# EISA Configuration File Format

Configuration files contain definitions of hardware configurations, including switch and jumper settings, software initializations, and so on. The EISA Configuration Utility (ECU) uses this information to allocate resources and create a conflict-free system. The resulting configuration is written to nonvolatile memory and saved on diskette.

When you write a new EISA/ISA bus device driver, you must also write a configuration file for the new bus device, include it on the ECU diskette, and ship it to customers for installation and configuration.

The configuration file name should begin with an exclamation point, followed by the product ID and the .CFG file extension. The file itself contains several statement blocks that describe the expansion board or system board, as follows:

* Board identification block - This required section of the configuration file identifies the board, its name, manufacturer, and category.
* Initialization information block - Each I/O port, set of switches, and set of jumpers requires its own initialization information block that specifies how data is read or written, how switches are set initially, or how jumpers are connected. Software initialization provides instructions about software drivers during system configuration.
* Function statement block - Configuration options are defined in function statement blocks. Within a function statement block there can be choices and subchoices, each with its own configuration and initialization options. You can define several configuration options in this way, and the ECU will choose the optimal configuration for any system, one that has no conflicts in the assignment of system resources such as interrupt levels and direct memory access (DMA) channels.

System boards require an additional statement block to describe the amount of nonvolatile memory available, the number of expansion slots on the system board, and other characteristics of the system board.

This chapter describes each section of the configuration file. It also discusses the additional statement blocks required for system boards.

## B.1    Expansion Board Identification Block

Each configuration file must begin with a board identification block. Four required fields provide the basic identification of the board. Optional fields (in brackets) provide additional identification information.

### Syntax

**BOARD   
ID = identifier   
NAME = board\_name   
MFR = manufacturer\_name   
CATEGORY = category   
[ SLOT = slot\_type [ ,type ] ]   
[ LENGTH = length ]   
[ AMPERAGE = amps ]   
[ SKIRT = { YES | NO } ]   
[ READID = { YES | NO } ]   
[ BUSMASTER = latency ]   
[ SIZING = gap ]   
[ IOCHECK = { VALID | INVALID } ]   
[ DISABLE = { SUPPORTED | UNSUPPORTED } ]   
[ VERSION = version\_number ]   
[ REV = revision ]   
[ COMMENTS = comment\_string ]   
[ HELP = help\_string ]**

identifier

Specifies a 7-character identifier for an expansion board. The identifier consists of a 3-character manufacturer code, a 3-character hexadecimal product identifier, and a 1-character revision number.

board\_name

Specifies up to 90 characters of text that identify the product, including information such as part numbers.

manufacturer\_name

Specifies up to 30 characters of text that identify the board manufacturer.

category

Specifies a 3-character identifier that represents the board category. The following identifiers are available:

|  |  |
| --- | --- |
| **Identifier** | **Description** |
| COM | Communications device |
| KEY | Keyboard |
| MEM | Memory board |
| MFC | Multifunction board |
| MSD | Mass storage device |
| NET | Netword board |
| NPX | Numeric coprocessor |
| OSE | Operating system/environment |
| OTH | Other |
| PAR | Parallel port |
| PTR | Pointing device |
| SYS | System board |
| VID | Video board |

slot\_type [,type] ...

Specifies one of the following identifiers, which represent the type of slot in which the user can install the expansion board:

|  |  |
| --- | --- |
| **Identifier** | **Description** |
| ISA8 | 8-bit ISA expansion board |
| ISA16 | 16-bit ISA expansion board |
| ISA8OR16 | 8-bit or 16-bit ISA expansion board (the default) |
| EISA | EISA expansion board |
| EMB[( n)] | Embedded device, using the specified I/O range |
| VIR | Virtual device |
| OTHER | Vendor-specific expansion slot |

You can include an optional text string (type) enclosed in quotation marks to describe the slot.

length

Specifies the length of the board in millimeters. You should include a LENGTH statement so that the ECU can optimize slot allocation. Otherwise, the default value of 330 is used. The LENGTH statement does not apply to embedded or virtual devices.

amps

Specifies the maximum amount of continuous 5-volt current (in milliamps) required by the base configuration of the expansion board. If your device requires more than 5 volts of power, you should include an AMPERAGE statement. Otherwise, the default value of 0 is used.

{**YES**|**NO**}

With the SKIRT statement, indicates whether a drop-down skirt is present to prevent installation into a 16-bit slot. The default value is NO.

