4. EISA System Configuration

EISA provides a mechanism for automatic configuration of expansion boards and the system board. The EISA configuration mechanism consists of the following components:

- A software utility to configure the system board and expansion boards
- A software interface to the configuration utility that Configuration File Extensions can use to control and customize the configuration process
- Configuration files that accompany the software utility
- Configuration files that accompany the system board and expansion boards
- Nonvolatile memory for storing configuration information
- A mechanism to save and restore a backup copy of the system configuration information
- BIOS routines to read and write contents of nonvolatile memory
- Automatic detection and initialization of expansion boards by the system ROM power-up routine
- 1024-byte I/O address space for each EISA expansion board (slot-specific)

Expansion board manufacturers include a configuration file (also referred to as a CFG file) with each EISA expansion board, and optionally, with switch-programmable ISA products. The configuration utility, which is provided by the system manufacturer, uses the information contained in the configuration files to determine a conflict-free configuration of the system resources. The configuration utility stores the configuration and initialization information into nonvolatile memory and saves a backup copy on diskette. The system ROM power-up routines use the initialization information to initialize the system during power-up, and device drivers use the configuration information to configure the expansion boards during operation.

4.1 Devices Supported by Automatic Configuration

EISA systems provide automatic configuration for expansion boards plugged into the expansion bus, peripheral devices built into the system board, and software drivers that use system resources, such as an expanded memory (LIM EMS) emulator. The following information provides an overview of the mechanism used for automatic configuration of the devices.

4.1.1 Expansion Boards

Expansion boards install into EISA and ISA bus connectors. Each bus connector is referred to as a slot. The bus connectors are numbered sequentially from 1 to "n" (with 15 as a maximum "n"). For example, an EISA system with 7 bus connectors has slots numbered from slot 1 to slot 7.

4.1.1.1 EISA Expansion Boards

Each EISA slot has I/O address decoding hardware that provides the installed expansion board with a unique, 1024 byte, slot-specific I/O address space. EISA expansion boards use the slot-specific I/O address space for I/O registers (i.e., configuration and operational registers). The EISA system ROM uses configuration information from nonvolatile memory to initialize the configuration registers during power-up.

Refer to the section entitled Expansion Board Address Decoding and the one entitled System Board Slot-Specific I/O, of this specification for detailed information on the slot-specific I/O ranges.

An EISA expansion board must contain a readable product ID and must support the expansion board control bits ENABLE and IOCHKERR. Refer to the section entitled Expansion Board Control Bits and the one entitled EISA Product Identifier of this specification for detailed information.

4.1.1.2 ISA Expansion Boards

The EISA configuration utility also aids in configuration of ISA expansion boards that provide a configuration file. The utility uses the information from the configuration file to determine the correct switch and jumper settings and I/O port initializations for ISA expansion boards. The configuration utility displays the proper switch and jumper settings to the user.

ISA initialization and operational registers must occupy the ISA compatible expansion board I/O space (100h-3FFh). ISA systems do not support the EISA slot-specific I/O ranges. The EISA system ROM power-up routines automatically initialize the ISA registers that are specified in the configuration file.

4.1.2 System Board

Peripherals integrated onto the system board require automatic configuration support similar to expansion board peripherals. System board peripherals can be designed to use EISA slot-specific I/O ranges and the ISA system board I/O range.

4.1.2.1 System Board Peripherals That Use Slot-Specific I/O Space

A system board peripheral that uses slot-specific I/O is functionally similar to an expansion bus peripheral, but it is integrated onto the system board rather than installed in a bus connector. EISA automatic configuration treats the system board peripheral as an expansion board peripheral, except that it is referenced as an "embedded device."

4.1.2.2 System Board Peripherals That Use System Board I/O Space

System board peripherals that use ISA expansion board I/O space (100h-3FFh) can be treated as "virtual devices." The configuration utility stores the configuration and initialization information for "virtual devices" in nonvolatile memory during configuration. The system ROM automatically initializes the virtual device during power-up.

4.1.3 Software Drivers That Require System Resources

Software drivers that require system resources (i.e., memory allocation) are also treated as "virtual devices." Twoexamples include, a software driver that emulates expanded memory (LIM EMS) requires memory allocation for the page frame, or a software driver that requires a buffer which memory allocation to store data during a data transfer.

4.2 Configuration Utility

The EISA system manufacturer is responsible for supplying a configuration utility. The configuration utility uses configuration files to resolve conflicts in assignment of system resources such as interrupt levels and DMA channels. The configuration utility also extracts initialization information that is used for system board and expansion board initialization. The information is stored in nonvolatile memory and a backup is saved on diskette.

The type of nonvolatile memory and method of writing the data is not included in the EISA standard and is determined by the system manufacturer. The system manufacturer also provides BIOS routines to initialize the expansion boards with the information stored in nonvolatile memory. The BIOS routines also read configuration information from nonvolatile memory for device drivers and other system software.

All references to the configuration utility included in this specification refer to the configuration utility available from Micro Computer Systems, Inc. of Irving, Texas.

The configuration utility is used to configure an EISA computer. The configuration process provides the following functions:

- Read and parse configuration files
- Automatically allocate resources to create a conflict-free system
- Saves, configuration to diskette, which allows a common configuration to be ported to other similarly-configured machines
- Write configuration information into nonvolatile memory

System board and expansion board products can include CFG File Extensions that extend the capabilities of the configuration utility and customize the configuration process. For example, a CFG File Extension can be used to detect options installed on an expansion board, to accept and process user input (other than menu selections), or to write configuration information to non-EISA nonvolatile memory.

4.3 Configuration Files

The configuration files contain the expansion board ID, system resource requirements and initialization information for system board or expansion board devices.

The initialization information provides data for power-up initialization. The configuration utility stores the appropriate I/O port initialization information in nonvolatile memory. The system ROM reads the information from nonvolatile memory during power-up and initializes the I/O ports.

System resource requirements include memory, I/O ports, interrupts, and DMA channels. The configuration utility verifies that system resource selections do not conflict with resource allocations already selected for other devices. The configuration utility then stores the appropriate system resource information in nonvolatile memory. The system ROM reads the information from nonvolatile memory during power-up and initializes the devices and expansion boards.

A device driver can use a BIOS routine Call to determine the proper expansion board initialization and to determine the system resource configuration.

A software driver can use the BIOS routines to identify the functions of expansion devices and the resources allocated to the devices. The driver can determine the contents of each slot, its functions, the initialization information, and the system resources allocated for each function.

4.3.1 Configuration File Extensions

System board and expansion board products can include CFG File Extensions (also called overlay files,) that customize the configuration process. [2]

CFG File Extensions can be used to determine the installed hardware by reading from the hardware registers or other means. For example, the overlay may detect the presence of floating point coprocessors, disk drives (and determine drive type), or total amount of memory installed on a memory expansion board.

The overlay can control the configuration of a system board or expansion board. It can access the hardware, provide the user interface and process the user-specified configuration selections. Or the overlay can provide a limited set of configuration services and rely on the configuration utility to perform its normal functions.

Interaction between the configuration utility and the CFG File Extension is specific to the utility. Therefore, the CFG file extension must be written such that it uses the calling conventions and interface handling routines recognized by the utility.

---

2 A specification for CFG File Extensions is available from Micro Computer Systems, Inc. of Irving, TX. It describes overlays specific to the utility that allow system manufacturers to customize the configuration process.

4.3.2 Expansion Board Identifier (Product ID)

The expansion board identifier (product ID) is a unique product identification code that can be read by the system ROM or other software to identify or locate an expansion board. Information that can be combined in an expansion board ID includes the manufacturer's ID, product number and revision level. The exact method for selecting an expansion board ID is described in the section entitled, Product Identifier (ID).

EISA expansion boards must contain a readable product ID. The power-up routines use the ID to determine the slot in which the expansion board is installed. The expansion board is then programmed by the system ROM with the configuration parameters that are stored in nonvolatile memory.

ISA expansion boards should have a product ID provided in the configuration file. The product ID may or may not be readable. An expansion board ID is recommended for ISA expansion boards since it can be stored in nonvolatile memory with other manufacturer-specified information, such as the initialization information and resource requirements. The data stored in nonvolatile memory can then be accessed by software drivers to determine the expansion board configuration.

4.3.3 I/O Port Initialization Information

The configuration file contains I/O port initialization information necessary to configure an expansion board. The I/O port initialization information specifies the I/O port addresses and values for each alternative configuration.

4.3.4 System Resource Requests

Devices that require system resources include the resource request in the configuration file. The CFG file can contain requests for the following system resources:

- Memory--the amount of memory supported, starting address, whether it is writable or cacheable, and initialization parameters required
- I/O ports-port addresses and initialization parameters required
- Interrupts--interrupts supported, whether the interrupt can be shared, whether it is edge- or level-sensitive, and any initialization parameters required
- DMA channels--the choice of DMA channels, whether the channel can be shared, the channel's data size, the channel's cycle timing, and any initialization parameters required

4.4 Configuration File Filenames

The filename of an EISA or ISA configuration file consists of an exclamation point followed by the product ID and a filename extension, CFG. The exclamation point must be included as the initial character of all CFG filenames. Valid filenames have the following format:

!ACE1234.CFG !XYZ5678.CFG !ABC0000.CFG

The filename convention is the same for a system board, expansion board, embedded device or virtual device. For example, an expansion board with a product ID of ACE0101 has a configuration file named !ACE0101.CFG.

The expansion board manufacturer should ensure that the configuration file filename is updated to reflect revisions to the expansion device. For example, a product with an ID of ACE101 may have a configuration file named !ACE0101.CFG. A subsequent revision of the product would have an ID of ACE102. Therefore, the configuration file should be named !ACE0102.CFG. This ensures that the appropriate CFG file is loaded for the device.

The configuration utility includes a mechanism to manage duplicate IDs. For example, the configuration files for two expansion boards with ID ACE1234 installed in the same system could be renamed when copied to the configuration diskette: the first configuration file detected is copied to !ACE1234.CFG the second configuration file detected is copied and renamed from !ACE1234.CFG to 1ACE1234.CFG. The next one is renamed to 2ACE1234.CFG.

4.5 The Configuration Procedure

EISA system configuration requires the following hardware and software:

- An EISA computer system
- The EISA system board configuration file
- The configuration utility
- Optionally, EISA expansion boards and configuration files
- Optionally, ISA expansion boards and configuration files
- EISA or ISA Configuration File Extensions (where needed)

The following procedure describes an example configuration process for an EISA system with EISA and ISA expansion boards. This example configuration requires a bootable EISA computer with a display, keyboard and floppy diskette attached.

Start the procedure with the computer power switch "OFF."

Install EISA boards in the computer to allow "automatic detection" of the devices.

Insert the configuration utility diskette.

Turn the computer power switch "ON," booting from the configuration utility diskette.

Use the configuration utility commands to copy each configuration file and CFG File Extension to the configuration utility diskette. The configuration utility automatically renames the CFG files from expansion boards with duplicate IDs.

Let the configuration utility automatically select a conflict-free configuration. The user may override the automatic selections.

Set the switches on ISA expansion boards to the positions indicated by the configuration utility.

Turn the computer power switch "OFF" and install the ISA expansion boards in the expansion slots as indicated by the configuration utility.

Remove the configuration utility diskette.

Turn the computer power switch "ON" to the configured system, booting from the normal boot device (for example, the fixed disk).

Incorporate the software options into the operating system startup files as indicated by the configuration utility. The startup files can execute programs that require command line parameters (for example, /s, /g). The configuration utility indicates the proper parameters. For example, the configuration utility lists entries for the CONFIG.SYS and AUTOEXEC.BAT files of an MS-DOS operating system.

Reboot the system.

4.5.1 Configuration File Syntax

The following sections specify the syntax conventions used in this document and for configuration files.

4.5.2 Symbol Conventions

The configuration file syntax uses the following special symbols.

{} Empty braces indicate a null value.

\ The backslash within a text field identifies an embedded character. Embedded characters include the \t for up to an 8-space tab (or to the next tab stop), \n for a line feed, \" for quotation marks, and \\ for a backslash.

\t Embeds a tab within text.
Tab stops are: 1, 9, 17, 25, 33, ...

\n Replaces \n with a carriage return, line feed. The configuration utility automatically wraps text at the right margin to the next line (word wrap) for free-form text fields.

\" Embeds a quotation mark character within text that has quotation marks delimiting the entire field.

\\ Embeds a \ (backslash) character within text.

" " Information enclosed in quotation marks is free-form ASCII text. The text can contain embedded characters, including tabs and line feeds. Quotation marks can be used within a text field by entering a \".

- The dash (hyphen) separates the minimum and maximum values in a range.

| The vertical bar is equivalent to an OR statement. Items separated by a vertical bar (|) indicate that only one of the items is allowed.

space A blank space is equivalent to an AND statement. Information separated by a space indicates all items are included. The space serves to group items of an inclusive list. For example, the statement *(x and y) or (y and z)* is denoted: x y | y z.

; The semicolon precedes comments in the configuration file. The configuration utility ignores text that follows the semicolon (up to the end of the line).

4.5.3 Numerical Value Conventions

Numerical values within a configuration file must adhere to the following conventions:

- All numerical values are assumed to be decimal unless otherwise indicated. Decimal values can include a trailing d or D.
- Binary port values must be written with the MSBit on the left and may include a trailing b or B. A "1" or "0" in a bit position indicates the bit value.
- Decimal fractions are not allowed.
- Address values may be expressed as megabyte (indicated by an M suffix), kilobyte (indicated by a K suffix), or byte (no suffix). Values for megabytes or kilobytes must be given in decimal units but cannot include a trailing d or D. For example, two kilobytes can be represented either by 2K or 2048d, but not by 2Kd.
- Hexadecimal values must include a trailing h or H. In the case of hexadecimal values that begin with an alpha character, such as C68h, the value must also have a leading 0 (zero). And when noting slot-specific EISA port addresses, the value must be preceded by a 0Z (zero Z). For example, slot-specific port C80h would be represented as 0ZC80h.
- An x in a binary value indicates the bit is not used or a don't care.
- An r in a binary value indicates the hardware register must be read and the actual bit value masked into the "r" bit position.
- An n in a binary value for a tripole jumper indicates the jumper is not installed.

4.5.4 Keyword and Field Specification Conventions

Within this document the following conventions are followed when describing the configuration file.

Value	indicates that an ASCII string or number is required in this field; any numerical unit format can be entered for a value.
{}	may be selected to indicate that none of the resource selections are used.
List	indicates that a set of resource selections can be included in the field, each delimited vertical bar (|, logical OR).
Rangelist	indicates that a set of resource address range selections or lists can be included in the field, each delimited by a vertical bar (|, logical OR).
Valuelist	indicates that a set of values can be included in the field, each delimited by a vertical bar (|, logical OR).

textlist	indicates that a set of ASCII values can be included in the field; the textlist must be contained within double quotes, with each string delimited by a space.
Switchlist	indicates that a set of switches can be included in the field, each delimited by a space. A switchlist can also comprise a range of switches.
Jumperlist	indicates that a set of jumpers can be included in the field, each delimited by a space. A jumperlist can also comprise a range of jumpers.
Bitlist	indicates that a set of bit positions can be included in the field. A bitlist can also comprise a range of bits.
parameterlist	indicates that a set of ASCII values can be included in the description field of a software statement; the parameterlist must be contained within double quotes, with each string delimited by a vertical bar (|, logical OR).
[]	Items within square brackets are optional.
CAPS	Keywords are indicated by all capital letters. For example, BOARD, ID, NAME, and COMMENTS are keywords and are indicated by all capitals.
italic	Italic text used in the syntax provides descriptive information about the indicated field. For example, names, values, lists and ranges are indicated by italic text.
ASCII text	ASCII characters 20-255h are valid for fields that require ASCII text. Null strings are allowed.
(Optional)	When used within a statement title, indicates that the statement provides additional information, but is not required in the configuration file.

4.6 Configuration File Format

A configuration file consists of a board identification block, one or more initialization information blocks, and one or more function statement blocks. The configuration file begins with a board identification block, which provides the name and ID of the board as well as slot information. The initialization information blocks include the values to initialize I/O ports and for ISA boards, information about jumper and switch settings. The function statement blocks specify the resource requirements of the functions of the board. Additionally, CFG files for system boards may include a system description block (following the board identification block), which icludes information specific to the system board.

Every configuration file must include the board identification block. The initialization information blocks and function statement blocks are optional, but must be included to utilize automatic configuration.

The configuration file has the following structure:

Board Identification Block
    Board Identification and Slot Information
[*System Description Block*]
[*Initialization Information Block*
    I/0 port requests
    Switch and jumper settings
    Software initialization information]
[*Function Statement Block*
    Configuration Selections
    [*Resource requirements*]]
        .
        .
        .
[*Function Statement Block*
    Configuration Selections
    [*Resource requirements*]]

4.6.1 Board Identification Block

Each configuration file must begin with a board identification block. Four required fields must be included in the board identification block to provide the basic ID requirements of the board; optional fields can be included to provide additional board identification information.

System boards require special configuration files and are covered in the section entitled, System Board Configuration File.

The board identification block has the following format.

```
BOARD
    ID = "7-character ID"                    ;Product ID
    NAME = "descriptive name"
    MFR = "manufacturer name"
    CATEGORY = "3-character category"
    [SLOT = ISA8|ISA16|ISA8OR16|OTHER|EISA|VIR|EMB[(n)] [,"text"]...]
    [LENGTH = value]                         ;In millimeters
    [AMPERAGE = value]                       ;5V current used, in mA
    [SKIRT = YES | NO]
    [READID = YES | NO]                      ;Readable product ID
    [BUSMASTER = value]                      ;Maximum acceptable latency (in μs)
    [IOCHECK = VALID | INVALID]
    [DISABLE = SUPPORTED | UNSUPPORTED]
    [COMMENTS = "general information"]
    [HELP = "help information"]
```

BOARD Statement (Required)

Syntax:
BOARD

The BOARD statement identifies the beginning of the Board Identification Block.

ID Statement (Required)

Syntax:
ID = *"7-character ID"*

The ID statement contains the seven-character expansion board ID. The ID is the uncompressed, ASCII representation of the product ID (see the section entitled, EISA Product Identifier, for information on compressed IDs). The seven-character ID consists of a three-character manufacturer code, a three-character hexadecimal product identifier, and a one character hexadecimal revision number. For example, the second revision of an expansion board manufactured by the ACME board company might have an uncompressed ID such as ACE0102.

NAME Statement (Required)

Syntax:
NAME = *"descriptive name"*

The NAME statement contains text that identifies the product. Part numbers and other information may also be included. The NAME text field can contain up to 90 ASCII characters.

MFR Statement (Required)

Syntax:
MFR = "*manufacturer name*"

The MFR statement contains a text field that identifies the board manufacturer. The MFR text field can contain up to 30 ASCII characters.

CATEGORY Statement (Required)

Syntax:
CATEGORY = "*3-character category*"

The CATEGORY statement contains a 3-character text field (use uppercase for consistency) that identifies the board's functional category. The configuration utility displays the CATEGORY text field (in upper case) during system configuration.

The CATEGORY statement must use one of the following categories:

COM = communications device
KEY = keyboard
MEM = memory board
MFC = multifunction board
MSD = mass storage device
NET = network board
NPX = numeric coprocessor
OSE = operating system/environment
OTH = other
PAR = parallel port
PTR = pointing device
SYS = system board
VID = video board

SLOT Statement (Optional)

Syntax:
SLOT = *value* [,"text"]...

The SLOT statement identifies the type of slot in which the expansion board can be installed. Options that can be entered in the *value* field include: ISA8, ISA16, ISA8OR16, EISA, VIR, EMB(n), OTHER, and at a text string If the SLOT statement is omitted, the default is ISA16. For expansion devices that occupy physical slots (ISA8, ISA16, ISA8OR16, EISA, and OTHER), the value entered in the SLOT field is the actual size of the board's card edge. For example, an expansion board with an 8-bit card edge is set to SLOT = ISA8, an expansion board with a 16-bit card edge is set to SLOT = ISA16, and so on. ISA8OR16 is provided for 16-bit expansion boards that can also operate in an 8-bit slot.

A text string can be included with the slot statement following the value field. More than one text string can be included. Each text string must be enclosed in double quotes. The text is typically used to describe the slot. For example: SLOT=EISA,"MEMORY" could be used to describe an EISA slot reserved for a memory expansion board.

ISA8

This entry specifies an 8-bit ISA expansion board (fits in any slot of correct length).

ISA16
This entry specifies a 16-bit ISA expansion board (fits in an EISA or 16-bit slot of correct length).

ISA8OR16
This entry specifies an ISA expansion board configurable as 8- or 16-bit (fits in any EISA or ISA slot of correct length).

EISA
This entry indicates an EISA expansion board that requires a correct length EISA slot (fits in EISA slot only).

EMB[(n)]
This entry indicates a system board peripheral that uses slot-specific I/O space (embedded device). The slot-specific I/O range used determines the "n." The configuration utility searches for the device by checking the embedded device IDs if the "n" is omitted. The embedded devices are numbered sequentially from "y+1" (y equals the number of expansion bus connectors) to 15.

The system board configuration registers use the slot-specific I/O space, slot number 0, and are addressed as embedded device 0, EMB(0).

VIR
This entry indicates a virtual device. Virtual devices do not have slot-specific I/O or a readable ID. This entry is included for virtual devices so the configuration utility can perform conflict resolution and drivers can obtain configuration information regarding the devices. Any peripheral, device or software that needs a configuration file and is not covered by the other device types can be specified as a virtual device. Virtual devices are assigned numbers from 16 to a maximum of 64.

OTHER
This entry identifies a vendor-specific expansion slot.

LENGTH Statement (Optional)

Syntax:
[LENGTH = *value*]

The LENGTH statement specifies the length of the board in millimeters (a decimal integer). The LENGTH statement does not apply to embedded devices or virtual devices.

Expansion boards should include a LENGTH statement. The configuration utility cannot optimize the slot allocation if expansion boards do not specify length. If the LENGTH statement is omitted the configuration utility defaults to 330.

AMPERAGE Statement (Optional)

Syntax:
[AMPERAGE = *value*]

The AMPERAGE statement, when included in the board identification block, specifies the maximum amount of continuous 5V current (in milliamps) required by the base configuration of the expansion board. Installable options can specify additional 5V current requirements with an AMPERAGE statement in the CHOICE Statement Block (described later in this specification). The AMPERAGE statement does not apply to embedded devices or virtual devices.

Devices that require +5 volt power should include an AMPERAGE statement. The configuration utility cannot perform an accurate power usage verification if expansion boards do not specify their power requirement. If the AMPERAGE statement is omitted, the configuration utility defaults to AMPERAGE = 0.

SKIRT Statement (Optional)

Syntax:
[SKIRT = YES | NO]

The SKIRT statement indicates the presence of a drop-down skirt. (A drop-down skirt is an extended lower portion of an 8-bit expansion board that prevents installation into a 16-bit slot.) The default is NO.

READID Statement (Optional)

Syntax:
[READID = YES | NO]

READID specifies whether or not the expansion board has an ID that can be read from the EISA ID registers. The default value is NO.

BUSMASTER Statement (Optional)

Syntax:
[BUSMASTER = *value*]

The board identification block may include a BUSMASTER statement to identify the expansion board as a bus master and to specify the maximum acceptable latency. The latency *value* is a specification of the worst case acceptable time (in microseconds) from the bus master bus request to the bus grant. The configuration utility assumes an expansion board is not a bus master if the BUSMASTER statement is omitted.

IOCHECK Statement (Optional)

Syntax:
[IOCHECK = VALID | INVALID]

IOCHECK is an optional statement that indicates support of the EISA expansion board control register IOCHKERR bit. VALID indicates that the expansion board responds to reads of its IOCHKERR bit. INVALID indicates that the expansion board does not respond to reads of the IOCHKERR bit. The default is VALID.

DISABLE Statement (Optional)

Syntax:
[DISABLE = SUPPORTED | UNSUPPORTED]

DISABLE is an optional statement that indicates support of the EISA expansion board control register ENABLE bit. SUPPORTED indicates that the expansion board can be disabled by clearing the ENABLE bit. UNSUPPORTED indicates that the expansion board cannot be disabled by clearing the expansion board control register ENABLE bit. The default is SUPPORTED.

COMMENTS Statement (Optional)

Syntax:
[COMMENTS = "*general information*"]

The COMMENTS statement provides information about the expansion board. The configuration utility displays the contents of the COMMENTS text field in a window at least 40 characters wide. This COMMENTS text field can contain up to 600 ASCII characters.

HELP Statement (Optional)

Syntax:
[HELP = "*help information*"]

The HELP statement provides information about the expansion board if the user requests help during the configuration. The configuration utility displays the HELP information in a window at least 40 characters wide. The HELP text field can contain up to 600 ASCII characters.

Example Board Identification Block

The following example illustrates a board identification block for a multifunction board.

```
BOARD
    ID = "ACE0102"                          ;Revision 02
    NAME = "EISA Multifunction Board"
    MFR = "ACME Inc."
    CATEGORY = "MFC"                        ;Multifunction board
    SLOT = EISA                             :Requires EISA slot
    LENGTH = 330                            ;Full length board
    AMPERAGE = 3000                         ;3000 mA max current draw
    SKIRT = NO
    READID = YES
    COMMENTS = "The EISA Multifunction Board provides
        an asynchronous communication port,
        a parallel port, a game port and
        4 megabytes of memory. "
    HELP = "The EISA Multifunction Board supports
        full automatic configuration.
        You may want to select the expanded
        memory configuration instead of taking
        the default, which is extended memory. "
```

The SKIRT and length statements could be omitted from this board identification block, since the specified values equal the default value.

4.6.2 Initialization Information Block

The initialization information block consists of one or more of the following statement blocks:

I/O port initialization statement block

Switch configuration statement block

Jumper configuration statement block

Software initialization statement block

All expansion boards that require configuration must provide an initialization information block (IIB) in the configuration file. (A shorthand method described in the I/O Port INIT statement discussion in the section entitled INIT Statements, can be substituted for certain IIBs.)

4.6.2.1 I/O Port Initialization Statement Block

The I/O Port Initialization statement block begins with the IOPORT(i) statement. The syntax of the I/O port initialization statement block is:

```
IOPORT(i) = address                         ;I/O port address
    [SIZE = BYTE | WORD | DWORD]            ;Number of bits in I/O port
    [INITVAL = [LOC(bitlist) ] valuelist]   ;Initialization value
```

IOPORT(i) Statement (Required)

Syntax:
IOPORT(i) = *address*

The IOPORT(i) statement specifies the address of an I/O port. Each I/O port must have a separate IOPORT(i) statement with a different identifier, "i". The "i" can be any positive integer value from 1 to 32767. Resource and initialization statements use the IOPORT(i) to specify I/O port addresses.

See the "PORTVAR(j) Variable" section for an alternative method of specifying the I/O port address.

SIZE Statement (Optional)

Syntax:
[SIZE = BYTE | WORD | DWORD]

The SIZE statement specifies the number of bits in the I/O port. The default is BYTE.

