ISA/EISA/ISA-PnP binding to:

IEEE Std 1275-1994

Standard for Boot

(Initialization, Configuration)

Firmware

Revision: 0.4 (Unapproved Draft)

Date: September 23, 1996

Purpose of this ISA/EISA/ISA-PnP binding

This document specifies the application of Open Firmware to the bus used on the IBM® Personal Computer/AT™, commonly called the Industry Standard Architecture (ISA) Bus, including requirements and practices for address formats, interrupts, probing, and related properties and methods specific to the ISA Bus.

This document also specifies the application of Open Firmware to the 32 bit extension to the ISA standard, developed by the COMPAQ®-led consortium, known as the Extended Industry Standard Architecture (EISA™).

This document also extends the resource data structures defined for Plug and Play ISA (PnP ISA) adaptor cards to be used for configuring both legacy ISA and EISA devices.

The core requirements and practices specified by Open Firmware must be augmented by system-specific requirements to form a complete specification for the firmware for a particular system. This document establishes such additional requirements pertaining to the ISA/EISA Bus.

Task Group Members

The following individuals were members of the Task Group that produced this document:

Mitch Bradley, FirmWorks

Jordan Brown, SunSoft Inc.

Bob Coffin, (editor), IBM Corporation

David Kahn, Sun Microsystems

Dr. Luan Duy Nguyen, IBM Corporation

Lilian Walter, FirePower System Inc.

Registration Symbols and Trademarks

The following terms, denoted by a registration symbol (®) or trademark symbol(™) on the first occurrence in this publication, are registered trademarks or registration symbols of the companies as shown in the list below:

Trademark	Company
COMPAQ	COMPAQ Corporation.
IBM & IBM PC	International Business Machines
PC/XT & PC/AT	International Business Machines

All products or services mentioned in this document are identified by the trademarks, service marks, or product names as designated by the companies who market those products. Inquiries concerning such trademarks should be made directly to those companies.

Revision History

Revision 0.00	07/94	First version.

Revision 0.01	10/94	Simplify first cell of reg property. Drop “m” from text representation.
Revision 0.02	01/95	Miscellaneous changes, mostly editorial. Add support for twenty-bit aliasing in memory space.
Revision 0.03	03/21/95	Added EISA bus specification and some PnP ISA device support.
Revision 0.04	09/25/95	Added PnP ISA resource data structures and extend it to accommodate EISA configuration.
Revision 0.05	10/11/95	Changed document format and add Table of Contents. Changed content according to approval from the committee on 09/19/1995
Revision 0.1	02/28/96	Changed content according to approval from the committee on 01/16/96. Changed version number from .05 to 0.1.
Revision 0.2	04/12/96	Modified physical address representation (Section 2.2.1) to reflect PnP Format, subtractive decode property definition(Section 3.1.2), the definition of dma-alloc(Section 3.2.1), the “name” and “reg” properties (Section 4.1.1), changed “connectors” to “slot-names”(Section 4.1.2) and removed the “interrupt-choices” and “dma-choices” properties because the PnP Data Format contains the information in the dependent section (Section 6.3.13 & 6.3.14). Numerous editorial changes.
Revision 0.3	08/08/96	Added *Interrupt Mapping* Reference, removed proposed PnP *unit address* format, changed physical address format (‘i’ is optional and represents I/O Address Space and ‘m’ represent Memory Address Space, changed Child Nodes **“name”**, **“compatible”** and **“reg”** properties. Added **“pnp-id”** property. Numerous editorial changes.
Revision 0.4	09/23/96	Added **“slot-names-index”** property to Section 4.1.2. Changed wording of **“compatible”** property to reflect convention to represent PnP-ISA Adapters with multiple logical devices. Several editorial changes done.

Table of Contents

1. Overview and References

References and terms for the ISA/EISA/ISA-PnP binding are listed in this section.

1.1. References

This Open Firmware ISA/EISA/ISA-PnP binding standard *shall* be used in conjunction with the following publications. When the following standards are superseded by an approved revision, the revision *shall* apply.

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

[2] *Technical Reference, Personal Computer AT*; IBM part number 6280070 [or S229-9611-00 or 6139362.]

[3] *IEEE P996 Personal Computer Bus Standard*

[4] *Extended Industry Standard Architecture Specification*, Rev. 3.12, BCPR Service

[5] *Plug and Play Option ROM Specification*

[6] *Plug and Play BIOS Specification*, Ver 1.0a, May 1994

[7] *Plug and Play ISA Specification*, Ver 1.0a, May 1994

[8] *PCI Bus Binding to IEEE 1275*

[9] *Open Firmware Recommended Practice - Generic Names*

[10] *Open Firmware Recommended Practice - Interrupt Mapping*

1.2. Definitions of Terms

This standard uses technical terms as they are defined in the documents cited in "References", plus the following terms:

Bus controller: a hardware device that implements an ISA/EISA bus.

bus node: an Open Firmware device node that represents a bus controller. In cases where a node represents the interface, or “bridge”, between one bus and another, the node is both a bus node relative to the bus it controls, and a child node of its parent bus. Note that an Open Firmware device node is not in itself a physical hardware device; rather, it is a software abstraction that describes a hardware device.

child node: an Open Firmware device node that represents an ISA/EISA function. Such a node can correspond to either a device that is “hardwired” to a planar ISA/EISA bus, or to an “add in” expansion card that is plugged into a standard ISA/EISA expansion connector.

EISA: Extended Industry Standard Architecture

EISA device: a hardware device that connects to or “plugs in” to an EISA bus: one of a number of logically-independent parts of an EISA device. Many EISA devices have only one function per device; in such cases, the terms “EISA function” and “EISA device” can be used interchangeably.

ISA: Industry Standard Architecture.

ISA device: a hardware device that connects to or “plugs in” to an ISA/ EISA bus: one of a number of logically-independent parts of an ISA device. Many ISA devices have only one function per device; in such cases, the terms “ISA function” and “ISA device” can be used interchangeably.

PnP ISA device: a new generation of ISA cards that incorporate hardware mechanisms that enables resolution of resource conflicts between PnP ISA cards solely by software. A PnP ISA card may have more than one logical device, or, alternately, function.

