. The FCode bytes appear at successive byte addresses. The maximum size of an FCode Program on an SBus card is 64K bytes.

An SBus slot is probed as specified for the core word **probe-self**, with the following bus-specific stipulations:

– The check for device present shall be performed by attempting to read the byte at offset zero within the slot. A hardware error acknowledge in response to the attempted read indicates that no device is present in the slot. If the read succeeds, and the returned byte is a valid FCode start code, the FCode Program shall be interpreted as with **byte-load**, beginning at the address corresponding to offset zero of that slot, with a *spread* value of one.

NOTE—Some historical implementations checked for device presence by performing a 32-bit read first, followed by an 8-bit read, in order to work around a hardware bug that was present on the earliest SBus systems. Later historical implementations performed only 8-bit reads. This standard specifies the latter behavior. However, it is unwise, and generally unnecessary, for hardware designs to depend upon either behavior, because the automatic bus-sizing capability of SBus host bridge hardware allows an SBus card to easily support either size access from the CPU's point of view.

– If it is necessary to map the region containing the FCode Program, at least 64K bytes shall be mapped. Once the FCode Program at offset zero has been interpreted, the region containing that program is no longer needed as far as this specification is concerned, and may be unmapped.

5. Bus nodes

5.1. Properties

5.1.1. Core-specified properties for bus nodes

The following standard properties in the core specification have special interpretations for SBus.

"**device_type**" S

Standard *prop-name* to specify the implemented interface .

A Standard Package conforming to this specification and corresponding to a device that implements an SBus shall implement this property with the string value "sbus".

Copyright © 1992-1994 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

"**ranges**" S

Standard *prop-name* to specify child-to-parent address mapping.

The format of child physical addresses is as specified in 4.1.

"**reg**" S

Standard *prop-name* to define the package's registers.

For "transparent" SBus-to-Sbus bridges with no addressable registers, the "**reg**" property should specify the region of the parent's address space that is "mapped through" to the children of the SBus.

5.1.2. Bus-specific properties for bus nodes

"**burst-sizes**" S

prop-name, denotes transfer sizes allowed on this SBus.

prop-encoded-array: Integer that represents the set of transfer sizes that the controller allows for its SBus, encoded with **encode-int**. The value is the sum of the applicable values from the table below.

value	allowed transfer size
1	1 byte transfer
2	2 byte transfer
4	4 byte transfer (1 32-bit cycle)
8	8 byte transfer (2 32-bit cycles)
0x10	16 byte transfer (4 32-bit cycles)
0x20	32 byte transfer (8 32-bit cycles)
0x40	64 byte transfer (16 32-bit cycles)
0x8.0000	8 byte extended transfer (1 64-bit cycle)
0x10.0000	16 byte extended transfer (2 64-bit cycles)
0x20.0000	32 byte extended transfer (4 64-bit cycles)
0x40.0000	64 byte extended transfer (8 64-bit cycles)
0x80.0000	128 byte extended transfer (16 64-bit cycles)

The absence of this property is equivalent to a value of 0x17. This is a concession to existing SBus implementations which are not IEEE 1496-1993 compliant.

NOTES

—An IEEE 1496-1993-compliant SBus Controller shall allow all 32-bit transfer sizes, and if it allows any 64-bit transfers, it shall allow all 64-bit transfer sizes. Thus, for IEEE 1496-1993 compliant devices, the value of this property is either 0x7F (64-bit transfers are not allowed) or 0xF8.007F (64-bit transfers are allowed).

—Individual devices on the bus need not allow all transfer sizes; this property describes the characteristics of the bus itself as implemented by the controller.

"**bus-parity-generated**" S

prop-name, the presence of which denotes SBus parity generation.

prop-encoded-array: Zero-length array.

This property shall exist if and only if the associated device generates parity on its SBus.

"**clock-frequency**" S

prop-name to specify the bus clock frequency.

prop-encoded-array: Integer clock frequency in hertz, encoded with **encode-int**.

Typical values range from 16000000 to 25000000.

"**one-pending-retry**" S

prop-name, the presence of which denotes an SBus controller restriction.

prop-encoded-array: Zero-length array.

This property shall exist if and only if the associated SBus controller restricts retries from a particular slot to use the same address until the retry cycle is completed, as opposed to being able to interleave retries with different addresses.

"**slot-address-bits**" S

prop-name to specify the slot size.

prop-encoded-array: Integer number of address bits per slot, encoded with **encode-int**.

Typical values are either 25 or 28.

Copyright © 1992-1994 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

"**up-burst-sizes**" S

prop-name, denotes transfer sizes allowed to its parent bus.

prop-encoded-array: Integer that is encoded in the same way as the value of the "**burst-sizes**" property.

This property shall apply to a bus node that can act as a slave on its SBus, its value representing the set of transfer sizes the bus node can accept as a slave for forwarding to its parent bus.