INITVAL Statement (Optional)

Syntax:
[INITVAL = [LOC(*bitlist*)] *valuelist*]

The INITVAL statement specifies the source of the values written to an initialization port.

The *valuelist* portion specifies the source of each bit of a binary value. An "r" in a bit position indicates the bit value must be read from the port. An "x" in a bit position indicates the configuration utility determines the bit value based on the selected configuration. A "1" or "0" in a bit position indicates the bit is reserved and must be initialized to the specified value. The *valuelist* must be in MSBit to LSBit order.

The INITVAL statement may include the LOC(*bitlist*) string to reference individual bits. The *bitlist* contains a list or range of bit positions. The elements of the *bitlist* must be in MSBit to LSBit order. The following example illustrates valid INITVAL syntax.

```
INITVAL = 0000111100001111b          ;WORD port

INITVAL = 00001111b                  ;BYTE port

INITVAL = LOC(7-0)  001100rr         ;Byte port with "r" bits

INITVAL = LOC(7-2)  001100           ;Byte port (range)

INITVAL = LOC(7 6 1 0)  0011         ;4 bits specified
```

Example I/O Port Initialization Statement Block

The following example illustrates an I/O port initialization statement block. The two most significant bits are read from the I/O port, the next two bits are "1" and "0" respectively, and the four least significant bits are determined by the configuration utility.

```
IOPORT(1) = 3F8h          ;I/O port address
   INITVAL = rr10xxxxb    ;Bit pattern
```

4.6.2.2 Switch Configuration Statement Block

The switch configuration statement block begins with the SWITCH(i) statement. The syntax of the switch configuration statement block is:

```
SWITCH(i) = value                                   ;Number switches in set
   NAME = "switch name or description"
   STYPE = DIP | ROTARY | SLIDE                     ;Type of switch
   [VERTICAL = YES | NO]                            ;Switch orientation
   [REVERSE = YES | NO]                             ;Switch numbering scheme
   [LABEL = LOC(switchlist) textlist]               ;Switch labels
   [INITVAL = LOC(switchlist) valuelist]            ;Switch settings
   [FACTORY = LOC(switchlist) valuelist]            ;Factory setting
   [COMMENTS = "configuration comments"]
   [HELP = "configuration help information"]
```

SWITCH(i) Statement (Required)

Syntax:
SWITCH(i) = *value*

The SWITCH(i) statement specifies the number of switch positions in a set. Each set of switches must have a separate SWITCH(i) statement with a different identifier, "i". The "i" can be any positive integer value from 1 to 32767. The maximum number of switches is "16" for all switch types. *Value* indicates the number of switches in the switch block.

NAME Statement (Required)

Syntax:
NAME = "*switch name or description*"

The NAME statement contains the switch name as it is designated in the user documentation. The name can be up to 20 characters long.

STYPE Statement (Required)

Syntax:
STYPE = DIP | ROTARY | SLIDE

The STYPE statement designates the type of switch as DIP, ROTARY, or SLIDE. A DIP switch is a set of switches, each having an "ON" and "OFF" position. A ROTARY switch is a set of switches with a rotating selector that can be set to one switch position. A SLIDE switch is a set of switches arranged linearly with a slide mechanism that can be set to one switch position. All switches within the set are numbered beginning with 1.

VERTICAL Statement (Optional)

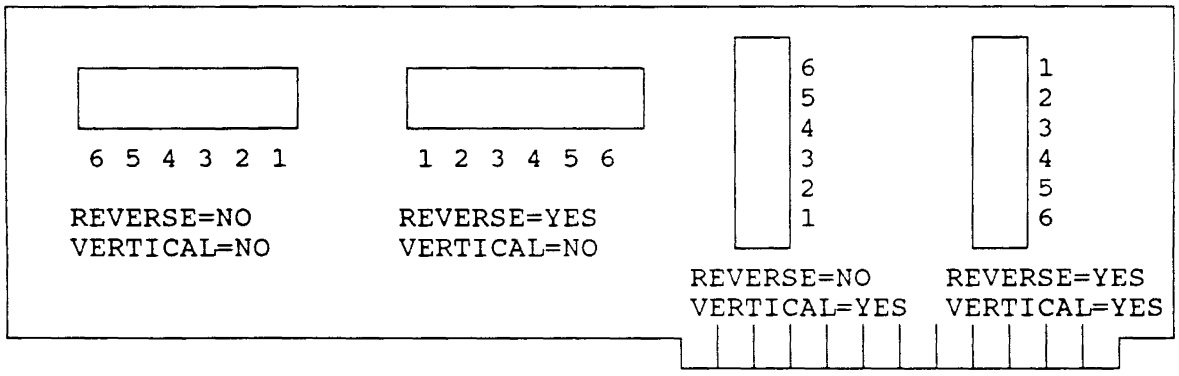
Syntax:
[VERTICAL = YES | NO]

The VERTICAL statement indicates the orientation of the switch on the expansion board. Refer to the figure below for an illustration of switch orientation. The VERTICAL statement defaults to "NO."

REVERSE Statement (Optional)

Syntax:
[REVERSE = YES | NO]

The REVERSE statement specifies the order that a DIP switch is numbered. REVERSE = YES indicates 1234..., REVERSE = NO indicates ...4321 order. Refer to the figure below for an illustration of switch numbering. The REVERSE statement defaults to "NO."



LOC(*switchlist*) *valuelist*

The switch configuration statements LABEL, INITVAL and FACTORY include the LOC(*switchlist*) *valuelist* (or *textlist*) string to reference individual switches. The *switchlist* contains a list or range of switch numbers. The elements of the *switchlist* must be in ascending order if REVERSE=YES or descending order if REVERSE=NO. A space must be included between elements as a delimiter.

The *textlist* specifies the ASCII switch name and the *valuelist* specifies the switch setting for each switch position. The *valuelist* must use the same order as the *switchlist*. A DIP switch can be set for "1" to indicate "ON," "0" to indicate "OFF," or "x" to indicate "don't care." The dip switch settings are not delimited with a space. The *valuelist* for a rotary or slide switch includes a "1" in the position number of the selected position. Zeros fill the other positions.

The following examples illustrate valid LOC(*switchlist*) *valuelist* strings:

```
REVERSE=YES
INITVAL = LOC(1 2 3 4) 0011              ;List of DIP switches

REVERSE=NO
INITVAL = LOC(4 3 2 1) 1100              ;List of DIP switches

REVERSE=YES
INITVAL = LOC(1-4) 0011                  ;Range of DIP switches

REVERSE=NO
INITVAL = LOC(4-1) 1100                  ;Range of DIP switches

REVERSE=YES
INITVAL = LOC(1 2 3 4) 00x1              ;DIP switches with a don't care

REVERSE=YES
INITVAL = LOC(1-8) 00010000 ;8-position rotary or slide switch
```

LABEL Statement (Optional)

Syntax:
[LABEL = LOC(*switchlist*) *textlist*]

The LABEL statement specifies labels for individual switches. Each label can compose up to 10 characters. If the LABEL statement is omitted, the default label is the switch number (...4321 for normal switches and 1234... for reverse switches). The following example illustrates use of the LABEL statement:

```
LABEL = LOC(4-1) "SW1-4" "SW1-3" "SW1-2" "SW1-1"
```

INITVAL Statement (Optional)

Syntax:
[INITVAL = LOC(*switchlist*) *valuelist*]

The INITVAL statement specifies the settings for factory-set switches that must not be changed. If the INITVAL statement is omitted, switch settings are determined by the configuration program or are "don't care." This statement is particularly important for switches that control undocumented options. The following example illustrates use of the INITVAL statement:

```
INITVAL = LOC(4 3 2 1) xxx0      ;DIP switch 1 may not be changed
```

FACTORY Statement (Optional)

Syntax:
[FACTORY = LOC(*switchlist*) *valuelist*]

The FACTORY statement indicates the factory settings for the switches.

COMMENTS Statement (Optional)

Syntax:
[COMMENTS = "*configuration comments*"]

The COMMENTS statement contains information to assist the user in configuring a switch. The COMMENTS text field can contain a maximum of 600 characters. The configuration utility displays the text in a window at least 40 characters wide.

HELP Statement (Optional)

Syntax:
[HELP = "*configuration help information*"]

The HELP statement contains information that is displayed to the user if requested. The HELP text field can contain a maximum of 600 characters. The configuration utility displays the text in a window at least 40 characters wide.

Example Switch Configuration Statement Block

The following example illustrates a switch configuration statement block.

```
            ;INITIALIZATION INFORMATION BLOCK
SWITCH(1)= 8                              ;1st switch--8 positions
    NAME = "SWITCH BLOCK 1"
    STYPE = DIP                           ;DIP switch type
    VERTICAL = YES                        ;Vertical orientation
    FACTORY = LOC(8-1) 11110000           ;Factory setting = 11110000
    INITVAL = LOC(8-1) xxxxxxx0           ;One reserved switch
SWITCH(2)= 2                              ;2nd Switch--2 positions
    NAME = "SWITCH BLOCK 2"
    STYPE = SLIDE                         ;SLIDE switch type
    LABEL = LOC(2 1) "IRQ9" "IRQ8"        ;Position labels IRQ9, IRQ8
    FACTORY = LOC(2 1) 10  ;IRQ9 Setting
```

4.6.2.3 Jumper Configuration Statement Block

The jumper configuration statement block begins with the JUMPER(i) statement. The syntax of the jumper configuration statement block is:

```
JUMPER(i) = value                                   ;Number of jumpers in set
    NAME = "jumper name or description"
    JTYPE = INLINE | PAIRED | TRIPOLE               ;Type of jumper
    [VERTICAL = YES | NO]                           ;Jumper orientation
    [REVERSE = YES | NO]                            ;Jumper numbering scheme
    [LABEL = LOC(jumperlist) textlist]              ;ASCII Jumper labels
    [INITVAL = LOC(jumperlist) valuelist]           ;Jumper settings
    [FACTORY = LOC(jumperlist) valuelist]           ;Factory setting
    [COMMENTS = "configuration comments"]
    [HELP = "configuration help information"]
```

JUMPER(i) Statement (Required)

Syntax:
JUMPER(i) = *value*

The JUMPER(i) statement specifies the number of jumper positions in a set. Each set of jumpers must have a separate JUMPER(i) statement with a different identifier, i. The "i" can be any positive integer value from 1 to 32767. The value field has two meanings here depending on the type of jumper defined. For inline jumpers, value refers to the number of connections. For tripole and paired jumpers, value refers to the number of tripole or paired sets.

NAME Statement (Required)

Syntax:
NAME = *"jumper name or description"*

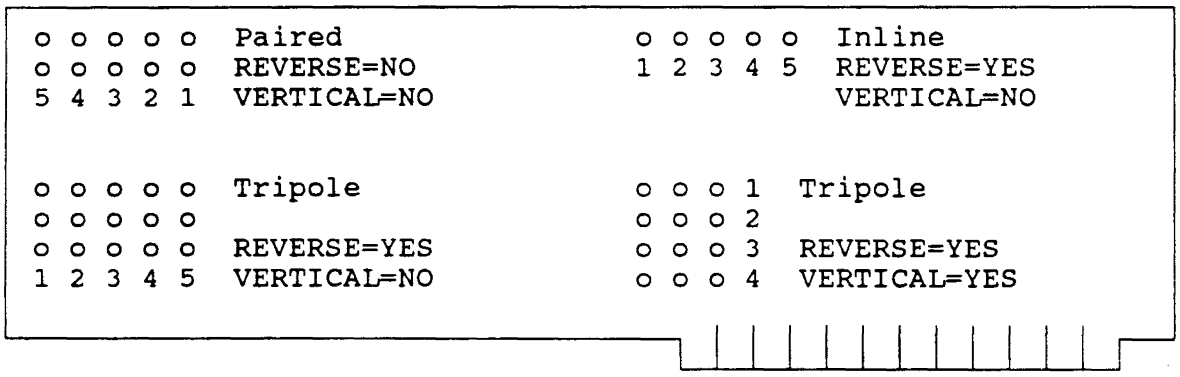
The NAME statement contains the jumper name as it is designated in the user documentation. The description can contain a maximum of 20 characters.

JTYPE Statement (Required)

Syntax:
JTYPE = INLINE | PAIRED | TRIPOLE

The JTYPE statement designates the type of jumper as INLINE, PAIRED, or TRIPOLE. INLINE jumpers are arranged in a straight line, such that each post can be connected to an adjacent post. PAIRED jumpers are arranged as a series of double posts, such that any single pair can be connected across the two posts. TRIPOLE jumpers are arranged as a series of triple posts, such that the middle post can be connected to either of the two adjacent posts.

The following figure illustrates each of the three JTYPEs.



VERTICAL Statement (Optional)

Syntax:
[VERTICAL = YES | NO]

The VERTICAL statement indicates the orientation of the jumper on the expansion board. The VERTICAL statement defaults to "NO."

REVERSE Statement (Optional)

Syntax:
[REVERSE = YES | NO]

The REVERSE statement specifies the order that a jumper is numbered. REVERSE = YES indicates 1234..., REVERSE = NO indicates ...4321 order. The REVERSE statement defaults to "NO."

LOC(*jumperlist*) *valuelist*

The jumper configuration statements LABEL, INITVAL and FACTORY include the LOC(*jumperlist*) *valuelist* string to reference individual jumper positions. The *jumperlist* contains a list of jumpers. The *valuelist* specifies the setting for each jumper. The *valuelist* must not be delimited with a space and must use the same order as the *jumperlist*.

A paired or tripole *jumperlist* can use a range to indicate the jumpers. The elements of the *jumperlist* must be in ascending order if REVERSE=YES, or descending order if REVERSE=NO. A space must be included between elements as a delimiter.

The *jumperlist* specifies inline jumpers by indicating the connection between two posts with a caret. For example, LOC(6^5 4^3 2^1) specifies the jumpers between posts 6 and 5, between posts 4 and 3, and between posts 2 and 1. The elements of the *jumperlist* must be in ascending order if REVERSE=YES, or descending order if REVERSE=NO. A space must be included between elements as a delimiter.

The paired and inline jumper *valuelist* settings can be indicated as "1" for "ON" (jumper installed), "0" for "OFF" (jumper not installed), or "x" for "don't care."

A tripole jumper *valuelist* settings can be indicated as "1" for "ON" (jumper installed in upper or right position), "0" for "OFF" (jumper installed in lower or left position unless otherwise marked), "n" for "NONE" (jumper not installed) or "x" for "don't care."

The following examples illustrate valid LOC(*jumperlist*) *valuelist* strings:

```
JTYPE=TRIPOLE
REVERSE=YES
INITVAL = LOC(1 2 3 4) 0011          ;List of tripole jumpers

JTYPE=PAIRED
REVERSE=NO
INITVAL = LOC(4 3 2 1) 1100          ;List of paired jumpers

JTYPE=PAIRED
REVERSE=YES
INITVAL = LOC(1-4) 0011              ;Range of paired jumpers

JTYPE=TRIPOLE
REVERSE=NO
INITVAL = LOC(4-1) 1100              ;Range of tripole jumpers

JTYPE=PAIRED
REVERSE=YES
INITVAL = LOC(1-4) x011              ;Range of paired jumpers with "x"

JTYPE=TRIPOLE
REVERSE=YES
INITVAL = LOC(1-4) x011              ;Range of tripole jumpers with "x"

JTYPE=TRIPOLE
REVERSE=YES
INITVAL = LOC(1-4) n011              ;Range of tripole jumpers with "n"

JTYPE=INLINE
REVERSE=NO
INITVAL = LOC(6^5 4^3 2^1) 101       ;List of inline jumpers
```

LABEL Statement (Optional)

Syntax:
[LABEL = LOC(*jumperlist*) *textlist*]

The LABEL statement specifies labels for individual jumpers. Each label can be composed of up s10 characters. If the LABEL statement is omitted, the default label is the switch number (...4321 for normal jumpers and 1234... for reverse jumpers). The following example illustrates use of the LABEL statement:

```
LABEL = LOC(4^3 2^1) "IRQ2" "IRQ3"          ;"IRQ2" (4^3), "IRQ3" (2^1)
```

INITVAL Statement (Optional)

Syntax:
[INITVAL = LOC(*jumperlist*) *valuelist*]

The INITVAL statement specifies the settings for factory-set jumpers that must not be changed. If the INITVAL statement is omitted, jumper settings are determined by the configuration program are "don't care." This statement is particularly important for jumpers that control undocumented options andrequire specific settings. The following example illustrates use of the INITVAL statement:

```
INITVAL = LOC(4 3 2 1) 0011      ;Paired (or tripole) jumper settings
```

FACTORY Statement (Optional)

Syntax:
[FACTORY = LOC(*jumperlist*) *valuelist*]

The FACTORY statement indicates the factory settings for the jumpers.

COMMENTS Statement (Optional)

Syntax:
[COMMENTS = "*configuration comments*"]

The COMMENTS statement contains information to assist the user in configuring a jumper. The COMMENTS text field can contain a maximum of 600 characters. The configuration utility displays the text in a window at least 40 characters wide.

HELP Statement (Optional)

Syntax:
[HELP = "*configuration help information*"]

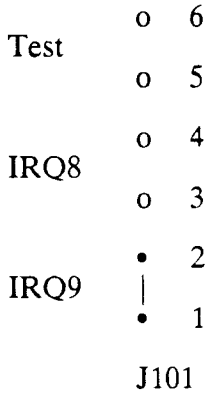
The HELP statement contains information that is displayed to the user if requested. The HELP text field can contain a maximum of 600 characters. The configuration utility displays the text in a window at least 40 characters wide.

Example Jumper Configuration Statement Block

The following example illustrates a jumper configuration statement block.

```
                ;INITIALIZATION INFORMATION BLOCK
JUMPER(1) = 5                                          ;1st set-5 positions(6 posts )
   NAME = "J101"
   JTYPE = INLINE                                      ;Inline jumper type
   VERTICAL = YES                                      ;Vertical orientation
   LABEL = LOC(6^5 4^3 2^1) "Test" "IRQ8" "IRQ9"       ;Labels Test, IRQ8, IRQ9
   INITVAL = LOC(6^5 4^3 2^1) 0xx                      ;Reserved jumper
   FACTORY = LOC(6^5 4^3 2^1) 001                      ;Factory Setting = IRQ9
```

The configuration utility displays a diagram to illustrate the jumper settings. For example:



4.6.2.4 SOFTWARE(Initialization) Statement Block (Optional)

Syntax:
SOFTWARE(i) = "*description*"

The software statement block begins with the SOFTWARE (i) statement. The syntax of the software configuration statement block is:

*Note: there are no other statements in the block.

The software initialization statement block provides user information and instructions about software drivers for display during system configuration. The instructions may, for example, indicate the software options to incorporate into the operating system startup files or a program that must be executed to initialize an expansion board. The software initialization statement block can include entries for the CONFIG.SYS and AUTOEXEC.BAT files of an MS-DOS operating system.

The startup files may execute programs that require command line parameters (for example, /s, /g).

Each software statement must have a separate SOFTWARE(i) statement with a different identifier, "i." The "i" can be any positive integer value from 1 to 32767. The *description* can be a maximum of 600 characters.

The configuration utility displays the software description with switch settings and other configuration information, during system configuration.

See the section on INIT Statements for more details about the software(i) statement.

4.6.3 FUNCTION Statement Block

A FUNCTION statement block consists of the following statements:

- FUNCTION Statement--identifies the name of the expansion board function (for example, "Asynchronous communications port").
- TYPE Statement--identifies the function type (for example: a communications port is type "COM").
- CHOICE Statements with resource description blocks--identify the configuration alternatives (i.e., initializations, I/O ports, interrupts, DMA channels and memory).

The FUNCTION statement block has the following format:

```
FUNCTION = "function name"
    [TYPE = "function type"]
    [COMMENTS = "information"]
    [CONNECTION = "connector orientation and description"]
    [HELP = "information"]
    CHOICE = "configuration name"
        [Resource Description Block]
    [CHOICE = "configuration name"
        Resource Description Block]
        .
        .
        .
    [CHOICE = "configuration name"
        Resource Description Block]
    [SUBFUNCTION STATEMENT BLOCK]
```

A separate function statement block must be supplied for each function of a multifunction expansion board. The following example illustrates the two function statement blocks for an expansion board with a communications port and a parallel port.

```
FUNCTION = "Asynchronous communications port"
    CHOICE = "configuration name"
        Resource Description Block
FUNCTION = "Parallel port"
    CHOICE = "configuration name"
        Resource Description Block
```

The configuration utility displays the software description with switch settings and other configuration information, during system configuration.

See the section on INIT Statements for more details about the software(i) statement.

4.6.3 FUNCTION Statement Block

A FUNCTION statement block consists of the following statements:

- FUNCTION Statement--identifies the name of the expansion board function (for example, "Asynchronous communications port").
- TYPE Statement--identifies the function type (for example: a communications port is type "COM").
- CHOICE Statements with resource description blocks--identify the configuration alternatives (i.e., initializations, I/O ports, interrupts, DMA channels and memory).

The FUNCTION statement block has the following format:

```
FUNCTION = "function name"
    [TYPE = "function type"]
    [COMMENTS = "information"]
    [CONNECTION = "connector orientation and description"]
    [HELP = "information"]
    CHOICE = "configuration name"
        [Resource Description Block]
    [CHOICE = "configuration name"
        Resource Description Block]
        .
        .
        .
    [CHOICE = "configuration name"
        Resource Description Block]
    [SUBFUNCTION STATEMENT BLOCK]
```

A separate function statement block must be supplied for each function of a multifunction expansion board. The following example illustrates the two function statement blocks for an expansion board with a communications port and a parallel port.

```
FUNCTION = "Asynchronous communications port"
    CHOICE = "configuration name"
        Resource Description Block
FUNCTION = "Parallel port"
    CHOICE = "configuration name"
        Resource Description Block
```

FUNCTION Statement (Required)

Syntax:
FUNCTION = "*function name*"

Each function statement block begins with a function statement that specifies the function name. The function name consists of free-form ASCII text with a maximum of 100 characters. All function names within a single configuration file must be unique, but different configuration files can have common function names.

The configuration utility displays the function name during configuration, but does not store it in nonvolatile memory.

TYPE Statement (Optional)

Syntax:
[TYPE = "*function type*"]

A functions statement block is supplemented with a TYPE statement that identifies the function type with a three-character ASCII string. The following table identifies commonly used function types.

Commonly Used Function Types

KEY--keyboard	PAR--parallel port
MEM--memory board	PTR--pointing device
MSD--mass storage device	COM--communications port
NET--network adapter	VID--video display adapter
NPX--numeric coprocessor	SYS--system board
OTH--other	OSE--operating system/environment

The TYPE statement should use one of the listed types when applicable. A TYPE statement can contain a type not included in the "Commonly Used Function Types" table above, but all types must be three-character ASCII strings. The type is stored in nonvolatile memory as upper-case. It should be entered in the configuration file in upper-case for consistency.

The function type can be supplemented by appending multiple, comma-delimited, ASCII strings to the initial three-character type. The supplemental type ASCII strings are not limited to three characters. For example, an asynchronous communications port can have the following TYPE statement:

TYPE = "COM,ASY"

The configuration utility stores the TYPE statement's ASCII string in nonvolatile memory during configuration. EISA systems provide a total of 80 bytes of nonvolatile memory to store the TYPE statement's ASCII string and SUBTYPE statement's ASCII string. The 80 bytes include the comma and semicolon delimiters between the type and SUBTYPE string fragments.

A device driver can use the type string to determine the general class of functionality of a device. The device driver can use the subtype string to determine the configuration of a device.

The resource description block section SUBTYPES are dicussed later in this specification.

COMMENTS Statement (Optional)

Syntax:
[COMMENTS = "*information*"]

A function statement block can include a COMMENTS statement that provides relevant information about the function. The comment could identify an expansion board manufacturer and part number, configuration instructions or any other useful information. The comment consists of free-form ASCII text with a maximum of 600 characters. The configuration utility displays the text in a window at least 40 characters wide.

The configuration utility displays the comment during configuration when the function is selected. It does not store the comment in nonvolatile memory.

HELP Statement (Optional)

Syntax:
[HELP = "*help information*"]

The HELP statement contains information that is displayed to the user if requested. The help text field can contain a maximum of 600 characters. The configuration utility displays the text in a window at least 40 characters wide.

CONNECTION Statement (Optional)

Syntax:
CONNECTION = "*connector orientation and description*"

A configuration file can specify the orientation and description of connectors by including the CONNECTION statement in the FUNCTION statement block.

The connection string consists of an ASCII string with a maximum length of 40 characters. Typical connection strings include "top," "bottom," "upper," "lower," "middle," etc. The configuration utility includes a command that displays the connection string.

4.6.3.1 CHOICE Statement Block

Each function statement block is accompanied by at least one choice statement block that specifies the initializations and system resource requirements of a possible configuration. The configuration utility uses the first choice statement block as the default. Multiple choice statement blocks are sequentially arranged in the order of preference. The choice statement block begins with a choice statement that specifies the "name" of the configuration. A choice statement block has the following syntax:

CHOICE = "*configuration name*"
[SUBTYPE = "*device description*"]
[DISABLE = YES | NO"]
[AMPERAGE = *value*]
[TOTALMEM = *rangelist* [STEP = *value*]]
Resource Description Block

A communications port, for example, can have the following function statement block and associated choice statement blocks:

FUNCTION = "Asynchronous Communications Port"
CHOICE = "COM1"
Resource Description Block
CHOICE = "COM2"
Resource Description Block

The system resource requirements (described in the "Resource Description Block" section) for the named configuration follow the CHOICE statement.

CHOICE Statement (Optional)

Syntax:
CHOICE = "*configuration name*"

The choice statement block begins with a CHOICE statement that specifies the "name" of the configuration. The "name" is an ASCII string with a maximum of 90 characters.

During configuration, the configuration utility displays all CHOICE statement configuration names for the selected function. The configuration utility does not store the name in nonvolatile memory.

DISABLE Statement (Optional)

Syntax:
[DISABLE = YES | NO]

A CHOICE statement can be used to disable the expansion board function. Each function to be disabled requires a separate DISABLE = YES statement. The default is DISABLE = NO. The following example illustrates use of the DISABLE = YES statement.

```
FUNCTION = "Communications Port"
    CHOICE = "COM1"
        Resource Description Block
    CHOICE = "COM2"
        Resource Description Block
    CHOICE = "Disable Communications Port"
        DISABLE = YES
```

SUBTYPE Statement (Optional)

Syntax:
[SUBTYPE = "*device description*"]

Each choice statement block can contain a subtype statement that names the configuration (with a short mnemonic) associated with the choice. The subtype can be supplemented by appending multiple, semicolon-delimited, ASCII strings to the initial subtype.

A device driver can use the SUBTYPE string to determine the configuration of a device. The device driver may use the type string to determine the general class of functionality of a device.