2. Bus Characteristics

This section describes the ISA/EISA Bus addressing and configuration characteristics and/or attributes.

2.1. Address Spaces

ISA/EISA has two address spaces (Memory, I/O) with different addressing characteristics.

2.1.1. Memory Space

Memory Space is the primary address space of the ISA/EISA bus; it corresponds to traditional memory and “memory-mapped” I/O.

The ISA bus allows for a 24-bit address space. The “8-bit” or “PC/XT” subset allows for a 20-bit address space, but, since hardware protocols prevent an appropriate decoding of such devices, no special provisions need to be made by this binding to deal with this addressing structure.

The EISA bus allows for a 32-bit address space.

2.1.2. I/O Space

I/O Space is similar to Memory Space, except that it is intended to be used with the special “I/O access” instructions that some processors have.

The ISA/EISA bus allows for a 16-bit I/O address space. The “8-bit” subset allows for a 10-bit or 11-bit aliased address space.

Devices are allowed to “alias” I/O addresses by ignoring all but the lower 10 bits of an I/O address. To conserve **“reg”** property space, a bit (the 't' bit, for ten-bit alias) is included in the encoding of I/O addresses to indicate that the corresponding **“reg”** range includes all such aliases.

Devices are also allowed to “alias” I/O addresses by ignoring all but the lower 11 bits of an I/O address. To conserve **“reg”** property space, a bit (the 'v’' bit, for eleven-bit alias) is included in the encoding of I/O addresses to indicate that the corresponding **“reg”** range includes all such aliases.

2.2. Address Formats and Representations

This section describes the physical address representation and the Open Firmware implications.

2.2.1. Physical Address Format: Numerical Representation

The numerical representation of an ISA/EISA address *shall* consist of two cells. For the numerical address format, the least-significant 32 bits of a cell is used. If the cell size is larger than 32 bits, any additional high-order bits are zero. Bit #0 refers to the least-significant bit. The decode format of the physical address is as follows:

```
        Bit#: 33222222 22221111 11111100 00000000
              10987654 32109876 54321098 76543210
phys.hi cell: 00000000 00000000 00000000 00000vti
phys.lo cell: nnnnnnnn nnnnnnnn nnnnnnnn nnnnnnnn
```

where for *phys.hi*:

t is 1 if the address is 10-bit aliased (for I/O)

v is 1 if the address is 11-bit aliased (for I/O)

i is 1-bit field denoting memory or I/O space

and for *phys.lo*:

nn..nn is a 32-bit unsigned number

Encoding of field “i”:

1 denotes I/O space, in which case:

t is set if 10-bit aliasing is present.

v is set if 11-bit aliasing is present.

nn..nn must be 65535 or less.

0 denotes Memory Space, in which case:

t is always 0.

v is always 0.

nn..nn is the 32-bit Memory Space address

Note: Since the numeric representation of an ISA/EISA physical address consists of 2 address cells, and the number of cells used to encode the size field of a child's reg property is 1(these are the IEEE 1275 defaults), explicit implementation of the standard properties #address-cells and #size-cells need not be present.

2.2.2. Physical Address Format: Text Representation

The text representation of an ISA/EISA address is one of the following forms:

[i]tNNNN ,[i]vNNNN or [i]NNNN where the strings are an I/O Address Space

or

mNNNNNNNN where the string is a Memory Address Space

where:

NNNN is an ASCII hexadecimal number in the range 0..0xFFFF

NNNNNNNN is an ASCII hexadecimal number in the range 0..0xFFFFFFFF

t is the letter 't', whose presence indicates 10-bit aliasing for I/O Address Space

v is the letter 'v', whose presence indicates 11-bit aliasing for I/O Address Space

m is the letter 'm', indicates Memory Address Space

i is the letter 'i', whose presence is optional, indicates I/O Address Space

Note: If string does not begin in ’m’, ’i’, ’t’ or ’v’, an I/O address format is assumed.

2.2.3. Correspondence between two representations

The correspondence between text representations and numerical representations is as follows:

[i][t]NNNN: corresponds to an I/O Space address with the numerical value of 0xNNNN. If 't' is present, only the low-order 10 bits of an I/O address range is indicated and aliases are assumed for all high-order bits within the ISA 64KB I/O address space.

The numerical value is:

i is 1

nn..nn is the value whose hex representation is NNNN

[i][v]NNNN: corresponds to an I/O Space address with the numerical value of 0xNNNN. If 'v' is present,

only the low-order 11 bits of an I/O address range is indicated and aliases are assumed for all high-order bits within the ISA 64KB I/O address space.

The numerical value is:

i	is 1
nn..nn	is the value whose hex representation is NNNN

mNNNNNNNN: corresponds to a Memory Space address. The numerical value is:

i	is 0
nn..nnn	is the value whose hex representation is NNNNNNNN

Conversion of hexadecimal numbers from text representation to numeric *shall* be case-insensitive, and leading zeros *shall* be permitted but not required.

Conversion from numeric representation to text representation *shall* use the lower case forms of the hexadecimal digits in the range a..f, suppressing leading zeroes.

2.2.4. Unit Address Representation

As required by this specification's definition of the **`"reg"`** property, the first entry or *phys-addr shall* be an *unit address* for an ISA/EISA Node. The *unit address* form is shown below for an ISA, ISA-PnP or EISA Adapter and is represented by Section 2.2.2. on page 9, *Physical Format: Text Representation* where, in summary form, is stated below:

I/O *Unit Address* - `[i]tNNNN`, `[i]vNNNN` or `[i]NNNN`

where `i` is optional.

or

Memory *Unit Address* - `mNNNNNNNN`

Note: It is an Open Firmware platform responsibility to maintain the ISA *Unit Address* persistent from boot-to-boot. If the assigned ISA *Unit Address* is changed for an Adapter (E.g.; due to a platform re-configuration and possible addition of a new ISA Adapter), then the OS may have to be reinstalled.