With the READID statement, indicates whether the expansion board's ID can be read from the EISA ID registers. The default value is NO.

latency

Indicates that the board is a bus master. The latency argument specifies the maximum acceptable latency, in microseconds, from the bus master bus request to the bus grant.

gap

Specifies the minimum number of Kbytes of memory to be used as the sizing gap. The default is 0.

{**VALID**|**INVALID**}

Specifies support of the EISA expansion board control register IOCHKERR bit. VALID (the default) indicates that the expansion board responds to reads of its IOCHKERR bit; INVALID indicates that the expansion board does not respond to this bit.

{**SUPPORTED**|**UNSUPPORTED**}

Specifies support of the EISA expansion board control register ENABLE bit. SUPPORTED (the default) indicates that the user can disable the expansion board by clearing the ENABLE bit; UNSUPPORTED indicates that the user cannot disable the expansion board by clearing this bit.

version\_number

Specifies the version number.

revision

Specifies the revision number.

comment\_string

Specifies up to 600 characters of text that the ECU can display when the user requests help.

help\_string

Specifies up to 600 characters of text to display if the user requests help.

## B.2    Initialization Information Block

The initialization information block defines how I/O ports, switches, jumpers, and software are initialized. Each is defined in its own initialization block.

### B.2.1    I/O Port Initialization Statement Block

An I/O port initialization block initializes an I/O port to a specific address or to the address of a PORTVAR variable. The PORTVAR variable initializes the I/O port address based on a configuration selection.

The following CHOICE statements assign addresses to PORTVAR(3). Initialization of IOPORT(1) is based on the selected CHOICE statement. If COM1 is chosen, the ECU writes a value of 00000001b to port address 3F9h. If COM2 is chosen, it writes the value to port address 2F9h.

IOPORT(1) = PORTVAR(3)

FUNCTION = "Serial Port"

CHOICE = "COM1"

PORTVAR(3) = 3F9h

INIT = IOPORT(1) 00000001b

CHOICE = "COM2"

PORTVAR(2) = 2F9h

INIT = IOPORT(1) 00000001b

### Syntax

**IOPORT( i ) = PORTVAR( j )   
   
IOPORT( i ) = address   
[ SIZE = { BYTE | WORD | DWORD } ]   
[ INITVAL [ LOC ( bitlist ) ] valuelist ]   
  
.  
.  
.**

i

Uniquely identifies the I/O port as a number between 1 and 32767.

j

Identifies the address of an I/O port assigned within a choice or subchoice statement block. This value is assigned to i when the ECU selects the CHOICE or SUBCHOICE statement.

address

Specifies the address of the I/O port.

{**BYTE**|**WORD**|**DWORD**}

Specifies the number of bits in the I/O port, as follows:

|  |  |
| --- | --- |
| BYTE | 8-bit (the default) |
| WORD | 16-bit |
| DWORD | 32-bit |

bitlist

Contains a list or range of bits to be referenced, from most significant bit (left-most) to least significant bit (right-most).

valuelist

For each bit in bitlist, a corresponding bit in valuelist states how the bit value is read or written, using one of the following bitlist flags:

* r indicates that the bit value must be read from the port.
* x indicates that the ECU determines the bit value based on the selected configuration.
* l or 0 indicates that the bit is reserved and must be initialized to the specified value.

### B.2.2    Switch Configuration Block

The switch configuration block defines the types, layout, and initial positions of switches on the expansion board. The configuration file should contain one switch configuration block for each set of switches.

### Syntax

**SWITCH( i ) = value   
NAME = name   
STYPE = { DIP | ROTARY | SLIDE }   
[ VERTICAL = { YES | NO } ]   
[ REVERSE = { YES | NO } ]   
[ LABEL = LOC( switchlist ) labelist ]   
[ INITVAL = LOC( switchlist ) valuelist ]   
[ FACTORY = LOC( switchlist ) valuelist ]   
[ COMMENTS = comment\_string ]   
[ HELP = help\_string ]   
  
.  
.  
.**

i

Uniquely identifies the switch being configured as a number between 1 and 32767.

value

Specifies the number of switches in the set, up to the maximum of 16.

name

Specifies a switch name up to 20 characters long.