A communications port may have SUBTYPE statements as follows:

```
FUNCTION = "Internal Modem"
    TYPE = "COM,ASY,MDM"
    CHOICE = "Modem assigned to COM1"
        SUBTYPE = "COM1"
        Resource Description Block
    CHOICE = "Modem assigned to COM2"
        SUBTYPE = "COM2"
        Resource Description Block
```

The SUBTYPE should be a short ASCII string. The SUBTYPE string supplements the type string by identifying the selected configuration (the type string identifies the type of device). The configuration utility stores the concatenated type and SUBTYPE ASCII strings, with a semicolon delimiter, in nonvolatile memory during configuration. EISA systems provide a total of 80 bytes of nonvolatile memory to store the type statement's ASCII string and SUBTYPE statement's ASCII string. The 80 bytes include the comma and semicolon delimiters between type and SUBTYPE string fragments.

AMPERAGE Statement (Optional)

Syntax:
[AMPERAGE = *value*]

The AMPERAGE statement, when included in the choice statement block, specifies the maximum amount of continuous 5V current (in milliamps) required by the option specified by the choice statement block. The total 5V current includes the amount specified in the board identification block plus the amount specified for the selected options. The AMPERAGE statement does not apply to virtual devices.

TOTALMEM Statement (Optional)

Syntax:
TOTALMEM = *rangelist* [STEP = *value*]

A choice statement block can contain a TOTALMEM statement that indicates the total amount of memory specified by the choice. The TOTALMEM statement is required for a memory block that can have its allocation split between system memory (SYS) and expanded memory (EXP).

See the TOTALMEM statement and example in the section entitled, Memory Description Block, for more detailed information.

4.6.3.2 SUBCHOICE Statement Block

The purpose of the subchoice statement block is to handle resource statement alternatives that are too complex for individual CHOICE statements (for example, memory configurations of some memory boards).

A choice statement block can include statements that specify alternative configurations. A subchoice statement block can use any statement that is valid for a choice statement block. The subchoice alternatives must be automatically selectable by the configuration utility with information available from the configuration files. The configuration utility does not present subchoice alternatives for selection by a user, although the user can scroll through the resources specified in subchoice statement blocks.

The syntax for the SUBCHOICE statement is shown below:

SUBCHOICE
Resource Description Block

A choice statement block can have as many subchoice statement blocks as needed. The configuration utility sequentially checks each subchoice resource description block and selects the first one that does not conflict with other devices in the configuration.

The combination of the choice resource description block and one subchoice resource description block contains the resource and initialization requirements for the configuration. The configuration utility includes the choice and the selected SUBCHOICE resource requirements in the data written to nonvolatile memory for use by the power-up routines.

The following example illustrates a configuration file fragment that specifies a memory allocation that back fills 128K of memory into the base address range between 512K and 640K if only 512K is installed. The remainder of memory on the expansion board is allocated to extended memory. The user selects the total amount of memory on the expansion board and views the subchoice alternatives. The subchoice selection (between back fill and extended memory) does not require input from the user, since the amount of base memory installed is available from the configuration file. The subchoice statement blocks are included in a single choice statement block that is presented to the user:

```
CHOICE = "Add Base and Extended Memory"
    TOTALMEM = 128K-2048K  STEP 128K

;   128K base memory back fill into range 512K-640K
;       (512K base memory already installed)

    SUBCHOICE
        FREE                                ;128K back fill
            MEMORY = 128K
            ADDRESS = 512K
            MEMTYPE = SYS
        COMBINE                             ;Extended Memory for the rest
            MEMORY = 0K-1920K  STEP 128K
            ADDRESS = 1M
            MEMTYPE = SYS

;   No base memory back fill
;       (640K base memory already installed)

    SUBCHOICE
        COMBINE                             ;All Extended Memory
            MEMORY = 128K-2048K  STEP 128K
            ADDRESS = 1M-16M  STEP 128K
            MEMTYPE = SYS
```

Selection of the starting address could be presented to the user as a sequence of CHOICE statements for selection by the user:

```
;    128K base memory back fill into range 512K-640K
;         (512K base memory already installed)

CHOICE = "Add Base and Extended Memory"
     TOTALMEM = 128K-2048K  STEP 128K
     FREE                        ;128K back fill
          MEMORY = 128K
          ADDRESS = 512K
          MEMTYPE = SYS
     COMBINE                     ;Extended Memory for the rest
          MEMORY = 0K-1920K  STEP 128K
          ADDRESS = 1M
          MEMTYPE = SYS

;    No base memory back fill
;         (640K base memory already installed)

CHOICE = "Add Extended Memory"
     TOTALMEM = 128K-2048K  STEP 128K
          COMBINE                ;All Extended Memory
          MEMORY = 128K-2048K  STEP 128K
          ADDRESS = 1M-16M  STEP 128K
          MEMTYPE = SYS
```

The configuration utility presents each named choice to the user for selection. The user can make the selection or let the configuration utility automatically make the selection.

SUBCHOICE statements are not appropriate if the user might need to make the selection. For example, the user may need to select a serial port as COM1 or COM2. The configuration utility presents the choices to the user, and the user either makes the selection manually or lets the configuration utility select automatically.

SUBCHOICE Statement (Optional)

Syntax:
[SUBCHOICE]

The subchoice statement block begins with a SUBCHOICE statement. The SUBCHOICE statement does not have a name field for display, since subchoice statement blocks are selected automatically by the configuration utility.

SUBFUNCTION Statement Block (Optional)

A function statement block may contain one or more subfunction statement blocks that specify the configuration information for a set of related components with separate resource or initialization requirements. A subfunction statement block provides separate configuration of the function's components.

A subfunction statement block can use any statement that is valid for a function statement block. The syntax of a subfunction statement block is:

```
SUBFUNCTION = "function name"
    [TYPE = "function type"]
    [COMMENTS = "information"]
    [CONNECTION = "connector orientation and description"
    [HELP = "information"]
    CHOICE = "configuration name"
        Resource Description Block
    [CHOICE = "configuration name"
        Resource Description Block]
        .
        .
        .
    [CHOICE = "configuration name"
        Resource Description Block]
```

The configuration utility stores the resource and initialization information from subfunction statement blocks with the function information. Subfunction statement blocks are not stored as separate functions in nonvolatile memory.

Syntax:
SUBFUNCTION = "*name*"

The subfunction statement block begins with a subfunction statement that specifies the *name* of the configuration. The *name* is an ASCII string with a maximum of 90 characters.

During configuration, the configuration utility displays all CHOICE configuration names for the selected subfunction.

The following example illustrates use of subfunction statement blocks to configure the parity and baud rate for an asynchronous communications port. The example includes the statement blocks with type and subtype strings. The resource and initialization statements are omitted for simplicity.

```
FUNCTION = "1200/2400 Baud Modem"
    TYPE = "COM,ASY,MDM"
    SUBFUNCTION = "Port Address"
        CHOICE = "COM1 Serial Port"
            SUBTYPE = "COM1"
        CHOICE = "COM2 Serial Port"
            SUBTYPE = "COM2"
    SUBFUNCTION = "Parity Selection"        ;No SUBTYPE under SF
        CHOICE = "ODD"
            SUBTYPE = "PARITY=ODD"          ;SUBTYPE under CHOICE
        CHOICE = "EVEN"
            SUBTYPE = "PARITY=EVEN"
    SUBFUNCTION = "Baud Rate Selection"
        CHOICE = "1200 Baud"
            SUBTYPE = "BAUD=1200"           ;SUBTYPE under CHOICE
        CHOICE = "2400 Baud"
            SUBTYPE = "BAUD=2400"
```

The type/subtype string for the "1200/2400 Baud Modem" function (with COM1, odd parity and 2400 baud selections) in nonvolatile memory is:

"COM,ASY,MDM;COM1;PARITY=ODD;BAUD=2400"

The example above used SUBTYPE statements under the CHOICE statements but not under the SUBFUNCTION statements. The following example illustrates an alternative method with the SUBTYPE statements under the SUBFUNCTION and the CHOICE statements:

```
FUNCTION = "1200/2400 Baud Modem"
    TYPE = "COM,ASY,MDM"
    SUBFUNCTION = "Port Address"
        CHOICE = "COM1 Serial Port"
            SUBTYPE = "COM1"
        CHOICE = "COM2 Serial Port"
            SUBTYPE = "COM2"
    SUBFUNCTION = "Parity Selection"
        TYPE = PARITY                       ;TYPE under SUBFUNCTION
        CHOICE = "ODD"
            SUBTYPE = "ODD"                 ;SUBTYPE under CHOICE
        CHOICE = "EVEN"
            SUBTYPE = "EVEN"
    SUBFUNCTION = "Baud Rate Selection"
        TYPE = BAUD                         ;TYPE under SUBFUNCTION
        CHOICE = "1200 Baud"
            SUBTYPE = "1200"                ;SUBTYPE under CHOICE
        CHOICE = "2400 Baud"
            SUBTYPE = "2400"
```

The type/subtype string for the "1200/2400 Baud Modem" function (with COM1, odd parity and 2400 baud selections) in nonvolatile memory is:

"COM,ASY,MDM:COM1,PARITY;ODD,BAUD;2400"

4.6.3.3 GROUP Statement Block

A group statement block may be used to enclose a set of functionstatement blocks that specify the configuration information for a set of related components with separate resource or initialization requirements.

A set of grouped function statement blocks allows separate configuration of a function's components. A grouped function statement block can use any statement that is valid for independent function statement blocks. The syntax of a grouped set of function statement blocks is:

```
GROUP = "name"
    [TYPE = "type"]
FUNCTION = "name"
    [TYPE = "function type"]
    [COMMENTS = "information"]
    [HELP = "information"]
    CHOICE = "name"
        resource description block
        .
        .
    [CHOICE = "name"
        resource description Block]
FUNCTION = "name"
    [TYPE = "function type"]
    [COMMENTS = "information"]
    [HELP = "information"]
    CHOICE = "name"
        resource description block
        .
        .
    [CHOICE = "name"
        resource description block]
FUNCTION = "name"
    [TYPE = "function type"]
    [COMMENTS = "information"]
    [HELP = "information"]
    CHOICE = "name"
        resource description block
        .
        .
    [CHOICE = "name"
        resource description block]
FUNCTION = "name"
    .
    .
ENDGROUP
```

The configuration utility saves the resource and initialization information for each function specified in the grouped set as a separate function entry in nonvolatile memory. The group statement block may include a TYPE statement. The group type string prepends to each TYPE string in the set of grouped function statement blocks. The configuration utility stores the group type string in nonvolatile memory for a grouped function statement block that omits the type statement.

Presentation of options during configuration and TYPE string prepending in nonvolatile memory are the only differences between a set of grouped FUNCTION statement blocks and a set of independent FUNCTION statement blocks.

GROUP and ENDGROUP Statements (Optional)

Syntax:
[GROUP = "name"]
.
.
.
[ENDGROUP]

A grouped set of function statement blocks begins with the GROUP statement and ends with an ENDGROUP statement. The group name can be a maximum of 60 characters. Each GROUP statement must have a corresponding ENDGROUP statement.

Example Use of Grouped FUNCTION Statement Blocks

The following configuration file fragment illustrates the use of grouped function statement blocks that specify the configuration options for a fixed disk controller and disk drive. For simplicity, the configuration file fragment includes the TYPE and SUBTYPE statements, but does not include resource or initialization statements. The GROUP statement block and some function statement blocks have a TYPE statement.

```
GROUP = Fixed Disk Drives     ;Fixed disk controller group
    TYPE = "MSD"              ;Prepends to each FUNCTION TYPE
FUNCTION = "Fixed Disk Controller Selection"
    TYPE = "DSKCTL"
    CHOICE = "Primary Controller"
        SUBTYPE = "PRI"
    CHOICE = "Secondary Disk Controller"
        SUBTYPE = "SEC"
FUNCTION = "Device for Unit 1"
    TYPE = "UNIT1"
    CHOICE = "Not Installed"
        SUBTYPE = "DSKDRV,TYP=00"
    CHOICE = "300mb - TYPE 38"
        SUBTYPE = "DSKDRV,TYP=38"
    CHOICE = "130mb - TYPE 43"
        SUBTYPE = "DSKDRV,TYP=43"
FUNCTION = "Device for UNIT 2"
    TYPE = "UNIT2"
    CHOICE = "Not Installed"
        SUBTYPE = "DSKDRV,TYP=00"
    CHOICE = "300mb - TYPE 38"
        SUBTYPE = "DSKDRV,TYP=38"
    CHOICE = "130mb - TYPE 43"
        SUBTYPE = "DSKDRV,TYP=43"
ENDGROUP
```

The grouped function statement blocks are stored separately in nonvolatile memory. The type string for each of the function statement blocks includes the group type string (prepended to the function type string). Nonvolatile memory contains the following type strings (assuming the choice selections are: primary controller with a 300 MB drive for UNIT1 and UNIT2 is not installed).

```
FUNCTION = "Fixed disk Controller Selection"
        TYPE string: MSD,DSKCTL;PRI

FUNCTION = "Device for Unit 1"
        TYPE STRING: MSD,UNIT1,DSKDRV,TYP=38

FUNCTION = "Device for Unit 2"
        TYPE string: MSD,UNIT2,DSKDRV,TYP=00
```

4.6.4 Resource Description Block

A resource description block may accompany each CHOICE statement to identify the initialization and system resource requirements of the named configuration. The resource description block can contain any of the following information:

- DMA Channel Description Block--specifies the choice of DMA channels supported, whether the channel can be shared, the channel's data size, the channel's cycle timing, and any initialization necessary
- Interrupt Description Block--specifies the choice of interrupts supported, whether the interrupt can be shared, whether the interrupt is edge or level sensitive, and any initialization necessary
- I/O Port Description Block--specifies the port address, and any initialization necessary
- Memory Description Block--specifies the amount of memory supported, the starting address, and whether the memory is cacheable it also identifies the memory as RAM or ROM, defines the memory usage (system, expanded, virtual or other), and specifies any initialization necessary to configure the memory
- Switch and Jumper Description Blocks--specify the switch and jumper settings for the configuration
- Programmable Port Initialization Block--specifies the initialization for programmable ports for the configuration
- Software Initialization Block--specifies any software initialization necessary

The syntax of a DMA resource description block is as follows:

```
[DMA = list
    [SHARE = YES | NO | "text"]
    [SIZE = BYTE | WORD | DWORD]
    [TIMING = DEFAULT | TYPEA | TYPEB | TYPEC]]
```

The syntax of an I/O port resource description block is as follows:

```
[PORT =  list/rangelist [STEP = value [COUNT = VALUE]
    [SHARE = YES | NO | "text"]
    [SIZE = BYTE | WORD | DWORD]]
```

The syntax of an interrupt resource description block is as follows:

```
[IRQ = list
    [SHARE = YES | NO | "text"]
    [TRIGGER = LEVEL | EDGE]]
```

The syntax of a memory resource description block is as follows:

```
[MEMORY = rangelist [STEP = value]
    [ADDRESS = rangelist [STEP = value]]
    [WRITABLE = YES | NO]
    [MEMTYPE = SYS | EXP | VIR | OTH]
    [CACHE = YES | NO]
    [SHARE = YES | NO | "text"]
    [SIZE = BYTE | WORD | DWORD]
    [DECODE = 20 | 24 | 32]
```

4.6.4.1 DMA Channel Description Block

A DMA channel description block consists of a group of statements that specifies the DMA channels required by an expansion board function. The configuration file can contain a maximum of four DMA description blocks for any one function. The syntax of a DMA channel description block is:

```
DMA = DMA channel number
    [SHARE = YES | NO | "text"]
    [SIZE = BYTE | WORD | DWORD]
    [TIMING = DEFAULT | TYPEA | TYPEB | TYPEC]
```

An OR operator can be used to separate multiple DMA channel lists (as illustrated in the following syntax) if each list supports identical SHARE, SIZE and TIMING characteristics:

```
DMA = value [| value] ...
    [SHARE = YES | NO | "text"]
    [SIZE = BYTE | WORD | DWORD]
    [TIMING = DEFAULT | TYPEA | TYPEB | TYPEC]
```

Multiple DMA channel description blocks must be used for a function with multiple DMA channels that have different share, size or timing characteristics, as illustrated in the following syntax:

```
DMA = DMA channel number   ;1st DMA channel
    [SHARE = YES | NO | "text"]
    [SIZE = BYTE | WORD | DWORD]
    [TIMING = DEFAULT | TYPEA | TYPEB | TYPEC]
DMA = DMA channel number   ;2nd DMA channel
    [SHARE = YES | NO | "text"]
    [SIZE = BYTE | WORD | DWORD]
    [TIMING = DEFAULT | TYPEA | TYPEB | TYPEC]
```

An expansion board function can request up to four DMA channels. Each channel selected during system configuration is stored in nonvolatile memory with the appropriate share, size and timing characteristics.

If the DMA channel is defined as "shared," then it is the responsibility of the device driver to initialize the DMA Extended Mode and DMA Command Registers before starting each DMA transfer. The System ROM, in this case, does not initialize these registers.

The device driver can get the DMA information (shared/not shared) from nonvolatile memory to decide if it needs to initialize the DMA Extended Mode Register and DMA Command Register.

If the DMA channel is defined as "not shared," then the configuration file should provide initialization values for the DMA Command Registers, and the system ROM will automatically program the DMA Extended Mode Registers as follows:

DMA Extended Mode Register
- **DMA channel cycle timing (from DMA data provided in the structure)**
- **DMA data size and addressing mode (from DMA data in the structure)**
- **T-C = 0; set to be output for this channel (ISA default)**
- **Stop Register = disabled (ISA default)**

The configuration file should not, in either case, provide initialization values for the DMA Extended Mode Registers.

DMA Statement (Optional)

Syntax:
DMA = value [| value] ...

The DMA statement marks the beginning of a DMA description block and specifies the DMA channel number (or list of channels or multiple lists of channels) supported by the configuration.

SHARE Statement (Optional)

Syntax:
[SHARE = YES | NO | "*text*"]

The SHARE statement specifies whether the function can share the DMA channel. The default for SHARE is NO. A text identifier can be specified to indicate that the function can only share the DMA channel with a device that has a matching identifier. The identifier can be up to 10 characters.

DMA channels can be shared by two devices that never require the channel simultaneously. For example, a floppy drive and tape drive attached to the same controller could share a DMA channel since the floppy drive and tape drive never use the channel at the same time.

Two devices that may need to transfer data at the same time cannot share a DMA channel. Two network adapters, for example, would have conflicting requirements for a single DMA channel.

SIZE Statement (Optional)

Syntax:
[SIZE = BYTE | WORD | DWORD]

The SIZE statement indicates the DMA device data transfer width as BYTE, WORD or DWORD. The default size is BYTE for DMA channels 0-3 and WORD for channels 4-7.

TIMING Statement (Optional)

Syntax:
[TIMING = DEFAULT | TYPEA | TYPEB | TYPEC]

The TIMING statement indicates the bus cycle type executed by the DMA controller during the transfer. The default transfer cycle type is default, which is compatible with ISA DMA devices. Higher performance ISA devices can use type A or type B for faster transfers. DMA devices that support EISA bus cycles can use type C (burst) DMA transfers, which provide the highest data transfer rate.

The DMA cycle types and timing are described in section 2 of this specification.

Example DMA Channel Request Block

The ACME tape controller can use DMA channel 3 or 5 and cannot share the channel. The ACME tape controller uses 16-bit DMA transfers and can support type B timing. The following diagram illustrates the DMA request block for the ACME tape controller:

```
DMA = 3 | 5
    SHARE = NO
    SIZE = WORD
    TIMING = TYPEB
```

4.6.4.2 Interrupt Description Block

An interrupt description block consists of a group of statements that specifies the interrupt requirements of an expansion board. The configuration file can contain a maximum of seven interrupt description blocks for any one function. The interrupt description block has the following format:

```
IRQ = value [| value] ...
    [SHARE = YES | NO | "text"]
    [TRIGGER = LEVEL | EDGE]
```

Multiple interrupt request blocks must be used for a function with multiple interrupts that have different share and trigger characteristics, as illustrated in the following syntax:

```
IRQ = value
    [SHARE = YES | NO | "text"]
    [TRIGGER = LEVEL | EDGE]
IRQ = value
    [SHARE = YES | NO | "text"]
    [TRIGGER = LEVEL | EDGE]
```

An OR operator can be used to separate multiple interrupts (as illustrated in the following syntax) if each interrupt supports identical share and trigger characteristics:

```
IRQ = value [| value] ...
    [SHARE = YES | NO | "text"]
    [TRIGGER = LEVEL | EDGE]
```

IRQ Statement (Optional)

Syntax:
IRQ = *Interrupt number*

The IRQ statement marks the beginning of an interrupt request block and specifies the interrupt number (or multiple interrupts) supported by the configuration.

Each interrupt selected during system configuration is stored in nonvolatile memory with the appropriate share and trigger characteristics. The interrupt device driver can retrieve the interrupt controller initialization information from nonvolatile memory to determine the method of handling interrupts.

The system ROM automatically determines the I/O port address and initialization values and programs the interrupt controller edge/level register. The configuration file should not provide initialization values for programming the interrupt controller edge/level register.

SHARE Statement (Optional)

Syntax:
[SHARE = YES | NO | *"text"*]

The SHARE statement indicates whether the function can share this interrupt. The default value for this field is NO. For EISA boards capable of sharing interrupts, this field should be SHARE = YES. A text identifier can be specified to indicate that the function can only share the interrupt with a device that has a matching identifier. The identifier can be up to 10 characters.

TRIGGER Statement (Optional)

Syntax:
[TRIGGER = LEVEL | EDGE]

The TRIGGER statement specifies whether the ROM initializes the interrupt controller to edge or level triggered. The default is TRIGGER = EDGE. In most cases, if the SHARE statement is YES, the TRIGGER statement should be set to LEVEL; however, there are some designs that require shared, edge-triggered interrupts, so a TRIGGER = LEVEL statement does not necessarily have to follow a SHARE = YES statement.

Example Interrupt Description Block

The ACME tape controller needs two interrupts. It can use interrupts 12 or 15, but it cannot share the assigned interrupts. The ACME tape controller needs the chosen interrupts to be edge triggered. Note that share and trigger fields could be omitted, because the defaults are used.

```
IRQ = 12 | 15
    SHARE = NO
    TRIGGER = EDGE
```

4.6.4.3 I/O Port Description Block

An I/O port description block consists of a group of statements that specifies the I/O ports used by a device. The configuration file can contain a maximum of 20 I/O port description blocks for any one function. The I/O Port Request Block has the following format:

```
PORT = range/list [STEP = value [COUNT = value]]
    [SHARE = YES | NO | "text"]
    [SIZE = BYTE | WORD | DWORD]
```

PORT Statement (Optional)

Syntax:
PORT = list/range [STEP = *value* [COUNT = *value*]]
or
PORT = *list*

The I/O Port Request Block begins with a PORT statement. The PORT statement can specify a single address, a list of addresses, or a rangelist that specifies the selections for the port address.

The STEP parameter that follows the *rangelist* identifies the address increment of the port selections. The COUNT parameter specifies the number of ports allocated from the selected STEP address block. If the COUNT parameter is omitted, the configuration utility uses a default COUNT value equal to the STEP value. If the STEP parameter is omitted, the configuration utility allocates the entire range (a COUNT without STEP is invalid). The following examples illustrate a PORT statement with a *rangelist*:

```
;allocates 16 ports: 300h-30Fh
PORT = 300h-30Fh

;allocates 4 ports: 300h-303h or 304h-307h or 308h-30Bh or 30Ch-30Fh
PORT = 300h-30Fh STEP = 4

;allocates 2 ports: 300h-301h or 304h-305h or 308h-309h or 30Ch-30Dh
PORT = 300h-30Fh  STEP = 4  COUNT = 2
```

SHARE Statement (Optional)

Syntax:
[SHARE = YES | NO | *"text"*]

The SHARE statement specifies whether the function can share the requested ports. The configuration utility uses a default of NO (the port cannot be shared) if the SHARE statement is omitted. A text identifier can be specified to indicate that the function can only share the port address with a device that has a matching identifier. The identifier may be up to 10 characters.

SIZE Statement (Optional)

Syntax:
[SIZE = BYTE | WORD | DWORD]

The SIZE statement specifies the size of the I/O port as BYTE (8-bit), WORD (16-bit) or DWORD (32-bit). The default size is BYTE.

4.6.4.4 Memory Description Block

The memory description block specifies the amount of memory on an expansion board and its starting address, whether the memory is cacheable, whether it is RAM or ROM, the type of memory (system, expanded, virtual or other), and initialization requirements of the memory. The configuration file can contain a maximum of nine memory description blocks for any one function. The memory request block has the following format:

```
MEMORY = list/range [STEP = value]
    [ADDRESS = rangelist [STEP = value]]
    [WRITABLE = YES | NO]
    [MEMTYPE = SYS | EXP | VIR | OTH]
    [SIZE = BYTE | WORD | DWORD]
    [DECODE = 20 | 24 | 32]
    [CACHE = YES | NO]
    [SHARE = YES | NO | "text"]
```

MEMORY Statement (Optional)

Syntax:
MEMORY = *range* [STEP = *value*]

The MEMORY statement signifies the beginning of the memory description block. The *range* following the MEMORY statement specifies the minimum and maximum amount of memory that can be put on the board. Each possible memory configuration can be listed separately (such as, 1M, 2M, 3M for one to three megabytes) or a minimum-to-maximum range can be specified (1M-3M). A minimum value of 1K is required and the minimum-to-maximum range must be at least 1K. The maximum range value is 64 megabytes.

If a *range* is specified, the STEP field must also be included to define the smallest increment by which additional memory can be added to the board.

ADDRESS Statement (Optional)

Syntax:
ADDRESS = *range* [STEP = *value*]
or
ADDRESS = *list*

The ADDRESS statement specifies the starting address of the memory. The ADDRESS statement is optional for memory if expanded or other is chosen for the memory type. The ADDRESS statement is required for system and virtual memory. The STEP parameter that follows the *range* identifies the addresses within the range that can be used as the starting address. The following example illustrates the valid starting address selections:

MEMORY = 1M
ADDRESS = 1M-4M STEP = 1M

Starting Address	Ending Address
100000h	1FFFFFh
200000h	2FFFFFh
300000h	3FFFFFh
400000h	4FFFFFh

WRITABLE Statement (Optional)

Syntax:
[WRITABLE = YES | NO]

The WRITABLE field indicates whether the memory is RAM or ROM; for ROM this field is NO. The default is YES.

MEMTYPE Statement (Optional)

Syntax:
[MEMTYPE = SYS | EXP | VIR | OTH]

The MEMTYPE field specifies whether the memory is SYStem (base and extended memory), EXPanded (LIM EMS memory available for use by an expanded memory manager), or OTHer (address space used for memory mapped I/O or bank-switched memory). The default is SYS. VIRtual indicates that the address space is used, but no physical memory occupies the address (address of a LIM page frame). Accesses to VIR memory do not generate addresses on the EISA bus. OTH is intended primarily for memory mapped I/O devices such as network adapters. OTH should include an ADDRESS statement only if it resides in the physical address space.