2.2.5. Open Firmware Implication

Open Firmware assumes a single address space model. ISA/EISA I/O space is a distinct address space from memory space, and so requires special consideration. On some processors, I/O space is physically mapped as a subset of memory space and so can be conveniently accessed using standard register access primitives **(`rb@`**, **`rw!`**, **`etc`.)**; all that need be done is that map-in recognize an I/O space physical address and offset it by the base of the I/O space area. On other processors, special instructions must be executed to access I/O space. On these processors, it is recommended that Open Firmware reserve a block of virtual addresses for I/O space access. When **`map-in`** is given an I/O physical address, it would return a virtual address in this reserved range. The register access words would examine each address they are given, and if the address is in the reserved range they would execute the special instructions required to access I/O space.

2.3. Bus-specific Configuration Variables

An Open Firmware-compliant User Interface on a platform with a single built-in EISA bus may implement the following EISA-specific Configuration Variable (Refer to [1], Appendix A to define the 'type code field' N for **`eisa-probe-list`**).

`eisa-probe-list` `( -- list-str list-len` **)** **N**

2.3.1. Format of a Probe List

Each EISA expansion slot is assigned an unique address, the most significant nibble (bits 12 to 15) of 16-bit I/O address. 0 is assigned to the embedded or plug-in ISA devices on the EISA bus. 1 is assigned to EISA slot one, and so on.

The probe list is thus a comma-separated list of EISA slots to probe, e.g. 1,2,3,4,5,6 on a 6-slot EISA bus.

There are no corresponding probe lists for ISA and PnP ISA because of the following:

> The configuration data of ISA adapter cards is stored in NVRAM
> (See “Format of Data Resource Information” on page 19.). By the nature of ISA cards, all the resources that are requested must be met.
>
> PnP ISA cards cannot be addressed with a physical address by slot.

3. Bus Nodes

This section describes the Open Firmware properties and methods for the ISA and EISA buses.

> **Note: A bridge is a parent of one bus and is a child of another. Consequently, a node representing a bridge is both a Bus Node and a Child node, with both sets of properties and methods.**

3.1. Properties

This section defines ISA and EISA bus Open Firmware Properties.

3.1.1. Open Firmware Properties for Bus Nodes

The following standard properties, as defined in Open Firmware, have special meanings or interpretations for the ISA/EISA bus.

`“name”` **`S`**

Standard *prop-name* to specify the name of the package.

prop-encoded-array: a string encoded as with **`encode-string`**.

The meaning of this property is as defined in Open Firmware. The name of the package defining the legacy AT style I/O bus is “*isa*.”

`“device_type”` **`S`**

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

prop-encoded-array: a string encoded as with **`encode-string`**.

The meaning of this property is as defined in Open Firmware. A Standard Package conforming to this specification and corresponding to a device that implements an ISA bus *shall* implement this property with the string value “*isa*”. A device that implements an EISA extension *shall* implement this property with the string value “*eisa*”.

`“ranges”` **`S`**

Standard *prop-name,* defines the mapping of parent address to child address spaces.

This property *shall* be present for all ISA/EISA bus bridges. There *shall* be an entry in the **`“ranges”`** property for each of the Memory and/or I/O spaces if that address space is mapped through the bridge. The format for physical addresses entries is as specified in Section 2.2. on page 8.

The child address space requirements are defined in this binding document and the parent address space requirements are defined by the appropriate bus binding.

3.1.2. Bus-specific Properties for Bus Nodes

The following bus-specific properties have special meanings or interpretations for the ISA and/or EISA bus.

`“eisa-slots”` **`S`**

prop-name to specify the bit map of EISA slots.

`“clock-frequency”` **`S`**

prop-name to specify the bus clock frequency.

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

`"slot-names"` **`S`**

prop-name: Describes external labeling of add-in platform bus slots.

prop-encoded-array: An integer, encoded as with **`encode-int`**, followed by a list of strings, each encoded as with **`encode-string`**.

The integer portion of the property value is a bitmask of available bus slots; for each add-in slot on the ISA bus, the bit corresponding to that slot’s ID number is set from least-significant to most-significant ID number. The number of following strings is the same as the number of slots; the first string gives a label for the slot with the smallest ID number, and so on. The string contents would be platform dependent. The absence of this property indicates no ISA slots.

Note: For an ISA bus, a method to get this information does not exist. Typically, a platform could provide a means to allow a user through a configuration utility to store a slot name in NVRAM (or other non-volatile storage). Open Firmware would then construct this property from the slot name information stored in NVRAM.

Note: The `"slot-names-index"` is used to provide the index into the available ISA slots.

`“subtractive-decode”` **`S`**

prop-name: If the bridge supports substractive decode, the property *shall* be present. If the property is present, the implication is that the memory address and I/O address space entries in the **`"ranges"`** property are subtractively decoded. The ISA/EISA Bridge will respond to an access not claimed by the parent of the ISA Node within the definition of the **`"ranges"`** property.

3.2. Methods

This section defines methods for the ISA and/or EISA Bus and Child Nodes.

3.2.1. Standard Open Firmware-defined Methods

A Standard Package implementing the “*isa*” or “*eisa*” device type *shall* implement the following standard methods as defined in Open Firmware, with the physical address representations as specified in Section 2.2. on page 8 of this document, and with additional ISA-/EISA-specific semantics:

`open` `( -- okay?)` **`M`**

Prepare this device for subsequent use.

`close` `( -- )` **`M`**

Close this previously-open device.

`map-in` `( phys.lo phys.hi size -- virt )` **`M`**

Map the specified subregion.

`map-out` `( virt size -- )` **`M`**

Destroy mapping from previous **`map-in`**.

`dma-alloc` `( size -- virt )` **`M`**

Allocate memory suitable for DMA use, by ISA and EISA. Since many I/O devices are restricted to 24-bit addressing, the memory that is allocated must be in an area that can be reached by 24-bit addressing.

```
dma-free        ( virt size -- )                          M
```

Free memory allocated with dma-alloc.

```
dma-map-in      ( virt size cacheable? -- devaddr )       M
```

Convert virtual address to device bus DMA address.

```
dma-map-out( virt devaddr size -- )                       M
```

Free DMA mapping set up with **dma-map-in**.

```
dma-sync        ( virt devaddr size -- )                  M
```

Synchronize (flush) DMA memory caches.

```
probe-self      ( arg-str arg-len reg-str reg-len fcode-str fcode-len
                -- )                                      M
```

If FCode exists, make a child node for adapter. Used to probe both ISA and EISA devices.

```
decode-unit     ( addr len -- phys.lo phys.hi )           M
```

Convert text representation of address to numerical representation.

```
encode-unit     ( phys.lo phys.hi -- addr len )           M
```

Convert numerical representation of address to text representation.