{**DIP**|**ROTARY**|**SLIDE**}

Specifies the switch type, as follows:

* DIP - A set of switches, each having two positions: ON and OFF
* ROTARY - A set of switches having a rotating dial
* SLIDE - A set of switches arranged linearly with a sliding mechanism

All switches within the set are numbered, beginning with 1.

{**YES**|**NO**}

Specifies the orientation of the switches. The default is NO (not vertical).

With the REVERSE statement, specifies the order in which DIP switches are numbered. When REVERSE = YES, switches are numbered from 1 to n. When REVERSE = NO (the default), switches are numbered from n to 1.

switchlist

Contains a list or range of switch numbers. You must separate the elements of the list by commas. You specify a range by starting and ending values separated by a hyphen. Switches are in ascending order if REVERSE = YES, or in descending order if REVERSE = NO.

labelist

Contains the ASCII switch labels, up to 10 characters each, that the ECU assigns to the switches in switchlist. The labels must appear in the same order as their corresponding switch numbers in switchlist. Surround each label with quotes and separate them by spaces.

The following statement labels four switches in the range 4 to 1, giving them the labels SW1-4, SW1-3, SW1-2, and SW1-1. The switches are specified in descending order because REVERSE = NO.

REVERSE = NO

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

valuelist

With the INITVAL statement, defines the settings for factory-set switches that must not change. If omitted, the ECU determines the switch settings or the user may set them to any value.

With the FACTORY statement, defines the factory settings for the switches.

For each element of switchlist, a corresponding element in valuelist specifies the setting of the switch, as follows:

* 1 indicates ON.
* 0 indicates OFF.
* x indicates that the switch can be set to either position.

The switch settings are not delimited by spaces.

The following statement sets switches 4 and 1 ON and switch 2 OFF. Switch 3 can be in either position.

INITVAL = LOC(4 3 2 1) 1x01

comment\_string

Specifies up to 600 characters of text to help the user configure the switches.

help\_string

Specifies up to 600 characters of text to display if the user requests help.

### B.2.3    Jumper Configuration Block Statement

The jumper configuration block defines the set of jumpers for an extension board.

### Syntax

**JUMPER( i ) = value   
NAME = name   
JTYPE = { INLINE | PAIRED | TRIPOLE }   
[ VERTICAL = { YES | NO } ]   
[ REVERSE = { YES | NO } ]   
[ LABEL = LOC( jumperlist ) labelist ]   
[ INITVAL = LOC( jumperlist ) valuelist ]   
[ FACTORY = LOC( jumperlist ) valuelist ]   
[ COMMENTS = comment\_string ]   
[ HELP = help\_string ]   
  
.  
.  
.**

i

Uniquely identifies the jumper being configured as a number between 1 and 32767.

value

Specifies either the number of jumper connections (for inline jumpers) or the number of tripole or paired sets (for tripole and paired jumpers).

name

Specifies the jumper name or description, which can be up to 20 characters long.

{**INLINE**|**PAIRED**|**TRIPOLE**}

Specifies the type of jumper, as follows:

* INLINE jumpers, arranged in a straight line such that each post can be connected to an adjacent post
* PAIRED jumpers, arranged as a series of double posts such that any single pair can be connected across the two posts
* TRIPOLE jumpers, arranged as a series of triple posts such that the middle post can be connected to either of the two adjacent posts

{**YES**|**NO**}

With the VERTICAL statement, specifies the orientation of the jumpers. The default is NO (not vertical).

With the REVERSE statement, specifies the order in which jumpers are numbered. When REVERSE = YES, switches are numbered from 1 to n. When REVERSE = NO (the default), switches are numbered from n to 1.

jumperlist

Contains a list of jumper numbers delimited by spaces; paired or tripole jumpers can specify a range of jumpers. You should arrange the elements of jumperlist in ascending order if REVERSE = YES, or in descending order if REVERSE = NO.

labelist

Contains the ASCII labels, up to 10 characters each, that the ECU assigns to the jumpers in jumperlist. The labels must appear in the same order as their corresponding jumper numbers in jumperlist. Surround each label by quotes and separate them by spaces.

valuelist

With the INITVAL statement, defines the settings for factory-set jumpers that must not change. If omitted, the ECU determines the jumper settings or the user may set them to any value.

With the FACTORY statement, defines the factory settings for the jumpers.

The valuelist contains a bit mask that specifies the setting of the paired and inline jumpers in jumperlist. Each bit in valuelist can be one of the following values:

* 1 indicates ON.
* 0 indicates OFF.
* x indicates that the jumper can be set to any position.