SIZE Statement (Optional)

Syntax:
[SIZE = BYTE | WORD | DWORD]

The SIZE statement identifies the memory as BYTE (8-bit), WORD (16-bit) or DWORD (32-bit) memory. The SIZE defaults to DWORD if the SIZE statement is omitted.

DECODE Statement (Optional)

Syntax:
[DECODE = 20 | 24 | 32]

DECODE is an optional statement that specifies the number of address lines decoded by a memory expansion board. The default is 32 for all memory boards.

CACHE Statement (Optional)

Syntax:
[CACHE = YES | NO]

The CACHE statement indicates whether the memory contents can be stored in cache memory. The memory on a graphics board, for example, generally should not be stored in a cache memory. The default is NO.

SHARE Statement (Optional)

Syntax:
[SHARE = YES | NO | *"text"*]

The SHARE statement indicates whether the memory in this space can be shared by another device. The default is NO. A text identifier can be specified to indicate that the function can only share the memory address range with a device that has a matching identifier. The identifier can be up to 10 characters.

TOTALMEM Statement (Optional)

Syntax:
TOTALMEM = list/range [STEP = *value*]

A choice statement block can contain a TOTALMEM statement that indicates the total amount of memory specified by the CHOICE. The TOTALMEM statement is required for a memory block that can have its allocation split between system memory (SYS), other memory (OTH) and expanded memory (EXP).

The TOTALMEM statement can include each possible memory size or provide a minimum-to-maximum range of possible configurations. A range must include the STEP keyword to indicate the smallest memory increment that can be added to the memory board.

DECODE Statement (Optional)

Syntax:
[DECODE = 20 | 24 | 32]

DECODE is an optional statement that specifies the number of address lines decoded by a memory expansion board. The default is 32 for all memory boards.

CACHE Statement (Optional)

Syntax:
[CACHE = YES | NO]

The CACHE statement indicates whether the memory contents can be stored in cache memory. The memory on a graphics board, for example, generally should not be stored in a cache memory. The default is NO.

SHARE Statement (Optional)

Syntax:
[SHARE = YES | NO | *"text"*]

The SHARE statement indicates whether the memory in this space can be shared by another device. The default is NO. A text identifier can be specified to indicate that the function can only share the memory address range with a device that has a matching identifier. The identifier can be up to 10 characters.

TOTALMEM Statement (Optional)

Syntax:
TOTALMEM = list/range [STEP = *value*]

A choice statement block can contain a TOTALMEM statement that indicates the total amount of memory specified by the CHOICE. The TOTALMEM statement is required for a memory block that can have its allocation split between system memory (SYS), other memory (OTH) and expanded memory (EXP).

The TOTALMEM statement can include each possible memory size or provide a minimum-to-maximum range of possible configurations. A range must include the STEP keyword to indicate the smallest memory increment that can be added to the memory board.

4.6.4.5 INIT Statements

INIT statements specify the initializations for alternative configurations. An INIT statement can be used to initialize any of the following:

DMA
IRQ
PORT
MEMORY

The configuration utility determines the initializations for the selected configuration and stores them in nonvolatile memory. The system ROM power-up routine performs the initializations.

I/O Port INIT Statement

Syntax:
INIT = IOPORT(i) [LOC(*bitlist*)] valuelist
or
INIT = PORTADR(*address*) [[BYTE | WORD | DWORD] *list*

The I/O port INIT statement specifies an I/O port and the binary value to write to the port for the configuration.

The INIT statement can specify the I/O port address, port size, and value directly in the PORTADR(*address*) form of the statement. The default port size is BYTE. This statement syntax provides a shorthand form of specifying I/O port values where no initialization information block is required. When this shorthand format is used, all bits must be specified with a 1, 0, or r (i.e., x's are not allowed to specify bits in this format).

The INIT statement can also indicate the address with an IOPORT(i) statement combined with the IOPORT(i) form of the INIT statement. The port size is specified with the IOPORT(i) statement, not in the INIT statement.

The *list* portion specifies the binary values to initialize the port. The values must be binary.

The INIT statement can include the LOC(*bitlist*) string to reference individual bits. The *bitlist* contains a list or range of bit positions. The elements of the *bitlist* must be in MSBit to LSBit order. A space must be included between elements as a delimiter.

```
INIT = PORTADR(0z800h) WORD 0000111100001111b    ;WORD port

INIT = PORTADR(0z800h) 00001111b                 ;Byte port

INIT = PORTADR(0z800h) 001100rr                  ;Byte port with "r" bits

INIT = IOPORT(1)(0z800h) LOC(7-2)  001100        ;Byte port (range)

INIT = IOPORT(2)(0z800h) LOC(7 6 1 0)  0011      ;4 bits specified

INIT = IOPORT(3)(0z800h) 00001111
```

Switch INIT Statement

Syntax:
INIT = SWITCH(i) LOC(*switchlist*) *valuelist*

The switch INIT statement specifies the switch positions and the appropriate setting for the configuration. SWITCH(i) indicates the switch being initialized. LOC(*switchlist*) *valuelist* identifies the switch positions and specifies the setting.

The LOC(*switchlist*) contains a list or range of switch positions. The elements of the *switchlist* must be in ascending order if REVERSE=YES, or descending order if REVERSE=NO. A space must be included between elements as a delimiter.

The *valuelist* specifies the switch setting for each switch position. The *valuelist* must use the same order as the *switchlist*. A DIP switch can be set for "1" to indicate "ON," or "0" to indicate "OFF." The dip switch settings are not delimited with a space. The *valuelist* for a rotary or slide switch indicates the selected position number by a "1" in the appropriate bit position.

Jumper INIT Statement

Syntax:
INIT = JUMPER(i) LOC(*jumperlist*) *valuelist*

The jumper INIT statement specifies the jumper positions and the appropriate setting for the configuration. JUMPER(i) indicates the jumper being initialized. LOC(*jumperlist*) specifies the jumper positions being specified.

The LOC(*jumperlist*) contains a list of jumper positions. The *valuelist* specifies the setting for each jumper position. The *valuelist* must not be delimited with a space and must use the same order as the *jumperlist*.

The *jumperlist* specifies paired and tripole jumpers by their jumper positions. A paired or tripole *jumperlist* can use a range to indicate the jumpers. The elements of the *jumperlist* must be in ascending order if REVERSE=YES, or descending order if REVERSE=NO. A space must be included between elements as a delimiter.

The *jumperlist* specifies inline jumpers by indicating the connection between two posts with a caret. For example, LOC(1^2 3^4 5^6) specifies the jumper between posts 1 and 2, between posts 3 and 4, and between posts 5 and 6. The elements of the *jumperlist* must be in ascending order if REVERSE=YES, or descending order if REVERSE=NO. A space must be included between elements as a delimiter.

Paired and inline jumper *valuelist* settings can be indicated as "1" for "ON" (jumper installed), "0" for "OFF" (jumper not installed). The paired jumper settings are not delimited with a space.

A tripole jumper *valuelist* settings can be indicated as "1" for "ON" (jumper installed in upper or right position), "0" for "OFF" (jumper installed in lower or left position) "n" for jumper not installed. The tripole jumper settings are not delimited with a space.

Software INIT Statement (Optional)

Syntax:
INIT = SOFTWARE(i) "*parameter*" [| *parameter*]...

The software INIT statement specifies the command line *parameter* that invokes a software command for the selected configuration. The (i) indicates the SOFTWARE(i) statement that contains text to display with the *parameters*. The *parameters* specify an ASCII string that appends to a software command, which specified in the SOFTWARE(i) text. For example, the following configuration file fragment illustrates use of the software INIT statement and SOFTWARE(i) statement that specify an entry into an MS-DOS AUTOEXEC.BAT file:

```
SOFTWARE(1) =
    "This example software initialization
    statement indicates that the NET.EXE
    file with command line parameters must
    be placed in the AUTOEXEC.BAT file: \n\n
    NET.EXE /I=n /D=n where:"

FUNCTION = "Expanded Memory Allocation"
    CHOICE = "4 MB Expanded Memory"
        INIT = SOFTWARE(1) "/I=4  /D=3"
```

4.6.5 Resource Group

A resource description block must have one or more group of resource and initialization statements. The elements of the resource description block are grouped together based on their interdependence. All resource and initialization statements must be in a group. The three types of group are:

- LINK groups, in which selection of any one resource in the group determines the selection of all other resources and initializations in the group.
- COMBINE groups, in which each resource selection is independent, but the initialization is determined by the combination of resource selections.
- FREE groups, in which each resource selection is independent, and the initializations are independent of the resource selections.

The groups begin with a keyword (LINK, COMBINE or FREE) and end at the next group keyword or at the end of the resource description block.

4.6.5.1 LINK Groups

The elements of linked group have a direct relationship with each other. The selection of one resource determines the other resources in the group and the initialization. Each statement in a linked group must have the same number of options. If the first option is chosen for one resource, the configuration utility automatically selects the first option for the other resource statements and the initialization statements. The syntax of a linked group is:

LINK
resource statement
.
.
resource statement
INIT statement
.
.
INIT statement

The following example illustrates the use of a linked group that provides selection of the interrupt or DMA channel. The user (or configuration utility) can select the interrupt or the DMA channel, but after making the one selection, the other resource and the initialization must correspond to the same option. An IRQ = 3 selection forces the configuration utility to select DMA = 2 and IOPORT(1) initialization 00001111b. A DMA = 5 selection forces the configuration utility to select IRQ = 4 and IOPORT(1) initialization 11110000b.

```
LINK
    IRQ = 3 | 4
    DMA = 2 | 5
    INIT = IOPORT(1) 00001111b | 11110000b
```

4.6.5.2 COMBINE Groups

The elements of combined groups have an indirect relationship with each other. Each resource selection is independent, but the initialization is directly determined by the combination of resource selections. The syntax of a linked group is:

```
COMBINE
    resource statement
    .
    .
    resource statement
    INIT statement
    .
    .
    INIT statement
```

The following example illustrates the use of a combined group that provides selection of a memory size and starting address. The user (or configuration utility) can select any memory size and starting address, and the configuration utility automatically selects the initialization that corresponds to the selected memory size and starting address. The table after the example lists the initialization value for each possible combination.

```
COMBINE
    MEMORY = 4M | 8M        ;Memory size
    ADDRESS = 1M | 4M       ;Starting address
    INIT = IOPORT(2) 00001111b | 01001111b | 10001111b | 11001111b
```

Memory Size	Starting Address	Port Initialization
4M	1M	00001111b
4M	4M	01001111b
8M	1M	10001111b
8M	4M	11001111b

The following example illustrates the use of a combined group in which the starting address selection and the initialization use a range with a step. The user (or configuration utility) can select any memory size and starting address, and the configuration utility automatically selects the initialization that corresponds to the selected memory size and starting address. The table after the example lists the initialization value for each possible combination.

```
COMBINE
    MEMORY = 4M | 8M | 12M
    ADDRESS = 4M-256M STEP = 4M
    INIT = IOPORT(1) 00000000b-10111111b
```

Memory Size	Starting Address	INIT Value
4M	4M	00000000b
4M	8M	00000001b
4M	12M	00000010b
4M	16M	00000011b
.	.	.
.	.	.
4M	244M	00111100b
4M	248M	00111101b
4M	252M	00111110b
4M	256M	00111111b
.	.	.
.	.	.
12M	244M	10111100b
12M	248M	10111101b
12M	252M	10111110b
12M	256M	10111111b

The following COMBINE fragment and INIT table illustrates the initialization value assignment sequence:

```
COMBINE
    RESOURCE1 1 | 2 | 3
    RESOURCE2 1 | 2 | 3
    RESOURCE3 1 | 2 | 3
    INIT = 00001b-11011b
```

INIT Value	RESOURCE1 Part of Combination	RESOURCE2 Part of Combination	RESOURCE3 Part of Combination
00001b	RESOURCE1 option 1	RESOURCE2 option 1	RESOURCE3 option 1
00010b	RESOURCE1 option 1	RESOURCE2 option 1	RESOURCE3 option 2
00011b	RESOURCE1 option 1	RESOURCE2 option 1	RESOURCE3 option 3
00100b	RESOURCE1 option 1	RESOURCE2 option 2	RESOURCE3 option 1
00101b	RESOURCE1 option 1	RESOURCE2 option 2	RESOURCE3 option 2
00110b	RESOURCE1 option 1	RESOURCE2 option 2	RESOURCE3 option 3
00111b	RESOURCE1 option 1	RESOURCE2 option 3	RESOURCE3 option 1
01000b	RESOURCE1 option 1	RESOURCE2 option 3	RESOURCE3 option 2
01001b	RESOURCE1 option 1	RESOURCE2 option 3	RESOURCE3 option 3
01010b	RESOURCE1 option 2	RESOURCE2 option 1	RESOURCE3 option 1
01011b	RESOURCE1 option 2	RESOURCE2 option 1	RESOURCE3 option 2
01100b	RESOURCE1 option 2	RESOURCE2 option 1	RESOURCE3 option 3
01101b	RESOURCE1 option 2	RESOURCE2 option 2	RESOURCE3 option 1
01110b	RESOURCE1 option 2	RESOURCE2 option 2	RESOURCE3 option 2
01111b	RESOURCE1 option 2	RESOURCE2 option 2	RESOURCE3 option 3
10000b	RESOURCE1 option 2	RESOURCE2 option 3	RESOURCE3 option 1
10001b	RESOURCE1 option 2	RESOURCE2 option 3	RESOURCE3 option 2
10010b	RESOURCE1 option 2	RESOURCE2 option 3	RESOURCE3 option 3
10011b	RESOURCE1 option 3	RESOURCE2 option 1	RESOURCE3 option 1
10100b	RESOURCE1 option 3	RESOURCE2 option 1	RESOURCE3 option 2
10101b	RESOURCE1 option 3	RESOURCE2 option 1	RESOURCE3 option 3
10110b	RESOURCE1 option 3	RESOURCE2 option 2	RESOURCE3 option 1
10111b	RESOURCE1 option 3	RESOURCE2 option 2	RESOURCE3 option 2
11000b	RESOURCE1 option 3	RESOURCE2 option 2	RESOURCE3 option 3
11001b	RESOURCE1 option 3	RESOURCE2 option 3	RESOURCE3 option 1
11010b	RESOURCE1 option 3	RESOURCE2 option 3	RESOURCE3 option 2
11011b	RESOURCE1 option 3	RESOURCE2 option 3	RESOURCE3 option 3

4.6.5.3 Free Groups

The elements of free-form groups have no relationship with each other. Each resource selection is independent and the initializations are independent of the resource selections. The syntax of a free-form group is as follows:

FREE
resource statements
INIT statements

The following example illustrates the use of a free-form group in which IRQ 2, 3, 4, or 5 can be selected. The IRQ selection is independent of all other resource declarations. The example does not include any IRQ initialization.

```
FREE
    IRQ = 2 | 3 | 4 | 5
```

4.6.6 PORTVAR(j) Variable

Syntax:
IOPORT(i) = PORTVAR(j)
combined with:
Portvar(j) = *address*

The variable, PORTVAR(j), can be used to modify an IOPORT(i) address based on a configuration selection. Each variable must have a separate PORTVAR(j) statement with a different identifier, "i". The "i" can be any positive integer value from 1 to 32767. The PORTVAR(j) variable replaces the address portion of the IOPORT(i) statement. The configuration utility assigns an address to the IOPORT(i) based on a PORTVAR(j) assignment statement within a choice or subchoice statement block.

The following configuration file segment illustrates the use of PORTVAR(j) to initialize a serial port interrupt. The example indicates an initialization value 00000001b is written to port address 3F9h for a COM1 selection or written to port address 2F9h for a COM2 selection. The configuration utility replaces the PORTVAR(3) variable with the port address (3F9h or 2F9h) based on the CHOICE selected.

```
IOPORT(1) = PORTVAR(3)
FUNCTION = "Serial Port"
    CHOICE = "COM1"
        PORTVAR(3) = 3F9h
        INIT = IOPORT(1) 00000001b
    CHOICE = "COM2"
        PORTVAR(3) = 2F9h
        INIT = IOPORT(1) 00000001b
```

4.7 System Board Configuration File

System board configuration files must supply additional information not required by expansion boards to the configuration utility. This information includes the amount of nonvolatile memory available, the number of expansion slots on the system board, the power available at each slot, and the size and type of each expansion slot. The system description block supplies the additional information.

4.7.1 Board Identification Block

The board identification block for system boards uses the same syntax as an expansion board identification block. The CATEGORY statement must equal "sys" and the SLOT statement must equal EMB(0). The syntax of the board identification block is:

```
BOARD
    ID = "7 character ID"
    NAME = "system board product name"
    MFR = "system board manufacturer name"
    CATEGORY = "SYS"
    SLOT = EMB(0)
    AMPERAGE = value          ;System board +5V current usage in mA
```

4.7.2 System Description Block

The system description block includes a SYSTEM statement, the amount of nonvolatile memory, and a description of the available slots. The system description block follows the board identification block in the configuration file. The syntax of the system description block is:

```
SYSTEM
    [NONVOLATILE = value] ;Bytes of nonvolatile memory
    [AMPERAGE = value]    ;Total +5V current (mA) from power supply
    [SLOT(1) = ISA8 | ISA16 | EISA | OTH [,"text"] [,"text"]...]
        [LENGTH = value]
        [SKIRT = YES | NO]
        [BUSMASTER = YES | NO]
    .
    .
    .
    SLOT(n) = ISA8 | ISA16 | EISA | OTH
        [LENGTH = value]
        [SKIRT = YES | NO]
```

SYSTEM Statement (Required)

Syntax:
SYSTEM

The SYSTEM statement identifies the beginning of the system description block. The SYSTEM statement follows the board identification block in the configuration file.

NONVOLATILE Statement (Optional)

Syntax:
NONVOLATILE = *value*

The NONVOLATILE statement specifies the total bytes of EISA nonvolatile memory in the system. The NONVOLATILE statement does not include the 64 bytes of ISA compatible nonvolatile memory.

The configuration data for one expansion slot, one virtual device or one embedded device (including the system board--EMB(0)), can use no more than 340 bytes of nonvolatile memory. A slot with a multifunction expansion board installed can use 340 bytes total for all expansion board functions. EISA systems must support at least 340 bytes of nonvolatile memory for each expansion slot, plus nonvolatile memory for the system board functions.

The system board designer can use the following equation to determine the minimum amount of EISA nonvolatile memory required:

Nonvolatile Memory =
(Expansion Slots + System Board + Embedded Devices + Virtual Devices) * 340

Where:

Expansion Slots = number of expansion connectors
A whole number between 1 and 15

System Board
EMB(0)--system board

Embedded devices = number of embedded devices on system board
A whole number between
1 and (15 - Physical Slots)

Virtual devices = number of system board virtual devices
Virtual devices ≥ 1

The following example illustrates the nonvolatile memory calculation for a system board with 1 embedded device, 8 expansion connectors and 2 virtual devices:

Assumptions:

System Board	1
Physical Slots =	8
Embedded devices =	1
Virtual devices =	2
Total =	12

Minimum Nonvolatile Memory = 12 * 340 = 4080 bytes

AMPERAGE Statement (Optional)

Syntax:
[AMPERAGE = *value*]

An AMPERAGE statement specifies the total amount of +5 volt power (in milliamps) available to expansion devices installed on the expansion bus.

value = power supply current

4.7.3 SLOT Statement Block (Optional)

Syntax:
SLOT(i) = ISA8 | ISA16 | EISA | OTH [,"text"] [,"text"]...

The SLOT(i) statement is used to specify an expansion slot as 8-bit ISA (ISA8), 16-bit ISA (ISA16), or 32-bit EISA (EISA). The i represents the slot number.

The SLOT(i) statement does not apply to the system board, embedded devices or virtual devices, when included as part of the system description block.

LENGTH Statement (Optional)

Syntax:
[LENGTH = *value*]

A LENGTH statement can accompany a SLOT(i) statement to specify the maximum length board (a decimal integer in millimeters) that can be installed in the slot.

System boards should include a LENGTH statement. The configuration utility cannot optimize expansion board slot allocation if system boards do not specify the slot lengths. If the LENGTH statement is omitted, the configuration utility assumes the maximum length of 341 millimeters and assigns slot numbers without regard to slot length.

SKIRT Statement (Optional)

Syntax:
[SKIRT = YES | NO]

Each SLOT(i) statement can also be accompanied by a SKIRT statement that specifies whether the slot supports a skirted expansion board. The default is YES if the SLOT(i) statement does not have an accompanying SKIRT statement.

BUSMASTER Statement (Optional)

Syntax:
[BUSMASTER = YES | NO]

The BUSMASTER statement specifies whether an EISA slot accepts a bus master expansion board. The slot defaults to BUSMASTER = YES if the BUSMASTER statement is omitted from the slot statement block and the slot is an EISA slot.

4.8 EISA System ROM Operations

EISA system ROM provides the following services to support automatic hardware configuration:

- The EISA system ROM power-up routines use the configuration information stored in nonvolatile memory to initialize the system board and expansion boards.
- The EISA system ROM provides BIOS routines that simplify reading and writing configuration data in nonvolatile memory.

4.8.1 EISA System ROM BIOS Routine Calls

Two BIOS routines are called by the configuration utility to initialize nonvolatile memory. One BIOS routine clears configuration information from nonvolatile memory and the other stores configuration information in nonvolatile memory. The BIOS routines are part of the INT15 handler and have the following call interface:

Clear Nonvolatile Memory
INT 15h, AH = D8h, AL = 02h (or 82h)

Write Nonvolatile Memory
INT 15h, AH = D8h, AL = 03h (or 83h)

Device drivers and the power-up BIOS routines use two other BIOS routine to retrieve configuration information from nonvolatile memory. One BIOS routine returns a subset of the configuration information stored in nonvolatile memory for one expansion board. The other routine returns all the configuration information about one expansion board function. The BIOS routines are called through the INT 15h handler with the following call interface:

Read slot configuration information
INT 15h, AH = D8h, AL = 00h (or 80h)

Read function configuration information
INT 15h, AH = D8h, AL = 01h (or 81h)

The BIOS routines are bimodal (real or protected mode) and can be called for execution as 32- or 16-bit code. Protected mode execution is accomplished by simulating an INT 15h instruction (push flags, call far) to the address pointed to by the INT 15h vector (0000:0054h). If INT 15h no longer points to the system ROM, then the industry standard entry point for INT 15h, F000:F859h, can be called directly. The INT 15h BIOS routines require 1536 bytes allocated from the stack for temporary RAM variables.

Protected mode operating systems that can create a code segment descriptor can call the INT 15h BIOS routines by creating a descriptor that has a base address of F0000h and executing a far call to the offset address of the industry standard entry point. The code segment descriptor must have a limit of FFFFh, and must have I/O privilege (current privilege level of code segment being executed must be equal to or less than IOPL). The code segment descriptor can have a D-bit of 0h (16-bit addressing and operands) or 1h (32-bit addressing and operands). The address segment D-bit can be set to 0h or 1h (indicating 16- or 32-bit data size) independent of the code segment D-bit setting.

A code segment other than F0000h may be used as long as it includes the 64 Kbytes starting at F0000h and has I/O privilege (current privilege level of code segment being executed must be equal to or less than IOPL).

The INT 15h system ROM BIOS routines adhere to the following conventions:

- Do not perform any segment register-dependent operations (all branch instructions are relative to the instruction pointer)
- Do not change the segment registers (including the code segment)
- Return to the calling routine with the interrupt flag unmodified
- Do not use privileged instructions (LMSW, LSL, etc.)
- Do not write data using a code segment (CS) override

4.8.1.1 Identify System Board Type

A device driver can identify an EISA system board by detecting the upper case ASCII string "EISA" at memory address F000:FFD9h through F000:FFDCh.

4.8.1.2 Read Slot Configuration Information, INT 15h, AH=D8h, AL=00h (or 80h)

This BIOS routine reads a subset of the configuration information for one expansion board or the system board from nonvolatile memory. The BIOS routine returns a summary that includes all functions of the expansion board.

INT 15h, AH=D8h, AL=00h (or 80h)

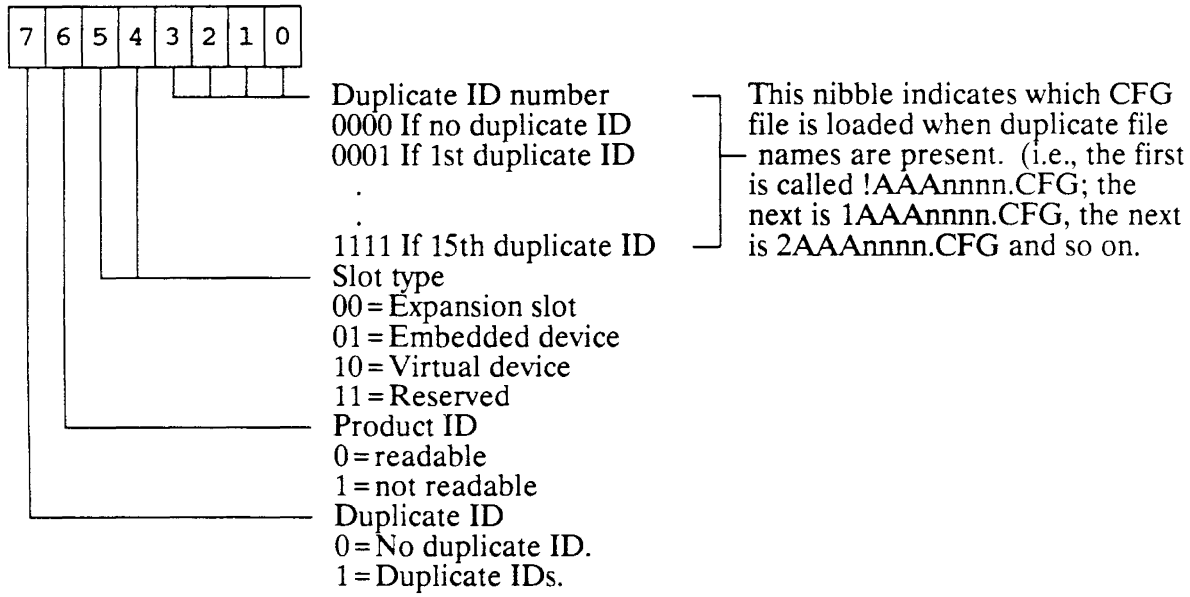
INPUT:

AH = 0D8h
AL = 00h (If CS specifies 16-bit addressing)
AL = 80h (If CS specifies 32-bit addressing)
CL = Slot Number (including embedded and virtual devices)
- 0 System board
- 1 Slot 1
- 2 Slot 2
- n Slot n

OUTPUT:

AH =
- 00h Successful completion (carry flag = 0)
- 80h Invalid slot number (carry flag = 1)
- 82h Nonvolatile memory corrupt (carry flag = 1)
- 83h Empty slot (carry flag = 1)
- 86h Invalid BIOS routine call (carry flag = 1)
- 87h Invalid system configuration (carry flag = 1)

AL =



BH = Major revision level of configuration utility
BL = Minor revision level of configuration utility
CH = Checksum (MSByte) of configuration file
CL = Checksum (LSByte) of configuration file
DH = Number of device functions
DL = Combined function information byte
- Bit 7: Reserved (0)
- Bit 6: Reserved (0)
- Bit 5: Slot has one or more port initialization entries
- Bit 4: Slot has one or more port range entries
- Bit 3: Slot has one or more DMA entries
- Bit 2: Slot has one or more interrupt (IRQ) entries
- Bit 1: Slot has one or more memory entries
- Bit 0: Slot has one or more function type definitions

DI and SI = Four byte compressed ID
- DI (lsb) = Byte 0
- DI (msb) = Byte 1
- SI (lsb) = Byte 2
- SI (msb) = Byte 3

4.8.1.3 Read Function Configuration Information, INT 15h, AH=0D8h, AL=01h (or 81h)

This BIOS routine reads all the configuration information for one expansion board function. The BIOS routine transfers the data block that contains the configuration information for the expansion board function to a table in memory. The BIOS routine stores the data block at the starting address pointed to by DS:SI. The table's data structure is defined later in this section.