3.2.2. Bus-specific Open Firmware-defined Methods

A standard package implementing the “*isa*” or “*eisa*” device type *shall* implement the following bus-specific method as defined in Open Firmware, with the physical address representations as specified in Section 2.2. on page 8 of this document, and with additional ISA-/EISA-specific semantics:

```
probe-pnp       ( -- )                                    M
```

Used to probe all PnP ISA cards.

4. Child Nodes

Logical devices and/or independent functions of a physical adapter card *shall* each be a child node of the “*isa*” or *"eisa"* bus node.

There are potentially four sources of input to Open Firmware in creating the child nodes:

1) For platform specific ISA and/or EISA devices (*built-in* devices), Open Firmware provides information directly in creating nodes in device tree.

2) For a legacy ISA and EISA adapter cards, the resource allocation information is provided by the user via some configuration utilities. The information is stored in NVRAM (or some other non-volatile storage). The format of this data is based on the PnP ISA resource data structures defined for PnP ISA adapter cards (presented in a later section of this document, Section 6. on page 18).

3) For a PnP ISA adapter card, the resource information is obtained directly from the adapter card.

4) For an ISA/EISA adapter card with FCode supported, the resource information is obtained directly from the adapter card.

4.1. Properties

This section defines core-specified and bus-specific child node properties.

4.1.1. Open Firmware Properties for Child Nodes

The following properties, as defined in Open Firmware, have special meanings or interpretations for the ISA/EISA bus.

"name" S

Standard *prop-name* to specify the name of the child node

prop-encoded-array: a string encoded as with **encode-string**. The name can be a generic name or a string in the form of pnp*vvv,pppp (*Refer to *Recommended Practice - Generic Names* [9]*)* where the string form *vvv* is a Vendor ID and the string *pppp* is a Product Number (See "Header" on page 19. for the string form).

The strings are defined to be in the following format:

pnp - literal string "pnp"
vvv - upper case ASCII characters with trailing blanks eliminated
pppp - lower case hexadecimal characters with leading zeroes eliminated

"compatible" S

Standard *prop-name:* Defines alternate **"name"** property values.

prop-encoded-array: The concatenation, with **encode+**, of an arbitrary number of text strings, each encoded with **encode-string**. The first entry should be an explicit, unique name that identifies the device precisely enough to be able to infer a detailed programming model *(*Refer to *Recommended Practice - Generic Names* [9]*).*

ISA, EISA or ISA PnP Card without Multiple Logical Devices:

The **"compatible"** property should consist of the following entry with a string value in the form of pnp*vvv,pppp* where *vvv,pppp* is a vendor ID.

ISA PnP Card with Multiple Logical Devices:

Each logical function is represented by it's own node in the device tree for the ISA Bus. For PnP-ISA cards, the **"compatible"** property should consist of the following string entries, in the order as shown:

1a) If the device has multiple logical devices:

The property value should contain a string in the form of pnp*vvv,pppp,fff* where *vvv,pppp* is the PnP vendor ID for the adapter (See "Header" on page 19.) and *fff* is the number of the logical device (See "Logical Device ID Record" on page 22.) with 0 denoting the first logical device, 1 denoting the second logical device, and so on.

1b) If the device has only a single logical device:

The property value should contain a string in the form of pnp*vvv,pppp* where *vvv,pppp* is the PnP vendor ID for the adapter.

2) If the device supplies a logical device ID:

The property value should contain a string in the form of pnp*vvv,pppp* where *vvv,pppp* is the PnP logical device ID for the adapter.

3) If the device supplies one of more PnP *compatible ID* records, then for each record in order:

The property value should contain a string in the form of pnp*vvv,pppp* where *vvv,pppp* is the PnP compatible ID (See "Compatible Device ID Record" on page 23.) for each adapter record.

"reg" S

Standard *prop-name*, defines device's addressable regions.

prop-encoded-array: array with an arbitrary number of (*phys-addr size*) pairs.

phys-addr is (phys.lo phys.hi), encoded as with **encode-phys**.

size is an integer, encoded as with **encode-int**.

The property value consists of a sequence of (*phys-addr size*) pairs. Each (*phys-addr size*) pair *shall* specify the address of an addressable region of Memory Space or I/O Space associated with the function. The first entry or component of the **"reg"** property is a *unit address* (See "Unit Address Representation" on page 10.).

Order of the **"reg"** property entries are determined by the device binding.

Note: As a general guideline for new device bindings, the preferred order of Memory and I/O addresses are in an ascending order.

"interrupts" **S**

Standard *prop-name*, the presence of which indicates that the function represented by this node is connected to an ISA/EISA bus interrupt line.

prop-encoded-array: list of an arbitrary number of integer pairs (*irq# type*) where:

irq# is an integer, encoded as with **encode-int**. The *irq#* integer represents the interrupt line to which this function's interrupt is connected. This value is in the range 0..15 (decimal).

type is an integer, encoded as with **encode-int**, indicating an interrupt type. The interrupt types are represented by the following integer values:

0 =	active low level sensitive type enabled
1 =	active high level sensitive type enabled
2 =	high to low edge sensitive type enabled
3 =	low to high edge sensitive type enabled

Refer to "IRQ Format Record" on page 23 for where the **"interrupts"** property obtains the interrupt attributes or characteristics. Also refer to *Recommended Practice - Interrupt Mapping* [10] for Open Firmware interrupt structure.

Note: ISA interrupt controller legacy uses Interrupt 2 or 9 to cascade two 8259's for 16 interrupt lines implementation.

"status" **S**

Standard *property name*, indicates the operational status of the device.

prop-encoded-array: Text string, encoded as **encode-string**.