For tripole jumpers, each bit in valuelist can be one of the following values:

* 1 indicates ON; the jumper was installed in upper or right position.
* 0 indicates OFF; the jumper was installed in lower or left position, unless otherwise marked.
* n indicates NONE; the jumper was not installed.
* x indicates that the jumper can be set to any position.

The values must appear in the same order as their corresponding jumper numbers in jumperlist, and they should not be separated by spaces.

comment\_string

Specifies up to 600 characters of text to help the user configure the jumpers.

help\_string

Specifies up to 600 characters of text to display if the user requests help.

### B.2.4    Software Initialization Statement Block

The software initialization statement block provides user information and instructions about software drivers for display during system configuration.

### Syntax

**SOFTWARE( i ) = description**

i

Uniquely identifies the software driver as a number between 1 and 32767.

description

Specifies up to 600 characters of text that describes the software. The ECU displays this text along with the switch and jumper settings.

## B.3    Function Statement Block

The function statement block identifies the name and type of the expansion board, plus initializations and system resource requirements for one or more possible configurations.

### Syntax

**GROUP = group\_name TYPE = group\_type   
FUNCTION = func\_name   
[ TYPE = func\_type ]   
[ SHOW = { YES | NO | EXP } ]   
[ CONNECTION = orientation ]   
[ COMMENTS = comment\_string ]   
[ HELP = help\_string ]   
[ choice statement block ]   
[ SUBFUNCTION = subfunc\_name   
function statement block ]   
  
.  
.  
.  
  
  
.  
.  
.  
  
ENDGROUP**

group\_name

Specifies a unique identifier for the group, and can be up to 60 characters long.

group\_type

Specifies a 3-character string that identifies the group type. See the description of func\_type for a list of commonly used types.

func\_name

Specifies the name of the function, which can be up to 100 characters long. All names within a configuration file must be unique, but different configuration files can have common function names. The ECU displays the function name during configuration but does not store it in nonvolatile memory.

func\_type

Specifies a 3-character string that identifies the function type. The following types are commonly used for both groups and functions:

|  |  |
| --- | --- |
| **Identifier** | **Description** |
| 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 |

You should use these types when applicable. However, if you define a new type, it must be 3 characters long and all uppercase.

You can supplement the function type with additional ASCII strings, separated by commas. These additional strings may be more than 3 characters as long as the entire list of types does not exceed 80 characters, including the comma separators.

The ECU stores the TYPE string in nonvolatile memory during configuration. A device driver can use the TYPE string to determine the general class of functionality of a device, and can use the SUBTYPE string to determine the configuration of a device.

{**YES** | **NO** | **EXP**}

Indicates whether the function is displayed during configuration. When SHOW = YES (the default), the function is always displayed. When SHOW = NO, the function is never displayed. When SHOW = EXP, the function is displayed only when the system is in expanded mode.

orientation

Specifies up to 40 characters that describe the orientation of connectors, such as top, bottom, upper, lower, middle, and so on. The ECU can display the connection string if the user requests it.

comment\_string

Specifies up to 600 characters of text to help the user configure the jumpers.

help\_string

Specifies up to 600 characters of text to display if the user requests help.

choice statement block

See [Section B.4](#choice_block).

subfunc\_name

Identifies a set of related components with separate resource or initialization requirements. You can include any statement that is valid for a function statement block in a subfunction block.

## B.4    Choice and Subchoice Statement Blocks

You must accompany each function statement block by at least one choice statement block that specifies the initializations and system resource requirements of a possible configuration. The ECU uses the first choice statement block as the default. You should arrange additional choice statement blocks in the order of preference. As the ECU searches for the best configuration, it picks the first choice statement block that satisfies the configuration requirements.

Subchoice statement blocks handle resource statement alternatives that are too complex for individual CHOICE statements, such as memory configurations. A subchoice statement block can contain any statement that is valid for a choice statement block. The subchoice alternatives must be automatically selectable by the ECU with information available from the configuration files. The ECU does not present subchoice alternatives, although the user can scroll through the resources that the subchoice statement blocks specify.