The caller can execute the "Read Slot Configuration Information" BIOS routine to determine the number of expansion board functions, and execute the "Read Function Configuration Information" BIOS routine to retrieve the data block for each function. The BIOS routine retrieves the function data block indicated by the function number in register CH. The caller can inspect the TYPE and SUBTYPE fields in each data block to identify the function.

INT 15h, AH=0D8h, AL=01h (or 81h)

INPUT:

AH = 0D8h
AL = 01h (If CS specifies 16-bit addressing)
AL = 81h (If CS specifies 32-bit addressing)
CH = Function number to read (0...n-1)
CL = Slot Number (including embedded and virtual slots)
0 = System Board
1 = Slot 1
2 = Slot 2
n = Slot n
DS = Segment for return data buffer
SI = Offset to return data buffer (16-bit call)
ESI = Offset to return data buffer (32-bit call)

OUTPUT:

AH = 00h Successful completion (carry flag = 0)
80h Invalid slot number (carry flag = 1)
81h Invalid function number (carry flag = 1)
82h Nonvolatile memory corrupt; (carry flag = 1)
83h Empty Slot (carry flag = 1)
86h Invalid BIOS routine call (carry flag = 1)
87h Invalid system configuration (carry flag = 1)

Standard Configuration Data Block Structure

The 320-byte data block pointed to by DS:SI contains the configuration information for one expansion board function. The field sizes of the data block are fixed sizes. A configuration file must not specify resources or initializations that cannot fit within this data structure. The 320-byte data block has the following structure:

Four-Byte Compressed ID — Total Bytes = 4, Offset = 00h

Byte 0
- Bit 7 — Reserved (0)
- Bit 6:2 — Character 1
- Bit 1:0 — Character 2

Byte 1
- Bit 7:5 — Character 2
- Bit 4:0 — Character 3

Byte 2
- Bit 7:4 — 1st hex digit of product number
- Bit 3:0 — 2nd hex digit of product number

Byte 3
- Bit 7:4 — 3rd hex digit of product number
- Bit 3:0 — 1-digit product revision number

ID and Slot Information — Total Bytes = 2, Offset = 04h

Byte 0
- Bit 7 - 0= no duplicate ID is present
  1= duplicate is present
- Bit 6 - 0= ID is readable
  1= ID is not readable
- Bit 5:4 - Slot type
  - 00= expansion slot
  - 01= embedded slot
  - 10= virtual slot
  - 11= reserved
- Bit 3:0 - Numeric identifier for duplicate CFG filenames (IDs)
  - 0000 = No duplicate CFG filenames
  - 0001 = 1st duplicate (CFG file 1ACE0105)
  - 0010 = 2nd duplicate (CFG file 2ACE0105)
  - .
  - .
  - 1111 = 15th duplicate (CFG file FACE0105)

Byte 1
- Bit 7 - 0= configuration is complete
  1= configuration is not complete
- Bit 6:2 - Reserved (0)
- Bit 1 - 0= EISA IOCHKERR not supported
  1= EISA IOCHKERR supported
- Bit 0 - 0= EISA ENABLE not supported (expansion board cannot be disabled)
  1= EISA ENABLE not supported (board can be disabled)

CFG File Extension Revision Level — Total Bytes = 2, Offset = 06h

Byte 0 = Minor revision level (0 if no CFG File Extension)
Byte 1 = Major revision level (0 if no CFG File Extension)

Selections — Total Bytes = 26, Offset = 08h

Byte 0 = 1st Selection
Byte 1 = 2nd Selection
.
.
Byte 25 = 26th Selection

Function Information — Total Bytes = 1, Offset = 022h

Byte 0
- Bit 7 - 0 = function is enabled; 1 = function is disabled
- Bit 6 - CFG extension Free-form data
- Bit 5 - Port initialization entry(s) follows
- Bit 4 - Port range entry(s) follows
- Bit 3 - DMA entry(s) follows
- Bit 2 - Interrupt (IRQ) entry(s) follows
- Bit 1 - Memory entry(s) follows
- Bit 0 - Type/subtype ASCII string entry follows

TYPE and SUBTYPE ASCII String — Total Bytes = 80, Offset = 023h

Byte 0 = 1st character of ASCII string
Byte 1 = 2nd character of ASCII string
.
.
Byte 79 = 80th character of ASCII string

For example, TYPE = COM,ASY;COM1 produces:

Byte 0 = C	Start of TYPE String
Byte 1 = O	
Byte 2 = M	
Byte 3 = ,	Delimiter for TYPE string fragments
Byte 4 = A	
Byte 5 = S	
Byte 6 = Y	End of TYPE string
Byte 7 = ;	Delimiter for SUBTYPE string
Byte 8 = C	Start of SUBTYPE string
Byte 9 = O	
Byte 10 = M	
Byte 11 = 1	End of SUBTYPE string
Byte 12 = 0	Zero fill to end of field
Byte 13 = 0	
.	
.	
Byte 79 = 0	

Memory Configuration

Total Bytes = 63
Offset = 073h

Byte 0 = Memory configuration byte
- Bit 7 - 0 = Last entry
  1 = More entries follow
- Bit 6 - Reserved (0)
- Bit 5 - 0 = Not shared memory
  1 = Shared memory
- Bit 4:3 - Memory Type
  - 00 = SYS (base or extended)
  - 01 = EXP (expanded)
  - 10 = VIRtual
  - 11 = OTHer
- Bit 2 - Reserved (0)
- Bit 1 - 0 = Not Cached
  1 = Cached
- Bit 0 - 0 = Read Only (ROM)
  1 = Read/Write (RAM)

Byte 1 = Memory Data Size
- Bit 7:4 - Reserved (0)
- Bit 3:2 - Decode Size
  - 00 = 20
  - 01 = 24
  - 10 = 32
  - 11 = Reserved (0)
- Bit 1:0 Data Size (Access size)
  - 00 = BYTE
  - 01 = WORD
  - 10 = DWORD
  - 11 = Reserved (0)

Byte 2 = LSByte Memory start address (divided by 100h)
Byte 3 = Middle Byte Memory start address
Byte 4 = MSByte Memory start address
Byte 5 = LSByte Memory size (bytes divided by 400h)
Byte 6 = MSByte Memory size (0 in this word means 64M)

Interrupt Configuration

Total Bytes = 14
Offset = 0B2h

Byte 0
- Bit 7 - 0 = Last entry
  1 = More entries follow
- Bit 6 - 0 = Not Shared
  1 = Shared
- Bit 5 - 0 = Edge Triggered
  1 = Level Triggered
- Bit 4 - Reserved (must be 0)
- Bit 3:0 - Interrupt (0-F)

Byte 1 = Reserved (0)

DMA Channel Description — Total Bytes = 8, Offset = 0C0h

Byte 0
- Bit 7 - 0 = Last entry; 1 = More entries follow
- Bit 6 - 0 = Not Shared; 1 = Shared
- Bit 5:3 - Reserved (0)
- Bit 2:0 - DMA Channel Number (0-7)

Byte 1
- Bit 7:6 - Reserved (0)
- Bit 5:4 - DMA Timing
  - 00 - Default (ISA compatible) timing
  - 01 - Type "A" timing
  - 10 - Type "B" timing
  - 11 - BURST (Type "C") timing
- Bit 3:2 - Transfer size
  - 00= 8-bit (byte) transfer
  - 01= 16-bit (word) transfer
  - 10= 32-bit (dword) transfer
  - 11= Reserved
- Bit 1:0 - Reserved (0)

Port I/O Information — Total Bytes = 60, Offset = 0C8h

Byte 0
- Bit 7 - 0 = Last entry; 1 = More entries follow
- Bit 6 - 0 = Not Shared; 1 = Shared
- Bit 5 - Reserved (0)
- Bit 4:0 - Number of Ports (minus 1)
  - 00000 = 1 port
  - 00001 = 2 sequential ports
  - .
  - .
  - .
  - 11111 = 32 sequential ports

Byte 1 = LSByte I/O Port Address
Byte 2 = MSByte I/O Port address

Initialization Data Total Bytes = 60
Offset = 0104h

Byte 0 = Initialization Type
- Bit 7 - 0 = Last entry
  1 = More entries follow
- Bit 6:3 - Reserved (0)
- Bit 2 - Port value or Mask value
  - 0 - Write value to port
  - 1 - Use mask and value
- Bit 1:0 - Type of access
  - 00 - Byte address (8-bit)
  - 01 - Word address (16-bit)
  - 10 - Dword address (32-bit)
  - 11 - Reserved (0)

Byte 1 = LSByte of port I/O address
Byte 2 = MSByte of port I/O address

IF Byte 0, Bit 2 = 0 (no mask), THEN
Bit 1:0 = Port width to write
- 00= Byte 3 = Port value
- 01= Byte 3 = LSByte of port value
  Byte 4 = MSByte of port value
- 10= Byte 3 = LSByte of port value
  Byte 4 = 2nd byte of port value
  Byte 5 = 3rd byte of port value
  Byte 6 = MSByte of port value
- 11= Reserved

IF Byte 0, Bit 2 = 1 (use mask), THEN
Bits 1:0 = Number of bytes/port value/mask
- 00= Byte 3 = Port value
  Byte 4 = Port mask (byte)
- 01= Byte 3 = LSByte of port value
  Byte 4 = MSByte of port value
  Byte 5 = LSByte of Port mask (word)
  Byte 6 = MSByte of Port mask (word)
- 10= Byte 3 = LSByte of port value
  Byte 4 = 2nd byte of port value
  Byte 5 = 3rd byte of port value
  Byte 6 = MSByte of port value
  Byte 7 = LSByte of port mask (dword)
  Byte 8 = 2nd byte of port mask (dword)
  Byte 9 = 3rd byte of port mask (dword)
  Byte 10 = MSByte of port mask (dword)
- 11= Reserved (0)

Free-form Configuration Data Block Structure

When the Free-form data bit is set in the Function Information byte (bit 6), the 320-byte data structure has the following specific format.

Four-Byte Compressed ID — Total Bytes = 4, Offset = 00h

Byte 0
- Bit 7 — Reserved (0)
- Bit 6:2 — Character 1
- Bit 1:0 — Character 2

Byte 1
- Bit 7:5 — Character 2
- Bit 4:0 — Character 3

Byte 2
- Bit 7:4 — 1st hex digit of product number
- Bit 3:0 — 2nd hex digit of product number

Byte 3
- Bit 7:4 — 3rd hex digit of product number
- Bit 3:0 — 1-digit product revision number

ID and Slot Information — Total Bytes = 2, Offset = 04h

Byte 0
- Bit 7 - 0= no duplicate ID is present
  1= duplicate is present
- Bit 6 - 0= ID is readable
  1= ID is not readable
- Bit 5:4 - Slot type
  - 00= expansion slot
  - 01= embedded slot
  - 10= virtual slot
  - 11= reserved (0)
- Bit 3:0 - Numeric identifier for duplicate CFG filenames (IDs)
  - 0000 = No duplicate CFG filenames
  - 0001 = 1st duplicate (CFG file 1ACE0105)
  - 0010 = 2nd duplicate (CFG file 2ACE0105)
  - .
  - .
  - 1111 = 15th duplicate (CFG file FACE0105)

Byte 1
- Bit 7 - 0= configuration is complete
  1= configuration is not complete
- Bit 6:2 - Reserved (0)
- Bit 1 - 0= EISA IOCHKERR not supported
  1= EISA IOCHKERR supported
- Bit 0 - 0= EISA ENABLE not supported (expansion board cannot be disabled)
  1= EISA ENABLE not supported (board can be disabled)

CFG File Extension Revision Level

Total Bytes = 2
Offset = 06h

Byte 0 = Minor revision level (0 if no CFG File Extension)
Byte 1 = Major revision level (0 if no CFG File Extension)

Selections

Total Bytes = 26
Offset = 08h

Byte 0 = 1st Selection
Byte 1 = 2nd Selection
.
.
Byte 25 = 26th Selection

Function Information

Total Bytes = 1
Offset = 022h

Byte 0
- Bit 7 - 0 = function is enabled, 1 = function is disabled
- Bit 6 - CFG extension Free-form data (=1)
- Bit 5 - Port initialization entry(s) follows
- Bit 4 - Port range entry(s) follows
- Bit 3 - DMA entry(s) follows
- Bit 2 - Interrupt (IRQ) entry(s) follows
- Bit 1 - Memory entry(s) follows
- Bit 0 - Type/subtype ASCII string entry follows

TYPE and SUBTYPE ASCII String

Total Bytes = 80
Offset = 023h

Byte 0 = 1st character of ASCII string
Byte 1 = 2nd character of ASCII string
.
.
Byte 79 = 80th character of ASCII string

Freeform Data

Total Bytes = 2 to 205
Offset = 73h

Byte 0 = Length of following data block
Byte 1 = 1st byte of freeform data
.
.
Byte 204 = 204th byte of freeform data

4.8.1.4 Clear Nonvolatile Memory, INT 15h, AH=D8h, AL=02h (or 82h)

This BIOS routine clears all EISA nonvolatile memory locations. The configuration utility uses the "Clear Nonvolatile Memory" BIOS routine Call prior to writing configuration information to nonvolatile memory.

The Clear Nonvolatile Memory BIOS routine does not clear the 64-byte ISA nonvolatile memory.

INT 15h, AH=D8h, AL=02h (or 82h)

INPUT:

AH = D8h
AL = 02h (If CS specifies 16-bit addressing)
AL = 82h (If CS specifies 32-bit addressing)
BH = Configuration utility major revision level
BL = Configuration utility minor revision level

OUTPUT:

AH = 00h Successful completion (carry flag = 0)
84h Error clearing nonvolatile memory (carry flag = 1)
86h Invalid BIOS routine call (carry flag = 1)
88h Configuration utility not supported (carry flag = 1)

If 88h is returned in AH, indicating an unsupported revision of the configuration utility, then the major revision number of the configuration utility that is supported is returned in AL.

4.8.1.5 Write Nonvolatile Memory INT 15h, AH=D8h, AL=03h (or 83h)

This BIOS routine writes configuration information for one slot into EISA nonvolatile memory. The "Write Nonvolatile Memory" BIOS routine also computes a CRC code (or checksum) after each call. The CRC code (or checksum) is a cumulative calculation that includes all data written to nonvolatile memory since the last "Clear Nonvolatile Memory" BIOS routine Call.

The Write Nonvolatile Memory BIOS routine does not write to the 64-byte ISA configuration memory.

INT 15h, AH=D8h, AL=03h (or 83h)

INPUT:

AH = D8h
AL = 03h (If CS specifies 16-bit addressing)
AL = 83h (If CS specifies 32-bit addressing)
CX = Length of data structure (CX = 0 indicates empty slot)
Length includes two bytes for configuration file checksum
DS = Segment of data buffer
SI = Offset of data buffer (16-bit call)
ESI = Offset of data buffer (32-bit call)

OUTPUT:

AH = 00h Successful completion (carry flag =0)
84h Error writing nonvolatile memory (carry flag = 1)
85h Nonvolatile Memory is full, (carry flag = 1)
86h Invalid BIOS routine call (carry flag = 1)

Standard Configuration Data Block Structure

The structure referenced by DS:SI in the Write Nonvolatile Memory BIOS routine CALL for a slot with a single function has the following format:

Four-Byte Compressed ID		Total Bytes = 4
Byte 0		
Bit 7	Reserved (0)	
Bit 6:2	Compressed character 1	
Bit 1:0	Compressed character 2	
Byte 1		
Bit 7:5	Compressed character 2	
Bit 4:0	Compressed character 3	
Byte 2		
Bit 7:4	1st hex digit of product number	
Bit 3:0	2nd hex digit of product number	
Byte 3		
Bit 7:4	3rd hex digit of product number	
Bit 3:0	1-digit product revision number	

ID and Slot Information Total Bytes = 2

Byte 0

Bit 7 - 0= no duplicate ID is present
1= duplicate is present

Bit 6 - 0= ID is readable
1= ID is not readable

Bit 5:4 - Slot type
00= expansion slot
01= embedded slot
10= virtual slot
11= reserved (0)

Bit 3:0 - Numeric identifier for duplicate CFG filenames (IDs)
0000 = No duplicate CFG filenames
0001 = 1st duplicate (CFG file 1ACE0105)
0010 = 2nd duplicate (CFG file 2ACE0105)
.
.
1111 = 15th duplicate (CFG file FACE0105)

Byte 1

Bit 7 - 0= configuration is complete
1= configuration is not complete

Bit 6:2 - Reserved (0)

Bit 1 - 0= EISA IOCHKERR not supported
1= EISA IOCHKERR supported

Bit 0 - 0= EISA ENABLE not supported (expansion board cannot be disabled)
1= EISA ENABLE not supported (board can be disabled)

CFG File Extension Revision Level Total Bytes = 2

Byte 0 = Minor revision level (0 if no CFG File Extension)
Byte 1 = Major revision level (0 if no CFG File Extension)

Function Length Total Bytes = 2

Length does not include these two bytes, or the checksum at the end of nonvolatile memory
Byte 0 = LSB length of following function entry
Byte 1 = MSB length of following function entry

Selections Total Bytes = 2 to 27

Byte 0 = Length of following selections field
Byte 1 = 1st Selection
Byte 2 = 2nd Selection
.
.
Byte 26 = 26th Selection

Function Information Total Bytes = 1

Byte 0
Bit 7 - 0= function is enabled
1= function is disabled
Bit 6 -CFG extension free-form data
Bit 5 - Port initialization entry(s) follows
Bit 4 - Port range entry(s) follows
Bit 3 - DMA entry(s) follows
Bit 2 - Interrupt (IRQ) entry(s) follows
Bit 1 - Memory entry(s) follows
Bit 0 - Type/subtype ASCII string entry follows

TYPE and SUBTYPE ASCII String Total Bytes = 2 to 81

Byte 0 = Length of following ASCII string field
Byte 1 = 1st character of ASCII string
Byte 2 = 2nd character of ASCII string
.
.
Byte 80= 80th character of ASCII string

For example, TYPE = COM,ASY;COM1 produces:
Byte 0 = 0Ch Length of string field
Byte 1 = C Start of TYPE string
Byte 2 = O
Byte 3 = M
Byte 4 = , Delimiter for TYPE string fragments
Byte 5 = A
Byte 6 = S
Byte 7 = Y End of TYPE string
Byte 8 = ; Delimiter for SUBTYPE string
Byte 9 = C Start of SUBTYPE string
Byte 10= O
Byte 11= M
Byte 12= 1 End of SUBTYPE string

Memory Configuration Total Bytes = 7 to 63

Byte 0 = Memory configuration byte
- Bit 7 - 0 = Last entry
  1 = More entries follow
- Bit 6 - Reserved (0)
- Bit 5 - 0 = Not shared memory
  1 = Shared memory
- Bit 4:3 - Memory Type
  00 = SYStem (base or extended)
  01 = EXPanded
  10 = VIRtual
  11 = OTHer
- Bit 2 - Reserved (0)
- Bit 1 - 0 = Not Cached
  1 = Cached
- Bit 0 - 0 = Read Only (ROM)
  1 = Read/Write (RAM)

Byte 1 = Memory Data Size
- Bit 7:4 - Reserved (0)
- Bit 3:2 - Decode Size
  00 = 20
  01 = 24
  10 = 32
  11 = Reserved (0)
- Bit 1:0 - Data Size (access size)
  00 = BYTE
  01 = WORD
  10 = DWORD
  11 = Reserved (0)

Byte 2 = LSByte Memory start address (divided by 100h)
Byte 3 = Middle Byte Memory start address
Byte 4 = MSByte Memory start address
Byte 5 = LSByte Memory size (bytes divided by 400h)
Byte 6 = MSByte Memory size

Interrupt Configuration Total Bytes = 2 to 14

Byte 0
- Bit 7 - 0 = Last entry
  1 = More entries follow
- Bit 6 - 0 = Not Shared
  1 = Shared
- Bit 5 0 = Edge Triggered
  1 = Level Triggered
- Bit 4 - Reserved (0)
- Bit 3:0 - Interrupt (0-F)

Byte 1 = Reserved (0)

DMA Channel Description Total Bytes = 2 to 8

Byte 0
- Bit 7 - 0 = Last entry
  1 = More entries follow
- Bit 6 - 0 = Not Shared
  1 = Shared
- Bit 5:3 - Reserved (0)
- Bit 2:0 - DMA Channel Number (0-7)

Byte 1
- Bit 7:6 - Reserved (0)
- Bit 5:4 - DMA Timing
  - 00 - Default (ISA compatible) timing
  - 01 - Type "A" timing
  - 10 - Type "B" timing
  - 11 - BURST (Type "C") timing
- Bit 3:2 - Transfer size
  - 00= 8-bit (byte) transfer
  - 01= 16-bit (word) transfer
  - 10= 32-bit (dword) transfer
  - 11= Reserved (0)
- Bit 1:0 - Reserved (0)

Port I/O Information Total Bytes = 3 to 60

Byte 0
- Bit 7 - 0 = Last entry
  1 = More entries follow
- Bit 6 - 0 = Not Shared
  1 = Shared
- Bit 5 - Reserved (0)
- Bit 4:0 - Number of Ports (minus 1)
  - 00000 = 1 port
  - 00001 = 2 sequential ports
  - .
  - .
  - .
  - 11111 = 32 sequential ports

Byte 1 = LSByte I/O Port Address
Byte 2 = MSByte I/O Port address

Initialization Data Total Bytes = 4 to 60

Byte 0 = Initialization Type
- Bit 7 - 0 = Last entry
  1 = More entries follow
- Bit 6:3 - Reserved (0)
- Bit 2 - Port value or Mask value
  0 - Write value to port
  1 - Use mask and value
- Bit 1:0 - Type of access
  00 - Byte address (8-bit)
  01 - Word address (16-bit)
  10 - Dword address (32-bit)
  11 - Reserved (0)

Byte 1 = LSByte of port I/O address
Byte 2 = MSByte of port I/O address

If Byte 0, Bit 2 = 0 (no mask), THEN
Bit 1:0 = Port width to write
- 00= Byte 3 = Port value
- 01= Byte 3 = LSByte of port value
  Byte 4 = MSByte of port value
- 10= Byte 3 = LSByte of port value
  Byte 4 = 2nd byte of port value
  Byte 5 = 3rd byte of port value
  Byte 6 = MSByte of port value
- 11= Reserved (0)

If Byte 0, Bit 2 = 1 (use mask), THEN
Bits 1:0 = Number of bytes/port value/mask
- 00= Byte 3 = Port value
  Byte 4 = Port mask (byte)
- 01= Byte 3 = LSByte of port value
  Byte 4 = MSByte of port value
  Byte 5 = LSByte of Port mask (word)
  Byte 6 = MSByte of Port mask (word)
- 10= Byte 3 = LSByte of port value
  Byte 4 = 2nd byte of port value
  Byte 5 = 3rd byte of port value
  Byte 6 = MSByte of port value
  Byte 7 = LSByte of port mask (dword)
  Byte 8 = 2nd byte of port mask (dword)
  Byte 9 = 3rd byte of port mask (dword)
  Byte 10= MSByte of port mask (dword)
- 11= Reserved (0)

Configuration Data for 2nd function . . .	Function length . . .
Configuration Data for 3rd function . . .	Function length . . .
Configuration Data for nth function . . .	Function length for nth function = 00
Configuration File Checksum	Total Bytes = 2
Byte 1 = MSByte of configuration file checksum Byte 0 = LSByte of configuration file checksum	

Free-formConfiguration Data Block Structure

When the free-form data bit is set in the Function Information byte (bit 6), the data block pointed to by DS:SI has the following specific format.

Four-Byte Compressed ID		Total Bytes = 4
Byte 0		
Bit 7	Reserved (0)	
Bit 6:2	Compressed character 1	
Bit 1:0	Compressed character 2	
Byte 1		
Bit 7:5	Compressed character 2	
Bit 4:0	Compressed character 3	
Byte 2		
Bit 7:4	1st hex digit of product number	
Bit 3:0	2nd hex digit of product number	
Byte 3		
Bit 7:4	3rd hex digit of product number	
Bit 3:0	1-digit product revision number	

ID and Slot Information Total Bytes = 2

Byte 0

Bit 7 - 0= no duplicate ID is present
1= duplicate is present

Bit 6 - 0= ID is readable
1= ID is not readable

Bit 5:4 - Slot type
00= expansion slot
01= embedded slot
10= virtual slot
11= reserved (0)

Bit 3:0 - Numeric identifier for duplicate CFG filenames (IDs)
0000 = No duplicate CFG filenames
0001 = 1st duplicate (CFG file 1ACE0105)
0010 = 2nd duplicate (CFG file 2ACE0105)
.
.
1111 = 15th duplicate (CFG file FACE0105)

Byte 1

Bit 7 - 0= configuration is complete
1= configuration is not complete

Bit 6:2 - Reserved (0)

Bit 1 - 0= EISA IOCHKERR not supported
1= EISA IOCHKERR supported

Bit 0 - 0= EISA ENABLE not supported (expansion board cannot be disabled)
1= EISA ENABLE not supported (board can be disabled)

CFG File Extension Revision Level Total Bytes = 2

Byte 0 = Minor revision level (0 if no CFG File Extension)
Byte 1 = Major revision level (0 if no CFG File Extension)

Selections	Total Bytes = 2 to 27
Byte 0 = Length of following selections field Byte 1 = 1st Selection Byte 2 = 2nd Selection . . Byte 26 = 26th Selection	

Function Information	Total Bytes = 1
Byte 0 Bit 7 - 0= function is not disabled 1= function is disabled Bit 6 -CFG extension free-form data (=1) Bit 5 - Port initialization entry(s) follows Bit 4 - Port range entry(s) follows Bit 3 - DMA entry(s) follows Bit 2 - Interrupt (IRQ) entry(s) follows Bit 1 - Memory entry(s) follows Bit 0 - Type/subtype ASCII string entry follows	

TYPE and SUBTYPE ASCII String	Total Bytes = 2 to 81
Byte 0 = Length of following ASCII string field Byte 1 = 1st character of ASCII string Byte 2 = 2nd character of ASCII string . . Byte 80= 80th character of ASCII string	

Free-form Data	Total Bytes = 2 to 205
Byte 0 = Length of following data block Byte 1 = 1st byte of freeform data . . Byte 204 = 204th byte of freeform data	

The following paragraphs specify the data structure fields that are not obvious from the configuration language specification.