4.1.2. Bus-specific Properties for Child Nodes

Standard Packages corresponding to devices that are children of an ISA or EISA bus *shall* implement the following properties, if applicable.

"dma" **S**

Standard prop-name, indicates that the function represented by this node is connected to an ISA/EISA bus DMA resource.

prop-encoded-array: list of an arbitrary number of integers (*dma# mode width countwidth busmaster*) indicating the dma channel number, dma mode, dma transfer width and dma count width where:

dma# is an integer, encoded as with **encode-int**, where the value represents the DMA resource to which this function is connected. This value is in the range of 0-3 and 5-7.

Note: DMA Channel 4 is used for cascading of 8259's and is not available.

mode is an integer, encoded as with **encode-int**, where the value indicates the mode of the corresponding dma channel:

0	device runs in compatibility mode.
1	device runs in DMA "A" mode.
2	device runs in DMA "B" mode.
3	device runs in DMA "F" mode.
4	device runs in DMA "C" mode.

width is an integer, encoded as with **encode-int**, where the value represents the dma transfer size of the corresponding dma channel. The value of the integer will be 8, 16 or 32.

countwidth is an integer, encoded as with **encode-int**, where the value represents the count width of the corresponding dma channel. The value of the integer will be 8, 16 or 32.

busmaster is an integer, encoded as with **encode-int**, where a value of 1 indicates a bus mastering capability and a 0 indicates no bus mastering capability.

"slot-names" **S**

prop-name: Describes external labeling of a connector(s).

prop-encoded-array: An integer, encoded as with **encode-int**, followed by a list of strings, each encoded as with **encode-string**.

The integer portion of the property value is a bitmask of available connectors; for each connector associated with the function, the bit corresponding to that connector's ID number is set from least-significant to most-significant ID number. The number of following strings is the same as the number of connectors; the first string gives a label for the connector with the smallest ID number, and so on. The string contents would be platform dependent. The absence of this property indicates no connectors.

Note: Each device that has a connector would identify the order and contents of the list of strings.

"slot-names-index" **S**

prop-name: Describes each add-in ISA slot with a unique number.

prop-encoded-array: An integer, encoded as with **encode-int**.

The value of this integer is a unique number with a range of 0 to *n*-1 for each ISA slot where *n* is the number of ISA add-in or plug-in slots. This number is used to index into the **"slot-names"** property to identify the value of the string associated with the slot name. The absence of this property indicates no ISA slots.

"eisa" **S**

prop-name: if exists, indicates an EISA device.

"pnp-csn" **S**

prop-name, is a property of device whose bus-interface is **"pnp"**.

prop-encoded-array: Integer encoded with **encode-int**. This is the Card Selection Number (CSN) that the Open Firmware assigned to the device represented by this node in the process of PnP device isolation.

"description" **S**

prop-name, is a property of which the value is the identifier string from the PnP data resource record.

prop-encoded-array: a string, as encoded with **encode-string**, of the identifier string from bytes 3 through *n* from the "ANSI Identifier String Record" on page 22.

"pnp-data" **S**

prop-name, is a property of device whose bus-interface is **"pnp"**.

prop-encoded-array: an array of bytes containing PnP Packages encoded as with **encode-bytes**. Format of the data resource structure is defined to be the PnP Data Resource Format(Section 6. on page 18). The adapter data format would be defined in an adapter binding.

"pnp-id" **S**

prop-name, is a property of device whose bus-interface is **"pnp"**.

prop-encoded-array: a string, as encoded with **encode-string**, of the header string from the PnP data resource record. Header format of the PnP Data Resource Structure (See "Header" on page 19.) is defined to be $V_1V_2V_3P_1P_2P_3P_4$SSSSSSSS where:

$V_1V_2V_3$	Vendor ID; string of upper case ASCII characters
$P_1P_2P_3P_4$	Product Number; string of lower case hexadecimal characters

SSSSSSSS	Serial Number; string of lower case hexadecimal characters

Conversion from numeric representation for $V_1V_2V_3$ text representation *shall* use the upper case forms of the ASCII characters in the range A..F, eliminating trailing blanks.

Conversion from numeric representation for $P_1P_2P_3P_4$ and SSSSSSSS text representation *shall* use the lower case forms of the hexadecimal digits in the range a..f, suppressing leading zeroes.

4.2. Bus-specific User Interface Commands

An Open Firmware-compliant User Interface on a platform with an ISA bus should implement the following ISA-specific user interface commands.

probe-isa (--)

The probe-isa word should probe the legacy ISA adapter cards first. The reason is that legacy ISA devices are the most inflexible with regards to resource assignments, generally not programmable and cannot be disabled; thus, the resource assignment must be satisfied first. PnP ISA adapter cards and EISA adapter cards can be probed in any order, depending on the implementation.

In addition, if there is a PCI bus in the platform to which the ISA bus is subordinate, **probe-pci** *shall* call **probe-isa** automatically and the ISA bus *shall* be probed before the PCI bus.

5. Encapsulated Drivers

This section describes a mechanism which allows the encapsulation of run-time drivers within the standard Open Firmware expansion ROM ISA and EISA adapters.

The FCode contained with a card's expansion ROM provides for Open Firmware drivers for the device. To enhance the "plug-and-play" of cards in common system platforms, it is desirable to be able to include run-time drivers within this expansion ROM, thus eliminating the extra step of installing drivers onto the OS boot device.

The information about run-time drivers is encoded as additional standard properties within the device tree. These properties are created by the FCode probe code of the plug-in card, and are used by the OS to locate and load the appropriate driver. Two new properties are defined; they differ as to how the location of the run-time driver is defined.

"driver,..." format **S**

This property, encoded as with **encode-bytes**, contains the run-time driver.

This format is used when the run-time driver is contained within the FCode image, itself. The value of the property is the encapsulated driver; the *prop-addr*, *prop-len* reported by the various **"get-property"** FCodes and/or getprop Client interface calls represent the location and size of the driver within the device tree's data space. I.e., **decode-bytes** could be used to copy the driver into the desired run-time location.

"driver-reg,..." format **S**

This property, encoded as with the **"reg"** standard property, contains a relative pointer to the run-time driver.