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

### Syntax

**CHOICE = configuration\_name   
[ SUBTYPE = device\_description ]   
[ FREEFORM = length, n ... ]   
[ DISABLE = { YES | NO } ]   
[ AMPERAGE = amps ]   
[ PORTVAR( j ) = address ]   
[ TOTALMEM = range\_list [ STEP = value ] ]   
[ HELP = help\_text ]   
[ resource description block ]   
[ SUBCHOICE   
choice statement block ]   
  
.  
.  
.  
  
  
.  
.  
.**

configuration\_name

Specifies the name of the configuration, which can be up to 90 characters long. During configuration, the ECU displays all the CHOICE statement configuration names.

device\_description

Specifies a 3-character mnemonic for the configuration associated with the choice. You can supplement the subtype with additional ASCII strings, separated by semicolons. These additional strings may be more than 3 characters as long as the entire list of types does not exceed 80 characters, including the separators.

The SUBTYPE string identifies the configuration of a device; the TYPE string identifies the type of device. The ECU stores the concatenated TYPE and SUBTYPE strings, with a semicolon delimiter, in nonvolatile memory during configuration. The device driver can access those strings to determine the type of device and its configuration.

**length, n ...**

Defines resources in a freeform manner, where length specifies the number of bytes of data, up to 203, and n is the data. A FREEFORM statement cannot be used with any resource description block, but can be used any other statement that is valid within a CHOICE statement block.

{**YES**|**NO**}

Disables or enables the expansion board function defined in the choice statement block. Each function to be disabled must have its own DISABLE = YES statement. The default is NO.

amps

Specifies the maximum amount of continuous 5-volt current (in milliamps) required by this choice. The total 5-volt current includes the amount in the board identification block plus the amount for the selected options. The AMPERAGE statement does not apply to virtual devices.

j

Uniquely identifies an I/O port, initialized in a prior IOPORT initialization statement block.

address

The address of an I/O port initialization block having the same address as PORTVAR( j).

range\_list

Indicates the total amount of memory required by the choice when a memory block can split its allocation between system memory and expanded memory. This can be either a list or a range of values.

value

When you specify a range for STEP, defines the smallest increment by which the user can add additional memory to the board.

help\_string

Up to 600 characters of text to display if the user requests help.

resource description block

See [Section B.5](#resource_block).

## B.5    Resource Description Block

You use resource description blocks within choice and subchoice statement blocks to identify the initialization and system resource requirements of the named configuration. A resource description block groups resource statements and INIT statements based on their interdependencies, as follows:

* LINK groups define elements that have a direct relationship to each other. The selection of one resource determines the other resources in the group and their initialization. Each statement in a LINK group must have the same number of options. If the first option is chosen for one resource, the ECU automatically selects the first option for the other resource and initialization statements.

The following LINK group defines the relationship between the interrupt and DMA channel. When the user or the ECU selects an interrupt or DMA channel, the ECU automatically selects the corresponding resources and initialization. That is, a selection of IRQ = 3 forces the ECU to select DMA = 2 and IOPORT(1) initialzation 00001111b. A selection of DMA = 5 forces the ECU to select IRQ = 4 and IOPORT(1) initialization 11110000b.

LINK

IRQ = 3 | 4

DMA = 2 | 5

INIT = IOPORT(1) 00001111b | 11110000b

* COMBINE groups define elements that have an indirect relationshp to each other. Each resource selection is independent, but the initialization is directly determined by the combination of resource selections. For example:
* COMBINE
* MEMORY = 4M | 8M
* ADDRESS = 1M | 4M
* INIT = IOPORT(2) 00001111b | 01001111b | 10001111b | 11001111b

This COMBINE group defines the following possible combinations:

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

* FREE groups define elements that have no relationship to each other. Each resource selection is independent and the initializations are independent of the resource selections. For example, the following FREE group can be selected independently from all other resource groups:
* FREE
* IRQ 2 | 3 | 4 | 5

### Syntax:

**{ LINK | COMBINE | FREE }   
[ resource statement ]   
[ INIT statement ]   
  
.  
.  
.**

resource statement

See [Section B.6](#resource_statements).

INIT statement

See [Section B.7](#INIT_statements).

## B.6    Resource Statements

Resource statements define the initialization and system resource requrements of the named configuration. The resource statement can define any of the following:

* DMA channel
* Interrupts
* I/O ports
* Memory
* Switches
* Jumpers
* Programmable ports
* Software

The sections that follow describe the syntax for these resource statements.