Configuration File Checksum

The configuration file checksum is a 16-bit logical (modula 64k) sum of ASCII values in the configuration file.

Configuration File Extension Revision Level

The Configuration File Extension revision level specifies the revision number for the overlay file. The configuration file extension checks the revision number when reconstructing the user displays from a backup copy of the configuration (a configuration saved to a disk file) or from reading nonvolatile memory (backtracking).

Function Length

Specifies the number of nonvolatile memory bytes that contain the function information. The two bytes of function length are not included in the count. The configuration file checksum bytes are not included.

Selections field

Nonvolatile memory contains numbers that indicate the function choices and resource alternatives selected during configuration. The configuration utility uses the selection numbers during a reconfiguration to display the default selections to a user (backtrack).

The backtrack routine reads selection numbers from nonvolatile memory for display as the defaults. Selections from all group types (COMBINE, LINK or FREE) have a selection number, even if there is only one resource to select.

Note 1: Each memory resource selection number requires one word of storage. Other resource selection numbers require one byte each.

Note 2: The selection numbers for a Function include the selections for its Subfunctions.

1. Selection number of Choice in the Function or Subfunction.

2. Selection number of Subchoice (if it exists).

3. Selection number of alternate choice in each group for LINK and COMBINE groups or the selection number for each resource in a FREE group.

4. When a Read Function Configuration Information BIOS routine call is issued, the information in Subfunctions are included in the Function. Thus the selection numbers in Subfunctions are grouped with the Function selection numbers.

   These selection numbers are repeated as needed.

EXAMPLE #1:

```
CFG FILE
FUNCTION = ...
    CHOICE(0) = ...0                    ;CHOICE 0 was chosen
        LINK
            Resource1 = 1 | 2           ;2nd alternate was chosen (1)
            Resource2 = 3 | 4
        FREE
            Resource3 = 5 | 7           ;2nd alternate was chosen (1)
            Resource4 = 6               ;1st resource was chosen (0)
            Resource5 = 7 | 8 | 9       ;3rd alternate was chosen (2)
    CHOICE(1) = ...
        ...
```

4.8.2 Initializing Nonvolatile Memory

The EISA configuration utility begins initializing nonvolatile memory by issuing the "Clear Nonvolatile Memory" BIOS routine Call that clears the configuration information from EISA nonvolatile memory. The configuration utility then issues repetitive "Write Nonvolatile Memory" BIOS routine Calls to load all EISA system board, embedded device, virtual device, and expansion board configuration data.

The configuration utility first builds a data structure that includes the configuration information for slot 0 (the system board), then executes the "Write Nonvolatile Memory" BIOS routine Call with a pointer to the data structure. The configuration utility repeats the sequence for each slot and device.

4.8.3 Power-up Initialization of EISA Systems

EISA systems must assume a reset condition after power-up reset occurs. Expansion boards can decode only the slot-specific I/O addresses used for initialization and must assume a disabled state.

The BIOS power-up routine performs the following steps to initialize EISA systems:

- It confirms the validity of configuration information in nonvolatile memory. If the configuration information is not valid the power-up routine aborts automatic configuration, issues an error message, then continues the power-up sequence.
- It compares the EISA product ID and slot information in nonvolatile memory with the actual installed hardware to confirm that the configuration has not changed. If the expansion board installed in a slot does not match the information stored in nonvolatile memory the power-up routine aborts initialization.
- It uses the configuration data to initialize the system board, expansion boards, embedded devices and virtual devices.
- It enables the system board, expansion boards, embedded devices and virtual devices for operation.

The system ROM automatically determines the I/O port address and initialization values and programs the following registers:

Interrupt controller edge/level register
DMA controller (Extended Mode Register)
    DMA channel cycle timing
    DMA data size and addressing mode
DMA controller (DMA Command Register)
    DRQ and DAK* assert level (high/low)
    Fixed or rotating priority scheme

The power-up routine initializes the system board and all EISA expansion boards before determining system memory size or searching for I/O devices (such as printer ports, communications ports, VGA, etc.). Since memory boards that have optional configuration as system or expanded memory are included in the memory size determination, neither an option ROM nor an operating system-dependent device driver is required.

4.8.4 Slot Initialization Sequence

The EISA power-up routine initializes expansion slots, embedded devices, virtual devices, and the system board configuration registers. The initialization takes place during every cold or warm boot.

The flow chart in Figure 103 specifies the EISA slot initialization sequence:

Figure 103 - Power-Up Slot Initialization

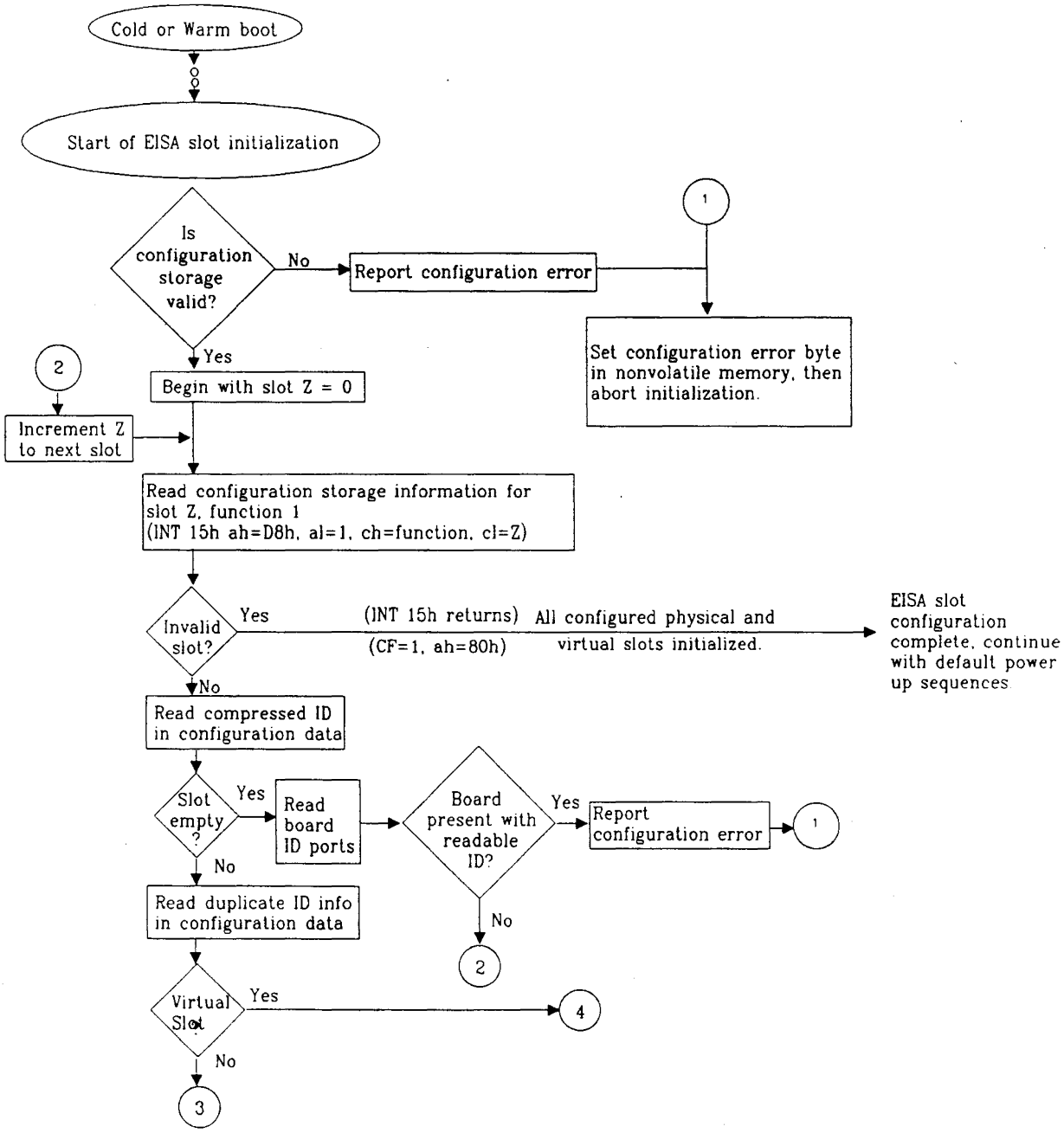
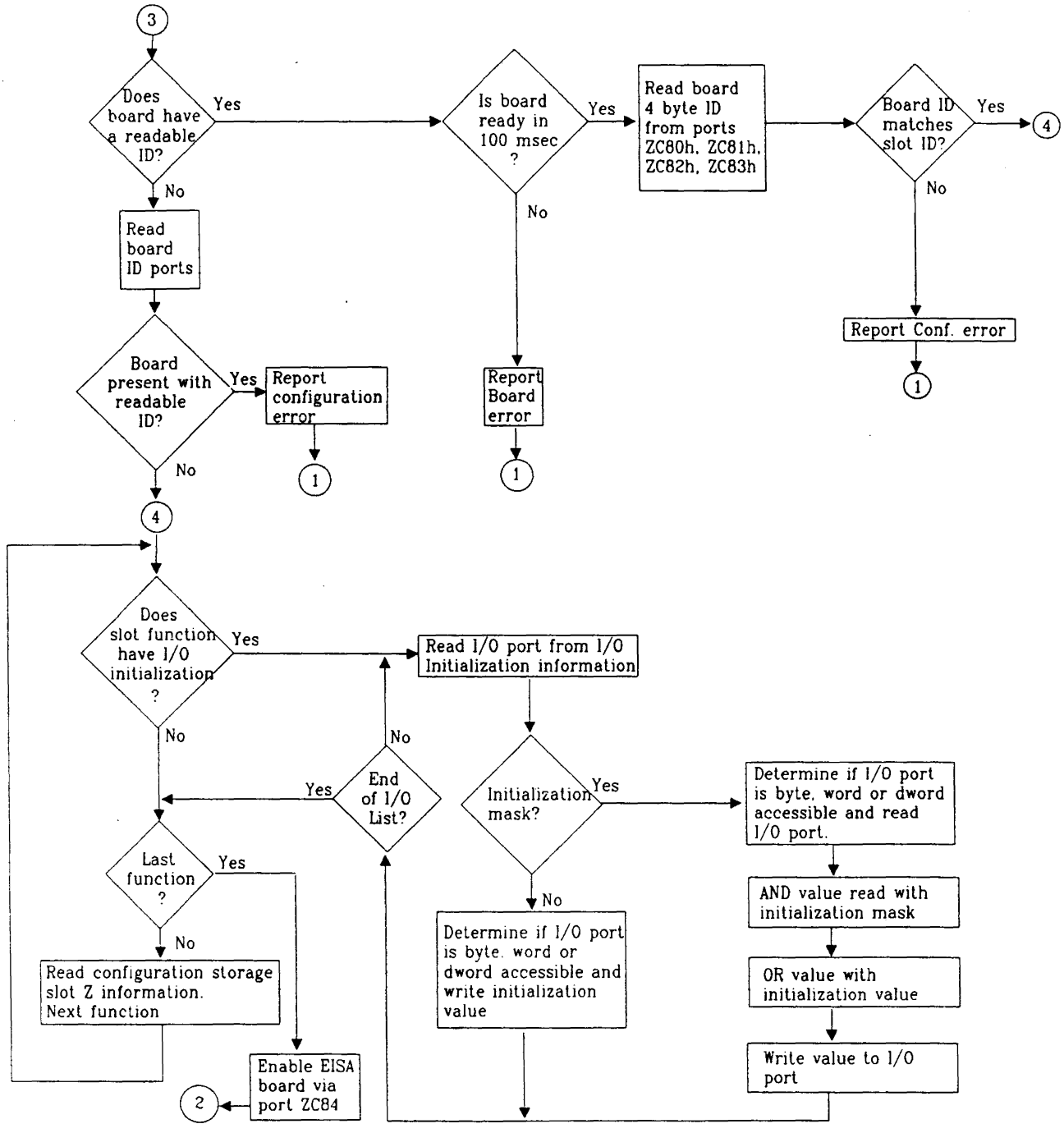


Figure 103 - Power-Up Slot Initialization (Continued)



The system ROM power-up routine can initialize critical devices in any order necessary to bring the system up. The power-up routine must then initialize devices sequentially by slot number and function number.

The power-up routine initializes critical devices first, then proceeds to initialize the EISA system board, EMB(0). The power-up routine then begins expansion board initialization beginning with expansion slot 1. The power-up routine issues a "Read Function Configuration Information" BIOS routine Call for slot 1, function 1. The power-up routine checks the product ID field of the data block returned for slot 1 function 1 to determine if the slot was configured as empty or with an expansion board installed.

If nonvolatile memory indicates the slot has an expansion board installed, and the readable ID bit indicates a readable ID, the power-up routine performs the I/O read to confirm that the product ID matches nonvolatile memory. If the product ID read operation indicates a not ready condition on the first try, the power-up routine waits 100 milliseconds, then retries the ID read. The power-up routine issues an error message if the ID read still indicates a not ready condition after the 100 millisecond delay, then aborts initialization.

If the product ID matches nonvolatile memory, the power-up routine performs the initialization by setting the I/O ports to the values indicated in nonvolatile memory and programming the system board controllers to properly allocate the system resources required by the expansion board.

After initializing each of the expansion board functions and the required system resources, the power-up routine enables the expansion board, then issues the "Read Function Configuration Information" BIOS routine Call for slot 2 function 1. The power-up routine continues the process until all functions in all expansion slots, embedded slots and virtual slots are configured.

The power-up routine does not initialize installed EISA or ISA expansion boards that do not have configuration information stored in nonvolatile memory.

4.8.5 Error Handling During Slot Initialization

Several error conditions can arise during slot initialization.

If an expansion board indicates a not ready condition when its product ID is read, the power-up routine waits 100 ms then retries the product ID read. If the expansion board still indicates a not ready condition an appropriate error is displayed and the power-up routine continues EISA expansion board initialization with the next slot.

If the ID of the EISA expansion board does not match the contents of nonvolatile memory then an appropriate error is displayed and the power-up routine continues EISA expansion board initialization with the next slot.

If nonvolatile memory indicates the presence of an EISA board with an ID and no matching board is found then an appropriate error is displayed and the power-up routine continues EISA expansion board initialization with the next slot.

If the ID of a slot is tagged not readable in the nonvolatile memory information then the power up routines attempt to read a valid ID from the slot being initialized. If a valid ID is read from the slot then an appropriate error is displayed and the power-up routine continues EISA expansion board initialization with the next slot.

If the nonvolatile memory information indicates that a slot is empty and a valid ID is read from the slot, then an appropriate error is displayed and the power-up routine continues EISA expansion board initialization with the next slot.

An error is displayed if nonvolatile memory slot information does not match what is determined to be in the slots.

An "incomplete configuration" message is displayed if the nonvolatile memory ID and Slot Information incomplete configuration bit is set.

4.8.6 Noncacheable Memory Map Initialization

EISA systems with cache memory can use the data in nonvolatile memory to construct a noncacheable address map. The power-up routine identifies noncacheable memory address ranges from the configuration information in nonvolatile memory. The power-up routine supplies the noncacheable addresses to hardware that disables the memory cache during accesses to the noncacheable addresses.

4.8.7 Writable Memory Map Initialization

EISA systems can use the data in nonvolatile memory to construct a writable address map. The power-up routine identifies RAM and ROM memory address ranges from the configuration information in nonvolatile memory. The power-up routine supplies the RAM and ROM addresses to hardware that disables memory writes during accesses to the ROM addresses.

4.8.8 Slot Mapping Information

The CFG language allows a translation between the EISA slot ID (i.e. the address range used for slot specific I/O) and the physical slot number (i.e. the user readable label for the slot). There is a need for drivers to be able to communicate with the user about devices using these physical slot numbers. Therefore slot ID to physical mapping information must be made available to drivers.

Additionally, due to design constraints, many EISA system vendors will provide systems that provide some EISA slots that cannot accept EISA bus masters. The CFG language allows EISA slots to be specified as either accepting EISA bus masters or not. Other design constraints make it impractical for EISA system vendors to match the bit encoding of Last EISA Bus Master Granted registers (I/O port addresses 0464h and 0465h described in EISA specification section 3.3). In error recovery situations drivers need to be able to convert from the bit encoding of the Last EISA Bus Master Granted registers to EISA slot ID of that bus master. Therefore this mapping must also be made available to drivers.

The CFG language FREEFORM statement provides the mechanism for system vendors to provide this mapping information to drivers. This mechanism can also be use to provide other information about any board in the system.

4.8.8.1 EISA System Information

The EISA System Information specification is a standard method for vendors of EISA systems and adapters to provide to software information about their products that is not otherwise provided in the EISA configuration information stored in nonvolatile memory. This information is stored using the FREEFORM CFG language statement.

When FREEFORM data is used the CFG language TYPE and SUBTYPE statements are the only method of identifying the data. Vendors should supply their unique data with a TYPE string that begins with the three letters of their EISA manufacturer code. The reserved manufacturer code of "ISA" will be used to identify EISA System Information.

Vendors provide EISA System Information with the following group of functions in the CFG file for their products:

```
GROUP = "EISA System Information"
      TYPE = "ISA"
[     FUNCTION = "descriptive text"
          SHOW = NO
          CHOICE = "descriptive text"
                SUBTYPE = "type"
                FREEFORM len, N1, N2, ... , Nlen]...
END GROUP
```

The type is an ASCII string specifying the type of information being provided, len specifies the number of bytes of information being provided, and N1 to Nlen are those bytes of information, len may not be more than 203. Currently only one type is defined:

MAP This information type is valid only for the system board (EISA slot ID 0) and contains the slot mapping information, len must be 30 bytes. If this information type is not present for the system board then the physical slot number is assumed to equal the EISA slot ID and ports 464h & 465h bits 0 to 7 & 0 to 6 are assumed to correspond to EISA slot ID 1 to 8 & 9 to 15. The 30 bytes of information are defined as follows:

N1 to N15 These specify the physical slot number and slot type corresponding to EISA slot ID's 1 to 15. Bits 0 to 4 of each byte is the physical slot number (embedded slots have a physical slot number of 0), Bits 5 to 7 encode the slot type as follows:

765	Slot Type
000	ISA 8-bit slot
001	ISA 16-bit slot
010	EISA, non-bus master slot
011	EISA, bus master slot
100	reserved
101	reserved
110	reserved
111	slot not present (Bits 0 to 4 are undefined).

N16 to N23 These specify the EISA slot number corresponding to bits 0 to 7 of port 0464h, if the bit does not represent a possible bus master then the value is zero (0).

N24 to N30 These specify the EISA slot number corresponding to bits 0 to 6 of port 0465h, if the bit does not represent a possible EISA bus master then the value is zero (0).

4.8.8.2 Instructions for Accessing the EISA System Information

To access the System Information drivers would use the EISA Int 15h Read Function Configuration Information (for a detailed description of this function see EISA specification section 4.8.1.3). In the processor's real address mode the function has this interface:

INPUT:	AX = 0D801h	Read configuration information
	CH = Function number	Varies from 0 to last function number
	CL = 0	System board slot number
	DS:SI	Pointer to 320 byte area for returned data

OUTPUT:	FLAGS:	Carry = 0	Successful completion
		Carry = 1	Error
			AH = Error Code (81h = invalid function number)

DS:SI	Area pointed to by DS:SI has been filled with the following:	
	Offset 22h	Bit 6: 1 = FREEFORM data present 0 = No freeform data
	Offset 23h	ASCII Type string (zero filled to 80 characters), for EISA System Information the first 4 characters will be "ISA" followed by the information type specifier.
	Offset 73h	Length of FREEFORM data.
	Offset 74h	FREEFORM data.

The procedure to extract this information would look like this:

```
For (each function of the desired slot, starting at 0)
{       while (function is valid)
        { if (function has FREEFORM data and Type string equals "ISA;type")
            {   copy FREEFORM data to driver
                break out of while loop
            }
        }
}
```

4.8.8.3 Example of EISA System Information

For example imagine a system board with the following slot configuration:

Physical	Logical	Type
1	7	ISA16
2	1	EISA-master (464h bit 0)
3	2	EISA-master (464h bit 1)
4	3	EISA-master (464h bit 2)
5	4	EISA-master (464h bit 3)
6	5	EISA-master (464h bit 4)
7	6	EISA-non-master
embedded	8	EISA-master (464h bit 5)
embedded	9	EISA-non-master

If the System Information FUNCTION block for this system board only provided type 00 information it would look like this:

```
GROUP = "System Information"
    TYPE = "ISA"
    FUNCTION = "Slot Mapping Information"
        SHOW = NO
        CHOICE = "Slot Mapping Tables"
            SUBTYPE = "MAP"
            FREEFORM 30, 62h, 63h, 64h, 65h, 66h, 47h, 21h,
                60h, 40h, 0E0h, 0E0h, 0E0h, 0E0h, 0E0h, 0E0h,
                1, 2, 3, 4, 5, 8, 0, 0,
                0, 0, 0, 0, 0, 0, 0
END GROUP
```

This data would be returned by Int 15 Read Function Configuration Information call:

```
Offset 22h 40h (bit 6 set)
Offset 23h "ISA;MAP" followed by bytes of zero
Offset 73h 1Eh (total number of FREEFORM data bytes)
Offset 74h 62h, 63h, 64h, 65h, 66h, 47h, 21h, 60h,
           40h, 0E0h, 0E0h, 0E0h, 0E0h, 0E0h, 0E0h, 0E0h,
           1, 2, 3, 4, 5, 8, 0, 0,
           0, 0, 0, 0, 0, 0, 0
```

EISA System I/O Address Map

I/O address Range (hex):	I/O Range Reserved for
0000-00FF	EISA/ISA System board
0100-03FF	ISA expansion boards
0400-04FF	Reserved, EISA System board
controllers	
0500-07FF	Alias of 100h-3FFh
0800-08FF	EISA System board
0900-0BFF	Alias of 100h-3FFh
0C00-0CFF	EISA System board
0D00-0FFF	Alias of 100h-3FFh
1000-10FF	Slot 1
1100-13FF	Alias of 100h-3FFh
1400-14FF	Slot 1
1500-17FF	Alias of 100h-3FFh
1800-18FF	Slot 1
1900-1BFF	Alias of 100h-3FFh
1C00-1CFF	Slot 1
1D00-1FFF	Alias of 100h-3FFh
:	:
:	:
:	:
0z000-0z0FF	Slot 'z'
0z100-0z3FF	Alias of 100h-3FFh
0z400-0z4FF	Slot 'z'
0z500-0z7FF	Alias of 100h-3FFh
0z800-0z8FF	Slot 'z'
0z900-0zBFF	Alias of 100h-3FFh
0zC00-0zCFF	Slot 'z'
0zD00-0zFFF	Alias of 100h-3FFh

4.9.1 Expansion Board Address Decoding

An expansion board that uses the slot-specific I/O ranges may, during I/O cycles, decode address bits LA<11:2>, and BE*<3:0> with AENx negated (low) to address any byte in the slot-specific I/O range. An expansion board that does not need the full I/O address range can decode fewer address bits, depending on the number of ports required. The expansion board must, at a minimum, decode address bits LA<9:8> low and AENx negated (low) to assure that the I/O address does not conflict with the ISA expansion board I/O address range.

See section 2.8.7 in this specification for additional information about EISA I/O decoding and the use of AENx to control slot-specific I/O addressing.

A device driver addresses the expansion board slot-specific addresses with a full 16-bit I/O address. The device driver appends the expansion board address bits, <11:0>, to the high order four bits represented by the hexadecimal slot number to form the 16-bit address, <15:0>.

Slot-specific addresses 0zC80h through 0zC83h are reserved for the product ID. Slot-specific address 0zC84h is reserved for expansion board control bits. All other slot-specific addresses can be used by the expansion board for configuration registers and general purpose I/O.

An EISA expansion board can also use the ISA expansion board I/O ranges, but must assure that the addresses do not conflict with other ISA expansion boards.

The following address ranges are not aliases of ISA expansion board I/O addresses and should be used by an EISA expansion board for I/O registers:

I/O address Range (hex):	I/O Range Reserved for:
1000-10FF 1400-14FF 1800-18FF 1C00-1CFF	Slot 1 Slot 1 Slot 1 Slot 1
2000-20FF 2400-24FF 2800-28FF 2C00-2CFF	Slot 2 Slot 2 Slot 2 Slot 2
: : :	: : :
0z000-0z0FF 0z400-0z4FF 0z800-0z8FF 0zC00-0zCFF	Slot 'z' Slot 'z' Slot 'z' Slot 'z'

The following address ranges are aliases of ISA expansion board I/O addresses:

I/O address Range (hex):	I/O Range Reserved for:
1100-13FF 1500-17FF 1900-1BFF 1D00-1FFF	Alias of 100h-3FFh Alias of 100h-3FFh Alias of 100h-3FFh Alias of 100h-3FFh
2100-13FF 2500-27FF 2900-2BFF 2D00-2FFF	Alias of 100h-3FFh Alias of 100h-3FFh Alias of 100h-3FFh Alias of 100h-3FFh
: : :	: : :
0z100-0z3FF 0z500-0z7FF 0z900-0zBFF 0zD00-0zFFF	Alias of 100h-3FFh Alias of 100h-3FFh Alias of 100h-3FFh Alias of 100h-3FFh

Slot-specific addresses 0zC80h through 0zC83h are reserved for the product ID. Slot-specific address 0zC84h is reserved for expansion board control bits. All other slot-specific addresses can be used by the expansion board for configuration registers and general purpose I/O.

An EISA expansion board that uses the ISA expansion board I/O ranges must assure that the addresses do not conflict with other ISA expansion boards.

4.9.2 Embedded Slot Address Decoding

Embedded slot address decoding works exactly like expansion board address decoding except that the embedded device is integrated onto the system board. The embedded slots use slot numbers that start after the last expansion slot number. For example, the first embedded slot is slot 8 if the EISA system has 7 expansion slots.

4.9.3 System Board Address Decoding

An EISA system board decodes 16 address bits during I/O cycles. The system board configuration registers and controller registers are mapped into the address ranges between 0000h and 0CFFh that are not aliases of ISA expansion board I/O addresses.

The following address ranges are not aliases of ISA expansion board I/O addresses and can be used by an EISA system board for I/O registers:

I/O address Range (hex):	I/O Range Reserved for:
0000-00FF	ISA System board peripherals
0100-03FF	ISA expansion boards
0400-04FF	Reserved - System board controllers
0800-08FF	System board
0C00-0CFF	System board

The following address ranges are aliases of ISA expansion board I/O addresses and cannot be used by an EISA system board:

I/O address Range (hex):	I/O Range Reserved for:
0500-07FF	Alias of 100h-3FFh
0900-0BFF	Alias of 100h-3FFh
0D00-0FFF	Alias of 100h-3FFh

4.10 EISA Product Identifier (ID)

EISA expansion boards, embedded devices and system boards have a four byte product identifier (ID) that can be read from I/O port addresses 0zC80h through 0zC83h (z=0 for the system board). For example, the system board ID can be read from I/O port addresses 0C80h-0C83h and the slot 1 product ID can be read from I/O port addresses 1C80h-1C83h.