This format is used when the driver is not directly contained within the FCode image, but rather, is located in some other portion of the Expansion ROM. The value is encoded in a **"reg"** format, where the address is relative to the expansion ROM's base address. This format conserves device tree (and, FCode) space, but requires the OS to perform the actions of mapping in the Expansion ROM, using the information supplied by this property and the address of the Expansion ROM, and copying the driver, itself.

The **"fcode-rom-offset"** property facilitates the generation of this property within the context of the FCode's image. The driver can be located relative to the ROM image that contains the FCode (but, does not have to be within the FCode, itself) without regard to the location of that ROM image relative to others within the same expansion ROM. Therefore, "self-relocating" images containing encapsulated drivers can be created that can be concatenated with other images without altering any data within an image (except, of course, for the indicator to properly indicate the last image).

5.1. Naming Conventions

The complete property name for these encapsulated drivers is chosen to allow multiple drivers to coexist within the expansion ROM. An OS will locate its driver by an exact match of its property name among any such "driver," ("driver-reg,") properties contained within the device tree for this device. The formats of the complete names are:

> "driver,OS-vendor,OS-type,Instruction-set" "driver-reg,OS-vendor,OS-type,Instruction-set"

The OS-vendor component is as defined for device-names; i.e., organizational unique identifier (e.g., stock symbol). The OS-type & Instruction-set components are as defined by the OS-vendor. An example would be:

> "driver-reg,AAPL,MacOS,PowerPC"

6. Data Resource Information

For legacy ISA and EISA adapter cards, the resource allocation information *shall* be provided by the user via firmware and/or configuration utilities that can be platform specific. This information could be stored in the platform's NVRAM. The Open Firmware Working Group recommends that the format of this data be based on the PnP ISA resource data structures defined for PnP ISA adapter cards. For a PnP ISA adapter card, the resource information can be obtained directly from the adapter card.

Here is an example data flow diagram showing how the information could be initialized and maintained:

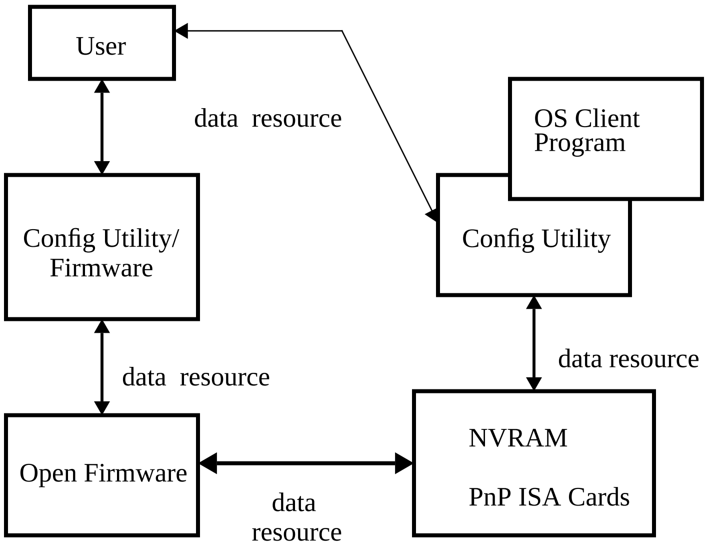

FIGURE 1 Example of data flow diagram for 'legacy' ISA Card information

The user must first run the configuration utility to describe the hardware configuration of the platform, in particular, the ISA and EISA plug-in cards. The configuration utility is responsible for warning the user of unsolvable resource conflict, based on the information that the user entered. For example, the user cannot have two primary IDE controllers. Of course, the inaccuracy of the information for ISA devices really cannot be detected by software.

The configuration utility updates the NVRAM with the provided information via the Open Firmware.

After reset, the Open Firmware recreates the comprehensive device tree, making use of the NVRAM data. It is also responsible for resolving resource allocation conflicts among programmable devices (PCI, EISA and PnP ISA). When a programmable device cannot be enabled due to resource conflicts, the **"status"** property value generated for that device should be failed.

Note: Refer to device bindings for the resource data.

6.1. Format of Data Resource Information

The location in NVRAM or other non-volatile storage where the data information is stored and the methods to read and write to it are to be determined by the platform bindings.

Open Firmware retrieves the resource information from NVRAM for ISA and EISA devices. In addition, the Open Firmware probes the PnP ISA devices for their resource information.

Regardless of the source of information, they all have the same format, as defined below. The information fully describes all resource requirements of a PnP ISA, legacy ISA and EISA cards as well as resource programmability and interdependencies. The EISA cards, in addition, contains additional information as to how to program the cards.

Each device has a header followed by a number of resource records and ended with the end tag record. The general format of data resource information for a device is:

0. Header
1. Plug and Play version number
2. ID String
3. Logical Device ID
    a. Compatible Device ID, if any, for this logical device
    b. Resource data to match what the logical device uses (IRQ, memory, I/O, DMA), of which the order for different types is unimportant, but is important for same type.
    c. Dependent functions, if any
        i. Start dependent function
        ii. Resource data
        iii. Repeat i and ii for each set of dependencies
        iv. End dependent function
4. Repeat 3 for each logical device
5. End tag

6.2. Header

The header is also known as the Serial Identifier in the PnP ISA Spec [7]. It is also known as the Device Product Identifier in the PnP BIOS Spec [6]. The first 4 bytes of the header is known as the Compressed ID in the EISA Spec[4]. The header is defined as below:

byte	definition	
0	bit 7	0
	bits[6:2]	1st char of vendor id
	bits[1:0]	2nd char of vendor id bits[4:3]
1	bits[7:5]	2nd char of vendor id bits[2:0]
	bits[4:0]	3rd char of vendor id
2	bits[7:4]	1st hex digit of product number

	bits[3:0]	2nd hex digit of product number
3	bits[7:4]	3rd hex digit of product number
	bits[3:0]	hex digit of revision level
4		serial number bits[7:0]
5		serial number bits[15:8]
6		serial number bits[23:16]
7		serial number bits[31:24]
8		checksum

The values above are in a ’little-endian’ format.