5.2. Methods

5.2.1. Core-specified methods for bus nodes

A Standard Package implementing the "**sbus**" device type shall implement the following standard methods from the core specification with physical address representations as specified in 3.1.

open	(-- okay?)	Prepare this device for subsequent use.
close	(--)	Close this previously-**open**ed device.
map-in	(phys.lo phys.hi size -- virt)	Map the specified region, return a virtual address.
map-out	(virt size --)	Destroy mapping from previous **map-in**.
dma-alloc	(size -- virt)	Allocate a memory region for later use.
dma-free	(virt size --)	Free memory allocated with **dma-alloc**.
dma-map-in	(virt size cacheable? -- devaddr)	Convert virtual address to device bus DMA address.
dma-map-out	(virt devaddr size --)	Free DMA mapping set up with **dma-map-in**.
dma-sync	(virt devaddr size --)	Synchronize (flush) DMA memory caches.
probe-self	(arg-str arg-len reg-str reg-len fcode-str fcode-len --)	Interpret FCode, as a child of this node.
decode-unit	(addr len -- phys.lo phys.hi)	Convert text unit-string to physical address.

5.2.2. Bus-specific methods for bus nodes

A Standard Package implementing the "**sbus**" device type shall implement the following bus-specific methods.

unique-key (-- byte) M

Returns unique "key" byte value.

Returns a unique value in the range {0x01 .. 0xEF} each time it is called. If **unique-key** is called more than 128 times before the system is reset, the return value is undefined . Return values are not unique across system resets.

The value is intended for use as a "key" value for the "Write-to-zero" technique of implementing transparent SBus-to-SBus bridge devices, as defined by IEEE 1496-1993.

Used by children of an SBus device node as:

```
" unique-key" $call-parent ( key-byte )
```

The implementation of **unique-key** for a device that is an SBus-to-SBus bridge shall call the **unique-key** method of its parent device, thus assuring uniqueness of the returned value throughout the chain of interconnected SBuses.

Copyright © 1992-1994 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6. Child nodes

6.1. Properties

6.1.1. Core-specified properties for child nodes

The following standard properties in the core specification have special meanings or interpretations for SBus.

"**reg**" S

Standard *prop-name*, defines device addresses.

This property is mandatory for SBus Child Nodes. Its definition is given in the core specification, with the following additional requirements:

If there are no addressable regions on the device other than the FCode ROM, a *phys.lo* value of zero and a size equal to the size of the FCode ROM shall be specified. If addressable regions other than an FCode ROM do exist, it is conventional to not include the FCode ROM address range in the list.

"**interrupts**" S

Standard *prop-name*, defines the interrupts used.

prop-encoded-array: *n* integers, each in the range 1–7, each encoded with **encode-int**. Each integer represents an SBus interrupt level used by this SBus card.

A Standard Package corresponding to a device that is capable of generating SBus interrupts shall implement this property.

6.1.2. Bus-specific properties for child nodes

Standard Packages corresponding to devices that are children of an SBus may implement the following properties.

"**parity-generated**" S

prop-name, the presence of which denotes SBus parity generation.

prop-encoded-array: Zero-length array.

This property shall exist if and only if the associated device generates parity on its SBus.

"**slave-burst-sizes**" S

prop-name, denotes transfer sizes allowed to the device.

prop-encoded-array: An array of integers, each encoded in the same way as the value of the "**burst-sizes**" property. Each array-entry value represents the allowed transfer sizes of the address range denoted by the corresponding entry in the "**reg**" property.

"**down-burst-sizes**" S

prop-name, denotes transfer sizes allowed to a subordinate bus.

prop-encoded-array: Integer that is encoded in the same way as the value of the "**burst-sizes**" property.

This property shall apply to a child node, typically a bus bridge, that can act as an SBus slave, its value representing the set of transfer sizes the child node can accept as a slave for forwarding to a subordinate bus.

6.2. Methods

There are no special requirements on the methods for SBus Child Nodes beyond those described in the core specification.

7. User interface extensions

This clause describes commands which are only applicable to SBus devices or have special meaning when used with SBus devices.

Copyright © 1992-1994 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

7.1. Bus-specific interpretations of standard commands

None.

7.2. Bus-specific FCodes

None.

NOTE—Many existing tokenizers accept the name **map-sbus** as a synonym for **map-low**; **map-sbus** was the original name of that function. The core specification has generalized its meaning to apply to any bus.

7.3. Bus-specific user interface commands

An IEEE 1275-compliant User Interface on a system with a built-in SBus should implement the following SBus-specific user interface commands.

probe-slot (slot# --)

Interprets FCode in the indicated SBus slot.

Selects the device node for the primary SBus by opening the node (as with **open-dev**), making the result the current instance, and making that node the active package. Execute that node's **probe-self** method with the following arguments:

arg-addr arg-len	A zero-length string.
reg-addr reg-len	A string of the form "`slot#,0`", where slot# is a hexadecimal digit corresponding to **probe-slot**'s *slot#* argument.
fcode-addr fcode-len	Same string as for *reg-addr reg-len*.

After the **probe-self** method returns, unselect the SBus device node by restoring the active package to its previous value, closing the current instance (as with **close-dev**), and restoring the current instance to its previous value.

Note: The name **probe-slot** presumes that SBus is the only expansion bus on the system. This is a concession to existing systems. Should a corresponding function exist in future bus supplements, it is recommended that that function's name include the name of the bus.

probe-sbus (--)

Interprets FCode for all SBus slots in **sbus-probe-list**.

For each character in the string given by **sbus-probe-list**, execute **probe-slot** with the number corresponding to that character, interpreted as a hexadecimal digit, as its argument. The first character in the string corresponds to the first slot to be probed, and so on.

In systems with a built-in SBus, **probe-all** should include the function of **probe-sbus**.

Copyright © 1992-1994 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.