The first two bytes (0zC80h and 0zC81h) contain a compressed representation of the manufacturer code. The manufacturer code is a three character code (uppercase, ASCII characters in range "A"-"Z") chosen by the manufacturer and registered with the firm that distributes this specification. System board and expansion board manufacturers follow the same procedure to choose and register their manufacturer code.

The manufacturer code "ISA" should be used to indicate a generic ISA adapter.

The three character manufacturer code is compressed into three 5-bit values so that it can be incorporated into the two I/O bytes at 0zC80h and 0zC81h. The compression procedure is:

Find hexadecimal ASCII value for each letter
ASCII for "A" - "Z": "A" = 41h, "Z" = 5Ah

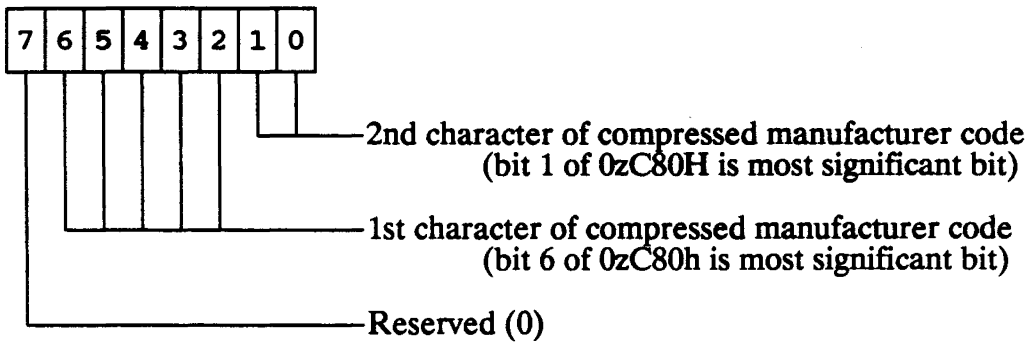
Subtract 40h from each ASCII value
Compressed "A" = 41h-40h = 01h = 0000 0001
Compressed "Z" = 5Ah-40h = 1Ah = 0001 1010

Retain 5 least significant bits for each letter
Discard 3 most significant bits (they are always zero)
Compressed "A" = 00001, Compressed "Z" = 11010

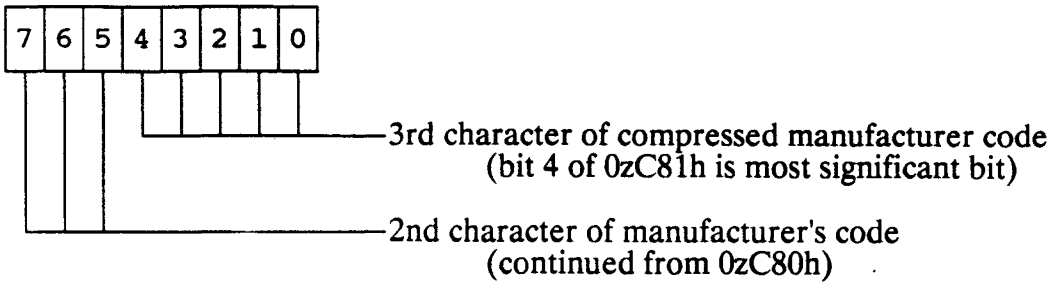
Compressed code = Concatenate "0" and the three 5-bit values
"AZA" = 0 00001 11010 00001 (a 16-bit value)
0zC80h = 00000111, 0zC81h = 01000001

The following figures show the format of the product ID (addresses 0zC80h - 0zC83h):

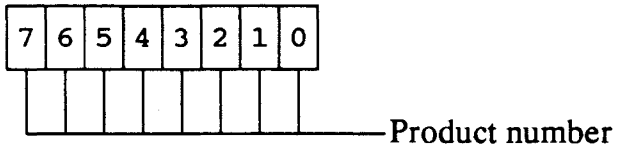
Product ID, 1st byte: 0zC80h



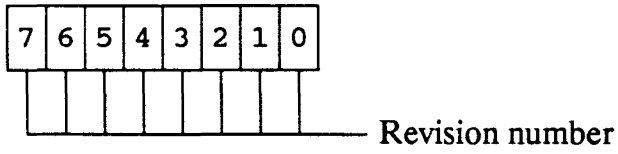
Product ID, 2nd byte: 0zC81h



Product ID, 3rd byte: 0zC82h



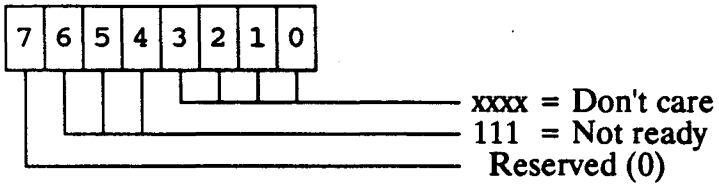
Product ID, 4th byte: 0zC83h



Reporting Not Ready During Access to the Product ID Register

An EISA device that requires a long power-up sequence may report a not ready condition when the power-up routine attempts to read the product ID. The expansion board must complete its power-up sequence and report its product ID within 100 ms after reporting the not ready condition. The expansion board supplies the following data in port 0zC80h to indicate the not ready condition:

Product ID, 1st byte: 0zC80h



An EISA adapter may report not ready on an attempt to access its EISA Product ID Register (0zC80h) for up to 100 milliseconds before it must return a valid Product ID (see preceding section). This time is measured from the first attempt to read the Product ID Register. The EISA adapter must be ready to accept the configuration information stored in the system's nonvolatile memory as soon as it reports a valid Product ID.

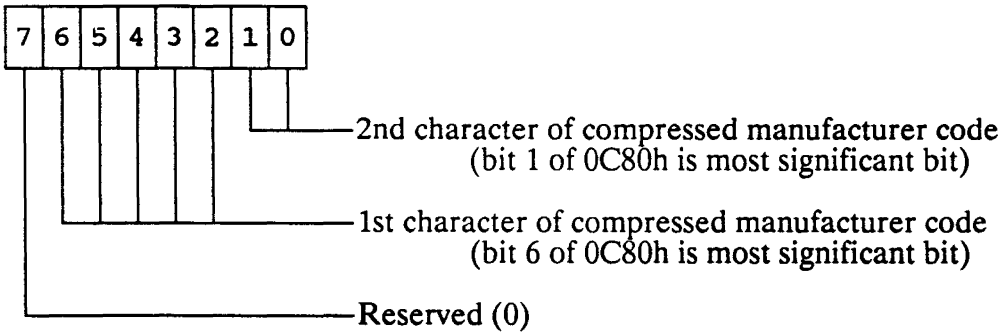
The length of time between power up and first access to the Product ID Register is not specified. Vendors of EISA adaptors should make no assumptions as to a minimum length of time between power up and the first access to the EISA Product ID Register.

The I/O addresses for the system board ID bytes are:

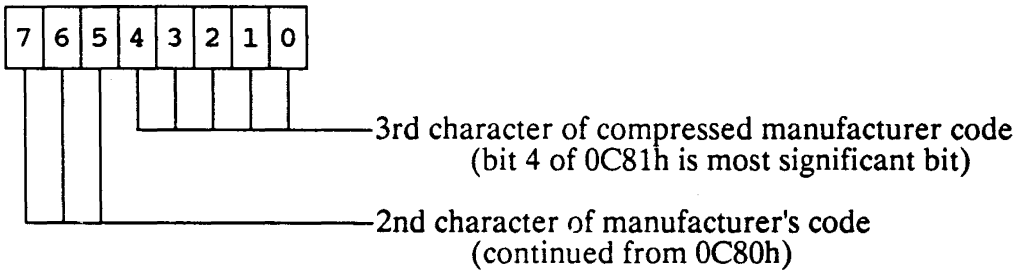
System Board ID, 1st byte: 0C80h
System Board ID, 2nd byte: 0C81h
System Board ID, 3rd byte: 0C82h
System Board ID, 4th byte: 0C83h

The following diagrams show the format of the system board ID.

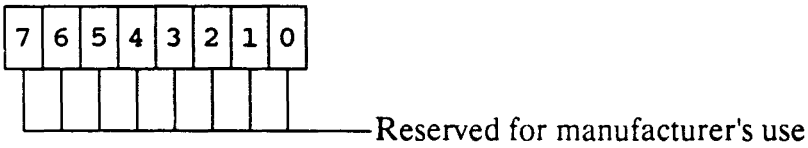
System Board ID, 1st byte: 0C80h



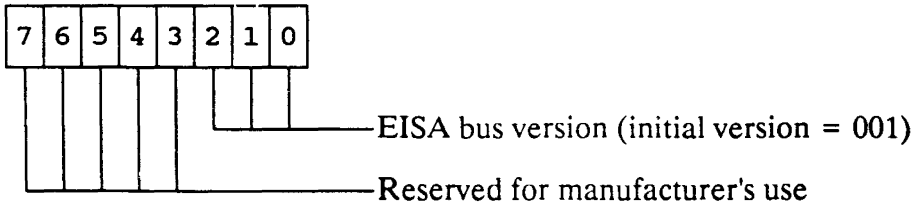
System Board ID, 2nd byte: 0C81h



System Board ID, 3rd byte: 0C82h



System Board ID, 4th byte: 0C83h



Identifying an EISA Expansion Board

1. Write FFh to 0C80h

   The procedure precharges the system board ID register (at I/O address 0C80h).

2. Read 0C80h

   If contents of 0C80h equals FFh, discontinue the identification process, the system board does not have a readable ID.

   If contents of 0C80h does not equal FFh and the most significant bit is a zero: the system board supports a readable ID that can be read at 0C80h-0C83h.

4.10.2 EISA Expansion Board Product ID

The first two bytes of the 4-byte product ID are a compressed representation of the manufacturer code. The third byte represents the product number and the fourth byte represents the product's revision level.

A revised expansion board that requires a modification to its configuration file must have a new product number and revision level in its ID. A revised expansion board that does not require a modification to the configuration file can use its original product number, with a new revision level.

The system ROM power-up routine reads the first four bytes of the ID to compare against the configuration information stored in nonvolatile memory. A match of the hardware ID and the ID stored in nonvolatile memory confirms that the configuration has not changed since system configuration. Bits 3:0 of the fourth byte are not used by the power-up routine.

Device drivers can use the product ID to determine the type of expansion board installed and the revision level.

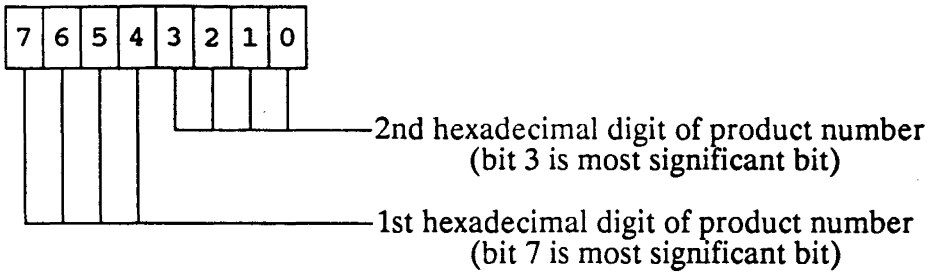
The compressed expansion board manufacturer code has the same format as a system board manufacturer code and is illustrated in the "EISA Product Identifier" section of this specification.

The I/O addresses (where "z" is the slot number) for the product ID bytes are:

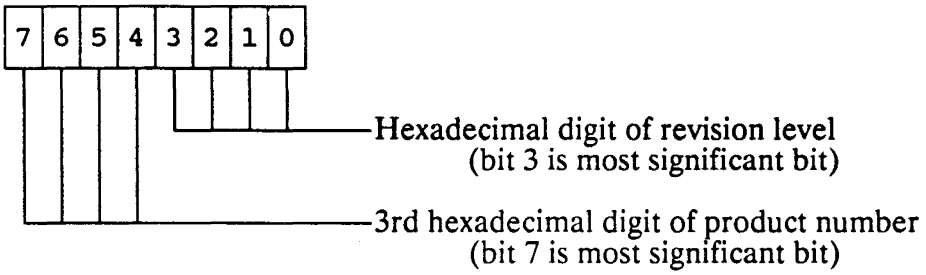
Product ID, 1st byte: 0zC80h
Product ID, 2nd byte: 0zC81h
Product ID, 3rd byte: 0zC82h
Product ID, 4th byte: 0zC83h

The following diagrams illustrate the third and fourth byte of the product ID.

Expansion Board Product ID, 3rd byte: 0zC82h



Expansion Board Product ID, 4th byte: 0zC83h



Identifying an EISA Expansion Board

1. Write FFh to 0zC80h

   The procedure precharges the expansion board ID register (at I/O address 0zC80h).

2. Read 0zC80h

   If contents of 0zC80h equals FFh, discontinue the identification process, the expansion board does not have a readable ID.

   If contents of 0zC80h does not equal FFh and the most significant bit is a zero: the expansion board supports a readable ID that can be read at 0zC80h-0zC83h.

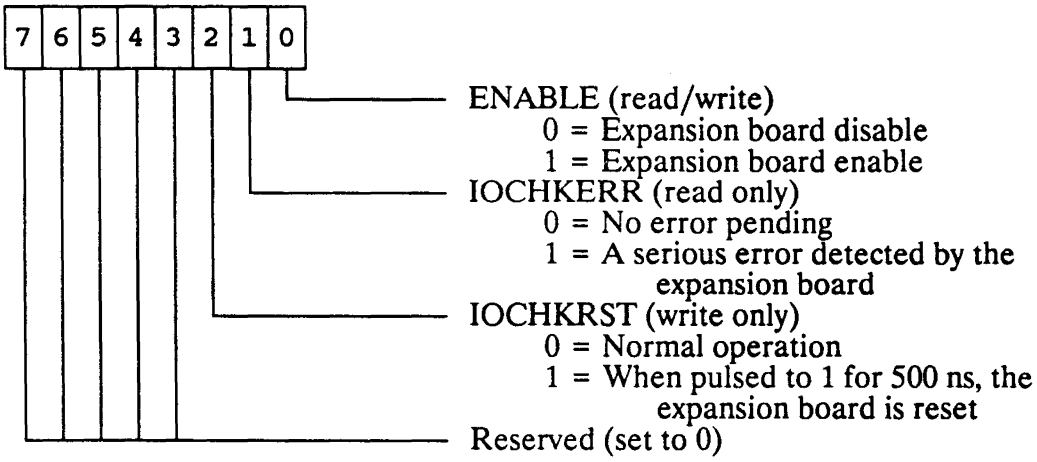
4.10.3 EISA Embedded Devices

The ID of an EISA embedded device has the same format as an expansion board product ID. The ID of an embedded device can be accessed through I/O addresses 0zC80h-0zC83h, where "z" is the embedded slot number.

4.11 Expansion Board Control Bits

Port 0zC84h contains ENABLE, IOCHKERR, and IOCHKRST bits for software control of programmable expansion boards. EISA expansion boards must indicate "IOCHKERR=INVALID" in the CFG file if ENABLE and IOCHKERR bits are not supported. The Expansion Board Control Bits are shown in the following figure.

Expansion Board Control Bits - 0zC84h



Bit 0 - Enable Bit (Read/Write)

The ENABLE bit can be set to enable an expansion board for operation, or cleared to disable operation. The bit can be read to determine the enabled or disabled state. The expansion board clears ENABLE after sampling RESDRV asserted and enters a disabled state. The expansion board must only decode slot-specific I/O while in the disabled state. The expansion board must disable all bus drivers while in the disabled state, except when responding to slot-specific I/O. EISA expansion boards must fully support the ENABLE bit functions.

Bit 1 - IOCHKERR Bit (Read Only)

The IOCHKERR bit can be read to determine if an expansion board has a pending error. The expansion board indicates a pending error by setting IOCHKERR, clearing the ENABLE bit and entering the disabled state. The expansion board may, but is not required to assert the bus signal IOCHK* when it sets IOCHKERR. Pulsing IOCHKRST resets IOCHKERR. EISA expansion boards must respond to a read access of the IOCHKERR bit. EISA expansion boards that do not need to indicate errors may always respond with the IOCHKERR bit cleared.

An expansion board sets IOCHKERR to indicate that a serious error has occurred. Parity errors and uncorrectable system errors exemplify problems that might cause an expansion board to set IOCHKERR. An expansion board always holds IOCHKERR set while asserting the bus signal, IOCHK*. The main CPU or bus master can poll the IOCHKERR bit for each expansion board to determine which board caused an error.

Bit 2 - IOCHKRST Bit (Write Only)

Pulsing IOCHKRST to a "1" for at least 500 ns resets an expansion board's hardware. The expansion board resets all logic, assumes a disabled state, clears IOCHKERR and clears ENABLE when IOCHKRST is pulsed. EISA expansion boards that never set the IOCHKERR bit may ignore write accesses to the IOCHKRST bit.

Example Sequence for an IOCHKERR

The system ROM power-up routine initializes the expansion board and sets the ENABLE bit to begin operation.

The expansion board begins decoding memory and I/O addresses outside the slot-specific I/O range and enables its bus drivers to drive the bus signals.

The device driver determines the slot-specific I/O address from the configuration data in nonvolatile memory. The device driver can then control the device operation.

The expansion board detects a serious error, clears the ENABLE bit, sets its IOCHKERR bit and asserts IOCHK*. The expansion board stops decoding memory addresses and I/O addresses outside its slot specific range and it floats all bus drivers (except the one driving IOCHK*) unless responding to slot-specific I/O.

The expansion board detects a serious error, clears the ENABLE bit, sets its IOCHKERR bit and asserts IOCHK*. The expansion board disables all bus signal drivers except the one driving IOCHK*. The expansion board stops decoding memory addresses and I/O addresses outside its slot specific range.

The assertion of IOCHK* invokes the NMI service routine. The NMI service routine sequentially polls the IOCHKERR bit for each EISA device until it finds a device with IOCHKERR set. The NMI service routine then begins the recovery procedure (restore the operation or disable the expansion board).

To restore the expansion board, correct the error, then pulse IOCHKRST to "1" for at least 500 ns to clear the IOCHKERR bit and negate the IOCHK* bus signal. The NMI service routine can then invoke the device driver to initialize the expansion board and set the ENABLE bit for operation.

To disable the expansion board, the NMI service routine must pulse IOCHKRST to "1" for at least 500 ns to clear the IOCHKERR bit and negate the IOCHK* bus signal. The NMI service routine can also display a message to the user indicating the action taken.

The NMI service routine returns execution to the routine interrupted by NMI. If multiple devices asserted IOCHK*, or if another device asserted IOCHK* during the NMI service, the NMI routine is invoked again to repeat the IOCHKERR poll and recovery procedure.

4.12 System Software Use of Configuration Information

Device drivers and system software can use the configuration information from nonvolatile memory for the following purposes:

- Determine the slot number of an EISA device
- Determine the I/O address of the EISA device registers specified during configuration
- Determine configuration information
- Determine the system resources used by an EISA or ISA device
- Initialize the device for operation

Use of the configuration memory by a product dependent device driver may differ from use by a product independent device driver. A device driver is product dependent if the driver is provided for use with a particular product (i.e., an ACE Ethernet network board). A device driver is product independent if the driver is provided for use with products from a variety of vendors (such as a parallel port).

4.12.1 Slot Search by Product Independent Device Driver

A product independent device driver should check the TYPE string of each function in each slot (including expansion slots, embedded devices and virtual devices) to determine the slot in which the desired function is installed. The driver should begin searching at Slot 0, function 1 and sequentially increment through each function of each slot until the last slot has been checked.

The device driver can use the "Read Slot Configuration Information" to determine the number of functions located in any slot, and use the "Read Function Configuration Information" BIOS routine Call to read the configuration information (which includes the TYPE string) for the function. The device driver terminates the search when it finds a function with the desired TYPE string or when the "Read Slot Configuration Information" BIOS routine Call returns an "Invalid slot number" error. The error indicates that all slots have been checked.

Device Driver Search for TYPE String

The following example illustrates a device driver search for a parallel port with TYPE = "PAR."

The device driver performs the search by executing a "Read Slot Configuration Information" BIOS routine Call for each slot to determine if a device is installed and the number of functions present in the slot. The device driver begins the search by executing a "Read Slot Configuration Information" BIOS routine Call for slot 0 to determine the number of functions addressed as slot 0.

The device driver then executes a "Read Function Configuration Information" BIOS routine Call for slot 0 function 1. The BIOS routine reads the function configuration information from nonvolatile memory and writes it to a table in system memory. The device driver inspects the TYPE field in the returned table to determine if the first three characters of the TYPE string equal "PAR," which indicates a parallel port. The device driver continues executing "Read Function Configuration Information" BIOS routine Calls and inspecting the TYPE field for each slot 0 function.

The device driver then executes a "Read Slot Configuration Information" BIOS routine Call for slot 1 to determine the number of functions addressed as slot 1. The device driver requests the function information from nonvolatile memory and inspects the TYPE field for each function in slot 1. The device driver continues the slot search until it locates one or all functions with TYPE = "PAR", or until the "Read Function Configuration Information" BIOS routine Call indicates that all slots have been searched (by returning "invalid slot").

If the device driver finds a function with TYPE = "PAR", it can determine the initialization and resource requirements from the table returned by a "Read Function Configuration Information" BIOS routine Call.

Device Driver Search for SUBTYPE String

A driver can search for a specific configuration of a function by scanning the SUBTYPE strings. The following example illustrates a device driver search for a serial port with SUBTYPE = "COM1."

The device driver first finds an asynchronous communications port by searching for the TYPE string fragment, "COM,ASY." The driver then scans past the remainder of the TYPE field (delimited by the semicolon) and compares the SUBTYPE string fragments to "COM1." If a SUBTYPE string fragment does not match "COM1", the driver continues searching for another TYPE "COM,ASY" and checking the SUBTYPE for "COM1."

4.12.2 Slot Search by a Product Dependent Device Driver

A product dependent device driver should check the product ID of the device in each slot (including expansion slots, embedded devices and virtual devices) to determine the slot in which its corresponding product is installed. The driver should begin searching at Slot 0 and sequentially increment through each slot until the last slot has been checked.

The device driver can use the "Read Slot Configuration Information" BIOS routine Call to read the product ID of the device in any slot. The device driver terminates the search when it finds the correct product ID or when the BIOS routine Call returns an "Invalid slot number" error. The error indicates that all slots have been checked.

4.12.3 Device Driver Initialization for EISA Expansion Boards

The device driver can use information from nonvolatile memory to determine EISA expansion board configuration and initializations necessary to restore expansion board registers to their power-up condition.

The EISA system ROM initializes the following interrupt and DMA controller configurations after performing all I/O initializations indicated in nonvolatile memory. A device driver may not change the configurations:

Interrupt controller edge/level register
DMA controller (Extended Mode Register)
    DMA channel cycle timing
    DMA data size and addressing mode
DMA controller (DMA Command Register)
    Fixed or rotating priority scheme

A DMA device that shares the DMA channel may not change the following DMA controller configuration:

DMA controller (DMA Command Register)
    DRQ and DAK* assert level (high/low)

The device driver can use the "Read Function Configuration Information" BIOS routine Call to get the configuration parameters from nonvolatile memory. The configuration parameters returned from nonvolatile memory represent the expansion board configuration initialized by the system ROM power-up routines. Subsequent operation of the expansion board may leave the configuration in a different state. Device drivers can read the expansion board configuration registers to determine the configuration after power-up.

4.13 Creating TYPEs and SUBTYPEs for Devices

The TYPE and SUBTYPE identifiers are used by product independent device drivers to identify, initialize and operate an installed device that is compatible with the device driver. System board and expansion board manufacturers must specify consistent and expandable TYPE and SUBTYPE identifiers for their products.

The following guidelines should be followed when creating TYPE and SUBTYPE strings to assure consistency and expandability.

4.13.1 TYPE Strings

The first segment of the TYPE string should identify the most general device characteristics (such as video, communications port) followed by TYPE string segments that identify more detailed device characteristics (such as VGA video adapter, asynchronous communications port). For example, the TYPE string for a VGA video adapter is "VID,VGA", where "VID" identifies a video board and "VGA" indicates VGA compatibility. The TYPE string for the asynchronous communications port is "COM,ASY", where "COM" identifies a communications board and "ASY" indicates compatibility with the PC-AT asynchronous port.

New TYPE segments should be appended to the TYPE string when a device is enhanced with additional capabilities. A device driver compatible with the original product determines its ability to control the device after checking the original TYPE segments. A device driver that supports enhanced capabilities checks the appended TYPE segments to determine the level of capability supported by the device .

For example, the TYPE string for a VGA video adapter (ACE) with a 1024x768 high resolution mode might be: "VID,VGA,ACE1024X768". Device drivers that support VGA identify the video adapter as VGA compatible and device drivers that support 1024x768 identify the video adapter as compatible with the 1024x768 mode.

Another vendor (XYZ) may offer a compatible video adapter with a new 1280x1024 mode. The TYPE string for the 1280x1024 video adapter might be: "VID,VGA,ACE1024X768,XYZ1280X1024". Device drivers that support VGA identify the video adapter as VGA compatible, device drivers that support 1024x768 identify the video adapter as compatible with the 1024x768 mode, and device drivers that support 1280X1024 identify the video adapter as compatible with the 1280X1024 mode.

4.13.2 SUBTYPE Strings

The SUBTYPE string identifies the device options selected during configuration. A device driver can scan the TYPE string to determine that the device is compatible with the driver, then scan the SUBTYPE string to determine the device configuration. For example, the video adapter described above might use the SUBTYPE field to indicate the power-up video display mode.

```
FUNCTION "VGA Video Adapter"
        TYPE = "VID,VGA,ACE1024x768,XYZ1280x1024"
        CHOICE(1) = "VGA Default Mode"
            SUBTYPE = "DMODE=VGA"
        CHOICE(2) = "1024X768" Default Mode
            SUBTYPE = "DMODE=ACE1024X768"
        CHOICE(3) = "1280X1024" Default Mode
            SUBTYPE = "DMODE=XYZ1280X1024"
```

The device driver can utilize the SUBTYPE string to determine the default mode set during power-up. The TYPE/SUBTYPE string for a selection of VGA as the default power-up video mode is:

```
"VID,VGA,ACE1024x768,XYZ1280x1024;DMODE=VGA"
```

A device driver should read the device configuration registers for configuration information that changes during device operation. A driver that needs detailed configuration information not specified in the SUBTYPE string should also read the device configuration registers.