The Vendor ID is an EISA manufacturer identifier in 5-bit compressed ASCII, where “00001”=”A”, ..., “11010”=”Z”. This field is assigned to each manufacturer by the EISA administrative agent. It is the vendor's responsibility to assign a unique product number and revision level for their products. The 32-bit serial number differentiates between multiple PnP ISA cards with the same Vendor ID when they are plugged into one system (used as part of the first component of the `"reg"` property of PnP Adapters). If this feature is not supported for a PnP Card, then this field should be set to 0xFFFFFFFF.

Note: The last nibble of byte 3 is defined as a revision level. Current industry practice uses the nibble as part of the product number.

The checksum is computed as follow:

b7 b6 b5 b4 b3 b2 b1 b0 = 0x6A

y b7 b6 b5 b4 b3 b2 b1

z y b7 b6 b5 b4 b3 b2

...

where y = x0 xor x1 xor byte 0 bit 0

z = x1 xor x2 xor byte 0 bit 1

and the process repeats for each bit of the first 8 bytes of the header. After 64 iterations, x7' x6' x5' x4' x3' x2' x1' x0' is the checksum.

6.3. Resource Records

Following the header is a collection of resource records for the device. The resource records have been extended from the PnP data format in order to support the EISA Bus(“DMA Format Record” on page 23). There are two basic kinds of records, small and large.

Small records have the following format:

byte	**definition**	
0	bit 7	0 = small
	bits[6:3]	small record type
	bits[2:0]	number of bytes following byte 0
1-n	actual information	

Large records have the following format:

byte	definition	
0	bit 7	1 = large
	bits[6:0]	large record type
1	number of bytes following byte 2, bits[7:0]	
2	number of bytes following byte 2, bits[15:8]	
3-n	actual information	

Small record types are:

1	Plug and Play version number
2	logical device id
3	compatible device id
4	IRQ format
5	DMA format
6	start dependent function
7	end dependent function
8	I/O port descriptor
9	fixed location 10-bit I/O port descriptor
0xa-0xd	reserved
0xe	vendor defined
0xf	end tag

Large record types are:

1	24-bit memory range descriptor
2	ANSI id string
3	Unicode id string
4	vendor defined
5	32-bit memory range descriptor
6	32-bit fixed memory range descriptor
7-0x7f	reserved

6.3.1. Plug and Play Version Number Record

The Plug and Play version number identifies the version of the PnP specification with which the card is compatible. For legacy and EISA cards, this record is not required.

byte	definition

0		0x0a, length = 1
1		PnP version number in packed BCD
	bits[7:4]	major version
	bits[3:0]	minor version
2		vendor specific version number

6.3.2. ANSI Identifier String Record

Each adapter card is required to have an ANSI identifier string. The **`"description"`** property is derived from this string.

byte	definition
0	0x82
1	identifier string length, lower 8 bits
2	identifier string length, upper 8 bits
3-n	identifier string, not zero terminated

6.3.3. Unicode Identifier String Record

byte	definition
0	0x83
1	length of string + 2, lower 8 bits
2	length of string + 2, upper 8 bits
3	country identifier, lower 8 bits
4	country identifier, upper 8 bits
5-n	identifier string

6.3.4. Logical Device ID Record

The logical device id provides a mechanism for uniquely identifying multiple logical devices embedded in a single card. This identifier may be used to select the appropriate device driver.

The first entry of the "compatible" property comes from this logical device id record.

byte		definition
0		0x15 or 0x16, length = 5 or 6
1	bits[6:2]	1st char of device id
	bits[1:0]	2nd char of device id bits[4:3]
2	bits[7:5]	2nd char of device id bits[2:0]
	bits[4:0]	3rd char of device id
3	bits[7:4]	1st hex digit of function number
	bits[3:0]	2nd hex digit of function number
4	bits[7:4]	3rd hex digit of function number

	bits[3:0]	hex digit of revision level
5	bits[7:1]	if set, indicate commands supported per logical device for registers in the range of 31 to 37 respectively
	bit[0]	if set, indicates this logical device is capable of participating in the boot process. Otherwise, the Open Firmware does not activate this device. Thus this bit should be 0 for redundant adapter cards.
6	bits[7:0]	if set, indicate commands supported per logical device for registers in the range of 38 to 3f respectively

Note: The last nibble of byte 4 is defined as a revision level. Current industry practice uses the nibble as part of the function number.

6.3.5. Compatible Device ID Record

This record provides the ID of other devices with which this device is compatible. The operating system uses this information to load compatible device drivers if necessary. The Open Firmware uses this record to generate the **"compatible"** property.

The rest of the **"compatible"** property comes from this Compatible Device ID Record, in the order that they appear.

byte	**definition**
0	0x1c, length = 4
1-4	compatible device id in the same format as the logical device id bytes 1-4

The “**compatible**” property assumes the value of the ASCII representation of the compatible device id.

6.3.6. IRQ Format Record

This record indicates that the device uses an IRQ and supplies a mask with bits set indicating the IRQ levels implemented in this device. This record is repeated for each separate IRQ required.

byte		**definition**
0		0x22 or 0x23, length = 2 or 3
1	bits[7:0]	representing IRQ[7:0]
2	bits[7:0]	representing IRQ[15:8]
3	bits[7:4]	reserved, must be 0
	bit[3]	active low level sensitive (EISA)
	bit[2]	active high level sensitive
	bit[1]	high to low edge sensitive
	bit[0]	low to high edge sensitive (ISA)

6.3.7. DMA Format Record

This record indicates that the device uses a DMA channel and supplies a mask with bits set indicating the channels actually implemented in this device. This record is repeated for each separate DMA channel required.

byte		**definition**
0		0x2a or 0x2d, length = 2 or 5
1	bits[7:0]	representing DMA channel[7:0]
2	bit[7]	reserved, must be 0
	bits[6:5]	DMA channel speed supported
		00 compatibility mode
		01 type A DMA
		10 type B DMA
		11 type F DMA
	bit[4]	if set, DMA may execute in count by word mode
	bit[3]	if set, DMA may execute in count by byte mode
	bit[2]	if set, logical device is a bus mastering device
	bits[1:0]	DMA transfer type
		00 8-bit only
		01 8- and 16-bit
		10 16-bit only
		11 reserved
3		extended dma channel speed supported
	bit[7]	1 = ok to use extended bytes
	bits[6:0]	0 compatibility mode
		1 type A DMA
		2 type B DMA
		3 type F DMA
		4 type C DMA
4		extended DMA count type: 8, 16, 32
5		extended DMA transfer type: 8, 16, 32

Bytes 3-5 in the record are added to support EISA function. PnP ISA cards will have only 0x2a in byte 0. The additional bytes have been added to support extended EISA function which is capable of 32-bit DMA and type C DMA.