4.13.3 Standard TYPE Table

The following TYPEs should be used wherever possible for the applicable devices. System and expansion board manufacturers can create additional TYPEs for devices that do not apply to the standard TYPEs listed here. For example, a manufacturer of a fax board can create a new TYPE = "FAX" or can use the "COM" prefix (i.e., "COM,FAX"). The new TYPEs become a de facto standard if other vendors use the same TYPE.

The standard device TYPEs for commonly used devices that are part of the industry standard system architecture are listed below.

DEVICE TYPE	DEVICE DESCRIPTION
"COM,ASY" "COM,ASY,FIFO" "COM,SYN" "KEY,nnn,KBD=xx"	ISA compatible 8250-based serial port ISA compatible 16550-based serial port (with FIFO) ISA compatible SDLC port Standard keyboards XX=country, nnn = number of keys. 083 084 101 103 xx = Keyboard Code AE = Arabic - English AF = Arabic - French AU = Australia BE = Belgium BF = Belgium - Flemish CE = Canadian - English CF = Canadian - French CH = China DN = Denmark DU = Dutch EE = European - English FN = Finland FR = France GR = Germany HA = Hungary IT = Italy IS = Israel KA = Kangi LA = Latin America ME = Middle East NE = Netherlands NO = Norway PO = Portugal SP = Spain SW = Sweden ST = Switzerland SF = Swiss - French SG = Swiss - German TA = Taiwan UK = United Kingdom US = United States

DEVICE TYPE	DEVICE DESCRIPTION
"CPU,8086" "CPU,80286" "CPU,80386SX" "CPU,80386" "CPU,80486"	8086 compatible microprocessor 80286 compatible microprocessor 80386SX compatible microprocessor 80386 compatible microprocessor 80486 compatible microprocessor
"MSD,DSKCTL" "MSD,FPYCTL" "MSD,TAPCTL"	ISA compatible fixed disk controller ISA compatible floppy disk controller Primary tape controller
"NPX,287" "NPX,387" "NPX,387SX" "NPX,W1167" "NPX,W3167"	Intel 287 numeric coprocessor Intel 387 numeric coprocessor Intel 387SX numeric coprocessor for 386SX Weitek 1167 numeric coprocessor Weitek 3167 numeric coprocessor
"JOY"	ISA compatible joystick adapter
"PAR" "PAR,BID"	ISA compatible parallel port Bidirectional parallel port
"PTR,8042"	8042 pointing device (mouse) interface
"VID,MDA" "VID,MDA,MGA" "VID,CGA" "VID,CGA,RTR" "VID,CGA" "VID,EGA" "VID,VGA"	ISA compatible monochrome adapter Hercules monochrome adapter Requires no write sync during retrace Requires write sync during retrace ISA compatible CGA adapter ISA compatible EGA adapter ISA compatible VGA adapter

4.14 Configuration Example

This section contains the configuration data structures associated with an example EISA Ethernet communication board. The example illustrates the configuration information for initialization ports, a DMA channel, an interrupt, RAM memory and ROM memory.

The example includes the configuration file, the configuration data structure returned by a "Read Function Configuration Information" BIOS routine Call, and the configuration data structure passed to the "Write Nonvolatile Memory" BIOS routine.

4.14.1 Configuration File

An example of a configuration file for an ethernet controller board is presented on the following pages. The CFG filename for this file is !ACE105.CFG

```
BOARD
    ID = "ACE0105"
    NAME = "ACME Ethernet Interface board - Revision 5"
    MFR = "ACME Board Manufact."
    CATEGORY = "NET"
    SLOT = EISA
    LENGTH = 330
    READID = yes

IOPORT(1) = 0zc94h
    INITVAL = 0000xxxx

IOPORT(2) = 0zc98h
    INITVAL = xxxxxxxxxxxxxxrr

IOPORT(3) = 0zc9ah
    INITVAL = xxxxxxrr

IOPORT(4) = 0zc9bh
    INITVAL = rrrrrxxx

IOPORT(5) = 0ZC85h
    INITVAL = xxxxxxxx

IOPORT(6) = 0ZC86h
    INITVAL = 0rrxxxxx

IOPORT(7) = 0ZC86h
    INITVAL = 1rrxxxxx

SOFTWARE(1) = "ACELINK.EXE - \n if using MS DOS
    Place the following command line in AUTOEXEC.BAT: \n
    \t\tACELINK /S = n /A = n\n
    Use the following values with the
    /S and /A parameters:"
```

```
; Function description starts here

GROUP = "Ethernet network interface"
    TYPE = "NET,ETH"
FUNCTION = "Network Interface Location"
    CHOICE = "File Server Init. - Node 0"
        SUBTYPE = "LAN0"
        FREE
            INIT = SOFTWARE(1) "/S=1 /A=0"
            INIT = IOPORT(5) LOC (5-2) 0000
    CHOICE = "Network user init. - Node 1"
        SUBTYPE = "LAN1"
        FREE
            INIT = SOFTWARE(1) "/S=0 /A=1"
            INIT = IOPORT(5) LOC (5-2) 0001
    CHOICE = "Network user init. - Node 2"
        SUBTYPE = "LAN2"
        FREE
            INIT = SOFTWARE(1) "/S=0 /A=2"
            INIT = IOPORT(5) LOC (5-2) 0010

; Additional detail may be added

    CHOICE = "Network user init. - Node 15"
        SUBTYPE = "LAN15"
        FREE
            INIT = SOFTWARE(1) "/S=0 /A=15"
            INIT = IOPORT(5) LOC (5-2) 1111

FUNCTION = "System resources alloc./init."
    CHOICE = "System Resources"
; DMA channel operates in Type C (burst) timing
        LINK
            DMA = 5 | 7
                SHARE = no
                SIZE = dword
                TIMING = TYPEC
            INIT = IOPORT(5) LOC (0) 0 | 1
; Interrupt is level-sensitive
        LINK
            IRQ = 2 | 5
                SHARE = yes
                TRIGGER = level
            INIT = IOPORT(5) LOC (1) 0 | 1
```

```
; Network board local ROM
        COMBINE
            MEMORY = 2K
                ADDRESS =  0C0000H | 0D0000h | 0E0000h
                MEMTYPE = oth
                WRITABLE = no
                SHARE = no
                SIZE = byte
                CACHE = yes
                DECODE = 32
            INIT = IOPORT(6) LOC (3-0) 1100 | 1101 | 1110

; Network board local Ram
FUNCTION = "Local RAM Initialization"
    CHOICE = "64K RAM"
        SUBTYPE = "64K"
        COMBINE
            MEMORY = 64K
                ADDRESS = 100000H-1F0000H  STEP = 64K
                WRITABLE = yes
                MEMTYPE = oth
                SIZE = dword
                CACHE = no
            INIT = IOPORT(7) LOC(4 3 2 1 0) 00000-01111
    CHOICE = "128K RAM"
        SUBTYPE = "128K"
        COMBINE
            MEMORY = 128K
                ADDRESS = 100000H-1F0000H  STEP = 64K
                MEMTYPE = oth
                WRITABLE = yes
                SIZE = dword
                CACHE = no
            INIT = IOPORT(7) LOC(4 3 2 1 0) 10000-11111
 ENDGROUP
```

```
; Serial Port section
FUNCTION = "Serial Port"
    TYPE = "COM,ASY"
    CHOICE = "COM1"
        SUBTYPE = "COM1"
        FREE
            IRQ = 4
                SHARE = yes
                TRIGGER = level
            PORT = 3f8h-3ffh
                SHARE = no
                SIZE = byte
            INIT = IOPORT(1) LOC (3-0) 0000
            INIT = IOPORT(2) LOC (15-2) 00000011111100
            INIT = IOPORT(3) LOC (7-2) 110000
            INIT = IOPORT(4) LOC (2-0) 010
    CHOICE = "COM2"
        SUBTYPE = "COM2"
        FREE
            IRQ = 3
                SHARE = yes
                TRIGGER = level
            PORT = 2F8h-2ffh
                SHARE = no
                SIZE = byte
            INIT = IOPORT(1) LOC (3-0) 0000
            INIT = IOPORT(2) LOC (15-2) 00000011111100
            INIT = IOPORT(3) LOC (7-2) 110000
            INIT = IOPORT(4) LOC (2-0) 000

    CHOICE = "Port disable"
        SUBTYPE = "Port disable"
        DISABLE = yes
        FREE
            INIT = IOPORT(4) LOC(0) 0
```

4.14.2 Read Slot Configuration Information BIOS Routine

The following example illustrates a "Read Slot Configuration Information" BIOS routine Call. The data block returned by the BIOS routine indicates an ACME Ethernet Board installed in slot 4.

Assume the following register assignments prior to executing the "Read Slot Configuration Information" BIOS routine Call:

INT 15h, AH = D8h, AL = 00h

INPUT:

```
AH =  0D8h
AL =  0      ;Read Slot Configuration Information
CL =  4      ;Slot number for ACME Ethernet Board
```

The following register values illustrate the parameters returned by the "Read Slot Configuration Information" INT15 Call:

OUTPUT:

```
AH       =  00h--Successful Completion (carry flag = 0)
AL       =  00h--No duplicate IDs and board ID is readable
BH       =  01h--Major Revision Level of Configuration Utility
BL       =  01h--Minor Revision Level of Configuration Utility
CH       =  ADh--Checksum of Configuration File (MSByte)
CL       =  09h--Checksum of Configuration File (LSByte)
DH       =  04h--Number of Functions on this board
DL       =  00111111b--Combined Function information byte
DI and SI =  Four byte compressed ID
             DI(lsb) = 04h (byte 0)
             DI(msb) = 65h (byte 1)
             SI(lsb) = 01h (byte 2)
             SI(msb) = 05h (byte 3)
```

4.14.3 Read Function Configuration Information BIOS Routine Call

The following examples illustrate the "Read Function Configuration Information" BIOS routine call. The data block returned by the BIOS routine indicates an ACME Ethernet Board installed in slot 4.

Assume the following register assignments prior to executing the "Read Function Configuration Information" INT15 call:

INT 15h, AH = D8h, AL = 01h

INPUT:

```
AH =  0D8h
AL =  01h          ;Read Function Configuration Information
CL =  04h          ;Slot number for ACME Ethernet Board
CH =  00h          ;Read the data block for function 0
DS:SI = 29B9:0600  ;pointer to the data block returned
```

The following register values illustrate the parameters returned by the "Read Function Configuration Information" BIOS routine call:

OUTPUT:

AH = 00h Successful completion (carry flag = 0)

The table on the following page illustrates the data block returned by the "Read Function Configuration Information" BIOS routine call for function 0.

Off-set	Byte #	Value	Description
00h	1	04h	1st Byte Expansion Board ID: ACE0105 (0465h)
	2	65h	2nd Byte Expansion Board ID
	3	01h	first and second hex digit of product number
	4	05h	third digit of product number/1-digit revision number
04h	5	00h	ID and slot information
	6	03h	Miscellaneous ID Information
06h	7	01h	Major Configuration Utility Revision Level
	8	01h	Minor Configuration Utility Revision Level
08h	9	00h	1st Selection
	10	00h	2nd Selection
	11	0	Not Used
	:	0	"
22h	35	21h	Function information (00001111b)
23h	36	N	TYPE string starts here
	37	E	
	38	T	
	39	,	Delimiter that separates TYPE string fragments
	40	E	
	41	T	
	42	H	End of TYPE string
	43	;	Delimiter to append subtype string
	44	L	
	45	A	
	46	N	
	47	0	End of SUBTYPE string
	48	0	Not Used
	:	0	"
	104h	261	80h Initialization Byte IOPORT(1)
	262	94h	LSB IOPORT ADDRESS
	263	4Ch	MSB IOPORT ADDRESS
	264	00h	PORT VALUE
	265	85h	Initialization Byte IOPORT(2)
	266	98h	LSB IOPORT ADDRESS
	267	4Ch	MSB IOPORT ADDRESS
	268	F0h	LST PORT VALUE
	269	03h	MSB PORT VALUE
	270	03h	LSB PORT MASK
	271	00h	MSB PORT MASK
	272	84h	Initialization Byte IOPORT(3)
	273	9Ah	LSB IOPORT ADDRESS
	274	4Ch	MSB IOPORT ADDRESS
	275	C0h	PORT VALUE
	276	03h	PORT MASK

Off-set	Byte #	Value	Description
	277	84h	Initialization Byte IOPORT(4)
	278	9Bh	LSB IOPORT ADDRESS
	279	4Ch	MSB IOPORT ADDRESS
	280	00h	PORT VALUE
	281	F8h	PORT MASK
	282	80h	Initialization Byte IOPORT(5)
	283	85h	LSB IOPORT ADDRESS
	284	4Ch	MSB IOPORT ADDRESS
	285	00h	PORT VALUE
	286	84h	Initialization Byte IOPORT(6)
	287	86h	LSB IOPORT ADDRESS
	288	4Ch	MSB IOPORT ADDRESS
	289	0Ch	PORT VALUE
	290	60h	PORT MASK
	291	04h	Initialization Byte IOPORT(7)
	292	86h	LSB IOPORT ADDRESS
	293	4Ch	MSB IOPORT ADDRESS
	294	80h	PORT VALUE
	295	60h	PORT MASK
127h	296	00h	Not Used
	.	00h	"
13Fh	.	00h	"

The following table illustrates the data block returned by the "Read Function Configuration Information" BIOS routine call for function 1. The register setup is the same as for the last call except CH=01h.

Off-set	Byte #	Value	Description
00h	1	04h	1st Byte Expansion Board ID: ACE0105 (0465h)
	2	65h	2nd Byte Expansion Board ID
	3	01h	first and second digit of product number
	4	05h	third digit of product number/1-digit revision number
04h	5	00h	ID and slot information
	6	03h	Miscellaneous ID Information
06h	7	01h	Major Configuration Utility Revision Level
	8	01h	Minor Configuration Utility Revision Level
08h	9	00h	1st Selection
	10	00h	2nd Selection
	11	00h	3rd Selection
	12	00h	4th Selection
	13	00h	5th Selection
	14	00h	Not Used
	.	00h	"
	.	00h	"

Off-set	Byte #	Value	Description
22h	35	0Fh	Function information (00001111b)
23h	36	N	TYPE string starts here
	37	E	
	38	T	
	39	,	Delimiter that separates TYPE string fragments
	40	E	
	41	T	
	42	H	End of TYPE string
	43	00h	Not Used
	.	00h	"
	.	00h	"
73h	116	18h	Memory Configuration: ROM - (00011000b)
	117	08h	ROM memory size (byte)
	118	00h	LSByte ROM Start Address (0D0000h/100h = 0D00h)
	119	0Ch	Middle Byte ROM Start Address
	120	00h	MSByte of ROM Start Address
	121	02h	LSByte ROM size (2048/400h = 0002h)
	122	00h	MSByte ROM size
	123	00h	Not Used
	.	00h	"
	.	00h	"
B2h	179	22h	Interrupt configuration: IRQ2 (00100010b)
	180	00h	Reserved
	181	00h	Not Used
	.	00h	"
	.	00h	"
C0h	193	05h	DMA configuration: DMA channel 5 (00000101b)
	194	38h	32-bit BURST transfers (00111000b)
	195	00h	Not Used
	.	00h	"
	.	00h	"
C8h	201	00h	Not Used
	.	00h	"
	.	00h	"

The following table illustrates the data block returned by the "Read Function Configuration Information" BIOS routine call for function 2. The register setup is the same as for the last call except CH = 02h.

Off-set	Byte #	Value	Description
00h	1	04h	1st Byte Expansion Board ID: ACE0105 (0465h)
	2	65h	2nd Byte Expansion Board ID
	3	01h	first and second hex digit of product number
	4	05h	third digit of product number/1-digit revision number
04h	5	00h	ID and slot information
	6	03h	Miscellaneous ID Information
06h	7	01h	Major Configuration Utility Revision Level
	8	01h	Minor Configuration Utility Revision Level
08h	9	00h	1st Selection
	10	00h	2nd Selection
	11	00h	3rd Selection
	12	00h	Not Used
	.	00h	"
	.	00h	"
22h	35	03h	Function information (00000111b)
23h	36	N	TYPE string starts here
	37	E	
	38	T	
	39	,	Delimiter that separates TYPE string fragments
	40	E	
	41	T	
	42	H	
	43	;	Delimiter to append subtype string
	44	6	
	45	4	
	46	K	End of SUBTYPE string
	47	00h	Not Used
	.	00h	"
	.	00h	"
73h	116	19h	Memory Configuration: RAM - (00011001b)
	117	02h	RAM Memory Data Size (Dword)
	118	00h	LSByte ROM Start Address (0D0000h/100h = 0D00h)
	119	10h	Middle Byte ROM Start Address
	120	00h	MSByte of ROM Start Address
	121	40h	LSByte ROM size (2048/400h = 0002h)
	122	00h	MSByte ROM size
	123	00h	Not Used
	.	00h	"
	.	00h	"

The following table illustrates the data block returned by the "Read Function Configuration Information" BIOS routine call for function 3. The register setup is the same as for the last call except CH=03h.

Off-set	Byte #	Value	Description
00h	1	04h	1st Byte Expansion Board ID: ACE0102 (0465h)
	2	65h	2nd Byte Expansion Board ID
	3	01h	first and second digit of product number
	4	05h	third digit of product number/1-digit revision number
04h	5	00h	ID and slot information
	6	03h	Miscellaneous ID Information
06h	7	01h	Major Configuration Utility Revision Level
	8	01h	Minor Configuration Utility Revision Level
08h	9	01h	1st Selection
	11	00h	2nd Selection
	12	00h	3rd Selection
	13	00h	Not Used
	:	00h	"
22h	35	15h	Function information (00011001b)
23h	36	C	TYPE string starts here
	37	O	
	38	M	
	39	,	Delimiter that separates TYPE string fragments
	40	A	
	41	S	
	42	Y	End of SUBTYPE string
	43	;	Delimiter to append subtype string
	44	C	
	45	O	
	46	M	
	47	2	End of SUBTYPE string
	48	00h	Not Used
	:	00h	"
B2h	179	23h	Interrupt configuration: IRQ3 (00100011b)
	180	00H	Reserved
	181	00h	Not Used
	:	00h	"
C8h	201	07h	Port IO Range entry (00000011b)
	202	F8h	LSB Port Address
	203	02h	MSB Port Address
	204	00h	Not Used
	:	00h	"

4.14.4 Write Nonvolatile Memory BIOS Routine CALL

The following example illustrates a Write Nonvolatile Memory BIOS routine call.

INT 15h, AH = D8h, AL = 03h

INPUT:

AH = 0D8h
AL = 03h
CX = 0041h
DS:SI = 15AA:0244

OUTPUT:

AH = 00h Successful completion (carry flag = 0)

The data structure that is passed to the Write Nonvolatile Memory BIOS routine for the ACME Ethernet board example:

Off-set	Byte #	Value	Description
00h	1	04h	1st Byte Expansion Board ID: ACE0105 (0465h)
	2	65h	2nd Byte Expansion Board ID
	3	01h	first and second digit of product number
	4	05h	third digit of product number/1-digit revision number
04h	5	00h	ID and slot information (00000000b)
	6	03h	Reserved
06h	7	01h	Major Configuration Utility Revision Level
	8	01h	Minor Configuration Utility Revision Level 00h if no CFG File Extensions
08h	9	34h	LSB length of function 0 entry
	10	00h	MSB length of function 0 entry
0Ah	11	02h	Length of following selections field
	12	00h	1st selection
	13	00h	2nd selection
0Dh	14	21h	Function 0 information byte (00100001b)
0Eh	15	0Ch	Length of following ASCII TYPE string
0Fh	16	4Eh	N TYPE string starts here
	17	45h	E
	18	54h	T
	19	2Ch	, Delimiter- separates TYPE string fragments
	20	45h	E
	21	54h	T
	22	48H	H End of TYPE string
	23	3Bh	; Delimiter to append SUBTYPE string
	24	4Ch	L SUBTYPE string starts here
	25	41h	A
	26	4Eh	N
	27	30h	0

Off-set	Byte #	Value	Description
1Bh	28	80h	Initialization Byte IOPORT(1)
	29	94h	LSB IOPORT ADDRESS
	30	4Ch	MSB IOPORT ADDRESS
	31	00h	PORT VALUE
	32	85h	Initialization Byte IOPORT(2)
	33	98h	LSB IOPORT ADDRESS
	34	4Ch	MSB IOPORT ADDRESS
	35	F0h	LST PORT VALUE
	36	03h	MSB PORT VALUE
	37	03h	LSB PORT MASK
	38	00h	MSB PORT MASK
	39	84h	Initialization Byte IOPORT(3)
	40	9Ah	LSB IOPORT ADDRESS
	41	4Ch	MSB IOPORT ADDRESS
	42	C0h	PORT VALUE
	43	03h	PORT MASK
	44	84h	Initialization Byte IOPORT(4)
	45	9Bh	LSB IOPORT ADDRESS
	46	4Ch	MSB IOPORT ADDRESS
	47	00h	PORT VALUE
	48	F8h	PORT MASK
	49	80h	Initialization Byte IOPORT(5)
	50	85h	LSB IOPORT ADDRESS
	51	4Ch	MSB IOPORT ADDRESS
	52	00h	PORT VALUE
	53	84h	Initialization Byte IOPORT(6)
	54	86h	LSB IOPORT ADDRESS
	55	4Ch	MSB IOPORT ADDRESS
	56	0Ch	PORT VALUE
	57	60h	PORT MASK
	58	04h	Initialization Byte IOPORT(7)
	59	86h	LSB IOPORT ADDRESS
	60	4Ch	MSB IOPORT ADDRESS
	61	80h	PORT VALUE
	62	60h	PORT MASK
3Eh	63	1Ah	LSB length of function 1 entry
	64	00h	MSB length of function 1 entry
	65	05h	Length of following selections field
	66	00h	1st Selection
	67	00h	2nd Selection
	68	00h	3rd Selection
	69	00h	4th Selection
	70	00h	5th Selection

Off-set	Byte #	Value	Description
46h	71	0Fh	Function 1 Information Byte (00001111h)
47h	72	07h	Length of following ASCII string field
48h	73	4Eh	N Type String Starts Here
	74	45h	E
	75	54h	T
	76	2Ch	, Delimiter that separates TYPE string
	77	45h	E
	78	54h	T
	79	48h	H
4Fh	80	18h	Memory Config Byte (00011010b OTH cacheable ROM) Although this memory is cacheable, caching isn't implemented in this configuration and is so represented.
	81	08h	Memory Data Size - Byte
	82	00h	LSB Mem Start Address (divided by 100h)
	83	0Ch	Middle Mem Start Address
	84	00h	MSB Memory Start Address
	85	02h	LSB Memory Size (bytes divided by 400h)
	86	00h	MSB Memory Size (0002*400 = 800h = 2k)
56h	87	22h	Interrupt Configutation Byte Although this interupt may be shared, it doesn't need to be in this configuration and is so represented.
	88	00h	Reserved
58h	89	05h	DMA Configutation: DMA Channel 5 (00000101b)
	90	38h	32-bit BURST Transfers (00111000b)
5Ah	91	18h	LSB length of function 2 entry
	92	00h	MSB length of function 2 entry
	93	03h	Length of following Selections field
	94	00h	1st Selection
	95	00h	2nd Selection
	96	00h	3rd Selection
60h	97	03h	Function 2 Info. Byte
61h	98	0Bh	Length of follow string field
62h	99	4Eh	N
	100	45h	E
	101	54h	T
	102	2Ch	, Delimiter that separates TYPE string
	103	45h	E
	104	54h	T
	105	48h	H
	106	3Bh	; Delimiter to append SUBTYPE string
	107	36h	6
	108	34h	4
	109	4Bh	K

Off-set	Byte #	Value	Description
6Dh	110	19h	Memory Configuration Byte (00011001b)
	111	02h	Memory Data Size (Dword)
	112	00h	LSB Memory Start Address (divided by 100h)
	113	10h	Middle Mem Start Address
	114	00h	MSB Memory Start Address
	115	40h	LSB Memory Size (bytes divided by 400h)
	116	00h	MSB Memory Size (0040*400 =10000h = 16k)
74h	117	17h	LSB length of function 3 entry
	118	00h	MSB length of function 3 entry
76h	119	03h	Length of following selections field
	120	01h	1st Selection
	121	00h	2nd Selection
	122	00h	3rd Selection
7Ah	123	15h	Function 3 Information Byte
7Bh	124	0Ch	Length of following ASCII string field
7Ch	125	43h	C
	126	4Fh	O
	127	4Dh	M
	128	2Ch	, Delimiter that separates TYPE string
	129	41h	A
	130	53h	S
	131	59h	Y
	132	3Bh	; Delimiter to append SUBTYPE string
	133	43h	C
	134	4Fh	O
	135	4Dh	M
	136	32h	2
88h	137	23h	Interrupt Configuration Byte Although this interupt may be shared, it doesn't need to be in this configuration and is so represented.
	138	00h	Reserved
8Ah	139	07h	Port IO Range entry (00000011b)
	140	F8h	LSB Port Address
	141	02h	MSB Port Address
8Dh	142	00h	LSB Last Function Length = 0
	143	00h	MSB Last Function Length = 0
8Fh	144	09h	LSB Configuration file Checksum
	145	ADh	MSB Configuration file Checksum

4.14.5 Read Physical Slot BIOS Routine Call

This BIOS routine obtains the ID of the EISA board in the selected slot. The routine writes 0FFh to the selected slot and reads data back from the same slot. If the returned data equals 0FFh, no board ID can be obtained. If any value other than 0FFh is returned and the MSB equals zero, then a valid device ID has been obtained. Slot addresses are 0zC80h-0zC83h.

INT 15h, AH=0D8h, AL=04h

INPUT
AH = 0D8h
AL = 04h if called from 16-bit segment
AL = 84h if called from a 32-bit segment
CL = Slot number (including embedded and virtual slots)
0 = system board
1 = slot 1
2 = slot 2
n = slot n

OUTPUT
AH = 00h Successful completion; Carry flag = 0
80h Invalid slot number; Carry flag = 1
83h Empty slot; Carry flag = 1
86h Invalid BIOS routine call; Carry flag = 1
DI and SI
Four-byte compressed ID
DI (lsb) = Byte 0
DI (msb) = Byte 1
SI (lsb) = Byte 2
SI (msb) = Byte 3

The following procedure describes the steps necessary for obtaining the EISA device ID.

1. Determine the type of processor present on the system board. I/O Port 61h will be accessed to check the status of refresh cycles. I/O Port 61h is not available for 8086/8088-based systems. The processor type can only be determined in real mode.

2. Wait for the start and completion of the next refresh cycle. The occurrence of a refresh cycle during slot access may produce erroneous data. During refresh cycles, bit 4 of I/O Port 61h toggles from 1 to 0.

3. Load the AL register with FFh. Then clear the processor prefetch by executing a "jmp $+2". Write the contents to AL to the slot address 0zC80h.

4. Read the contents of slot address 0zC80h into the AL register and compare those results to 0FFh. If the data equals FFh, no device ID is available. If the contents are not equal to FFh and the MSB=0, then an EISA device ID may be read.

THIS PAGE INTENTIONALLY LEFT BLANK