6.3.8. I/O Port Descriptor Record

byte		**definition**
0		0x47, length = 7
1	bits[7:1]	reserved, must be 0
	bit[0]	if set, the logical device decodes the full 16-bit ISA address
2		minimum I/O base address, bits[7:0]

3		minimum I/O base address, bits[15:8]
4		maximum I/O base address, bits[7:0]
5		maximum I/O base address, bits[15:8]
6		I/O base address alignment
7		number of contiguous I/O ports requested

6.3.9. Fixed Location 10-bit I/O Port Descriptor Record

byte		definition
0		0x4b
1		I/O base address, bits[7:0]
2	bits[1:0]	I/O base address, bits[9:8]
3		number of contiguous I/O ports requested

6.3.10. 24-bit Memory Range Descriptor Record

Mixing of 24-bit and 32-bit memory descriptors is not allowed.

byte		definition
0		0x81
1		9, least significant byte of record length
2		0, most significant byte of record length
3	bit[7]	reserved, must be 0
	bit[6]	memory is an expansion ROM
	bit[5]	memory is shadowable
	bit[4:3]	memory control
		00 8-bit memory only
		01 16-bit memory only
		10 8- and 16 supported
		11 reserved
	bit[2]	decode support type
		0 decode supports range length
		1 decode supports high address
	bit[1]	cache support type
		0 non-cacheable
		1 read cacheable, write-through
	bit[0]	write status
		0 ROM
		1 writeable

4	minimum base memory address, bits[15:8]
	memory base address [7:0] are assumed to be 0
5	minimum base memory address, bits[23:16]
6	maximum base memory address, bits[15:8]
7	maximum base memory address, bits[23:16]
8	base alignment, bits[15:8]
9	base alignment, bits[23:16]
10	length of memory range in 256 byte blocks, lower 8 bits
11	length of memory range in 256 byte blocks, upper 8 bits

6.3.11. 32-bit Memory Range Descriptor Record

byte		**definition**
0		0x85
1		7, least significant byte of record length
2		0, most significant byte of record length
3	bit[7]	reserved, must be 0
	bit[6]	memory is an expansion ROM
	bit[5]	memory is shadowable
	bit[4:3]	memory control
		00 8-bit memory only
		01 16-bit memory only
		10 8- and 16-bit supported
		11 reserved
	bit[2]	decode support type
		0 decode supports range length
		1 decode supports high address
	bit[1]	cache support type
		0 non-cacheable
		1 read cacheable, write-through
	bit[0]	write status
		0 ROM
		1 writeable
4		minimum base memory address, bits[7:0]
5		minimum base memory address, bits[15:8]
6		minimum base memory address, bits[23:16]

7	minimum base memory address, bits[31:24]
8	maximum base memory address, bits[7:0]
9	maximum base memory address, bits[15:8]
10	maximum base memory address, bits[23:16]
11	maximum base memory address, bits[31:24]
12	base alignment, bits[7:0]
13	base alignment, bits[15:8]
14	base alignment, bits[23:16]
15	base alignment, bits[31:24]
16	length of memory range, bits[7:0]
17	length of memory range, bits[15:8]
18	length of memory range, bits[23:16]
19	length of memory range, bits[31:24]

6.3.12. 32-bit Fixed Location Memory Range Descriptor Record

byte		**definition**
0		0x86
1		9, least significant byte of record length
2		0, most significant byte of record length
3	bit[7]	reserved, must be 0
	bit[6]	memory is an expansion ROM
	bit[5]	memory is shadowable
	bit[4:3]	memory control
		00 8-bit memory only
		01 16-bit memory only
		10 8- and 16-bit supported
		11 reserved
	bit[2]	decode support type
		0 decode supports range length
		1 decode supports high address
	bit[1]	cache support type
		0 non-cacheable
		1 read cacheable, write-through
	bit[0]	write status
		0 ROM

	1 writeable
4	base memory address, bits[7:0]
5	base memory address, bits[15:8]
6	base memory address, bits[23:16]
7	base memory address, bits[31:24]
8	length of memory range, bits[7:0]
9	length of memory range, bits[15:8]
10	length of memory range, bits[23:16]
11	length of memory range, bits[31:24]

6.3.13. Start Dependent Function Record

The resources required by a logical device may have interdependencies that need to be expressed to allow software to make resource allocation decisions about the device. Dependent functions are used to express these interdependencies. For example, serial I/O ports are linked with a particular IRQ.

byte	**definition**
0	0x30 or 0x31, length = 0 or 1
1	priority byte
	0 highest priority and preferred configuration
	1 lower priority but acceptable configuration
	2 sub-optimal configuration
	3-ff reserved

6.3.14. End Dependent Function Record

Dependent Functions are not nestable. Therefore, only one End Dependent Function record is allowed per logical device.

byte	**definition**
0	0x38, length = 0

6.3.15. Small Vendor Defined Record

byte	**definition**
0	0x71-0x77, length = 1-7
1-7	vendor defined

6.3.16. Large Vendor Defined Record

byte	**definition**
0	0x84
1	length of data, lower 8 bits
2	length of data, upper 8 bits

3-n	vendor defined data

6.3.17. End Tag Record

This record identifies the end of resource information for the adapter card.

byte	definition
0	0x79, length = 1 checksum covering all resource data after the serial identifier. This checksum is generated such that adding it to the sum of all data bytes will produce a zero sum. If the field is 0, the resource data is treated as if it checksummed correctly.

-- END OF DOCUMENT --