### B.6.1    DMA Channel Description Block

The DMA channel description block specifies the choice of DMA channels supported, whether the channel can be shared, the channel's data size, cycle timing, and any necessary initialization. You must define multiple DMA channel description blocks for a function with multiple DMA channels that have different share, size, or timing characteristics. An expansion board can request up to four DMA channels. Each channel selected during system configuration is stored in nonvolatile memory with the appropriate characteristics.

### Syntax

**DMA = channel\_number   
[ SHARE = { YES | NO | ident } ]   
[ SIZE = { BYTE | WORD | E-WORD | DWORD } ]   
[ TIMING = { DEFAULT | TYPEA | TYPEB | TYPEC } ]**

channel\_number

Specifies the DMA channel number.

{**YES**|**NO**| ident}

Specifies whether the function can share the DMA channel. The default is NO. Two devices can share a DMA channel if they never require the channel simultaneously. For example, a diskette drive and a tape drive attached to the same controller could share a DMA channel because they never use the channel at the same time.

An ident variable, up to 10 characters long, indicates that the function can share the DMA channel only with a device that has a matching identifier.

{**BYTE**|**WORD**|**E-WORD**|**DWORD**}

Specifies the width of the DMA device data transfer. The default size is BYTE for DMA channels 0 to 3 and WORD for channels 4 to 7.

{**DEFAULT**|**TYPEA**|**TYPEB**|**TYPEC**}

Specifies the bus cycle type that the DMA controller executes during the transfer. The DEFAULT timing is compatible with ISA DMA devices. DMA devices that support EISA bus cycles can use TYPEC (burst) DMA transfers, which provide the highest data transfer rate.

### B.6.2    I/O Port Description Block

The I/O port description block specifies the port address and any initialization.

### Syntax

**PORT = list\_range [ STEP = svalue [ COUNT = cvalue ] ]   
[ SHARE = { YES | NO | ident } ]   
[ SIZE = { BYTE | WORD | DWORD } ]**

list\_range

Specifies a single port address, a list of addresses, or a range of addresses.

svalue

Identifies the address increment of the port selections. If omitted, the ECU allocates the entire range.

cvalue

Specifies the number of ports allocated from the STEP address block. If omitted, the ECU sets cvalue equal to svalue.

{**YES**|**NO**| ident}

Specifies whether the function can share the requested ports. The default is NO.

The ident variable, a 10-character identifier, indicates that only a function with the same identifier can share the port address.

{**BYTE**|**WORD**|**DWORD**}

Specifies the size of the I/O port.

### B.6.3    Interrupt Description Block

The interrupt description block specifies the choice of interrupts supported, whether the interrupt can be shared, whether it is edge or level sensitive, and any initialization.

### Syntax

**IRQ = interrupt\_number   
[ SHARE = { YES | NO | ident } ]   
[ TRIGGER = { LEVEL | EDGE } ]**

interrupt\_number

Specifies the interrupt number (or multiple interrupts) that the configuration supports. 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.

{**YES**|**NO**| ident}

Specifies whether the function can share this interrupt. The default value is NO. For EISA boards capable of sharing interrupts, this field should be YES.

The ident variable, a 10-character identifier, indicates that only the function with the same identifier can share the interrupt.

{**LEVEL**|**EDGE**}

Specifies whether the ROM initializes the interrupt controller to edge- or level-triggered interrupts. The default is EDGE. In most cases, if the SHARE statement is YES, the TRIGGER statement should be set to LEVEL. However, some designs require shared, edge-triggered interrupts.

### B.6.4    Memory Description Block

The memory description block specifies the amount of memory on an expansion board and its starting address, whether the memory is cachable, whether it is RAM or ROM, and the type of memory (system, expanded, virtual, or other). The configuration file can contain up to nine memory description blocks for each function.

### Syntax

**MEMORY = memory [ STEP = step\_value ]   
[ ADDRESS = list\_range [ STEP = step\_value ] ]   
[ WRITEABLE = { YES | NO } ]   
[ MEMTYPE = { SYS | VIR | OTH } ]   
[ SIZE = { BYTE | WORD | DWORD } ]   
[ DECODE = { 20 | 24 | 32 } ]   
[ CACHE = { YES | NO } ]   
[ SHARE = { YES | NO | ident } ]   
  
.  
.  
.**

memory

Specifies a list or a range of memory configurations, with a minimum of 1K and a maximum of 64 megabytes.

step\_value

When a range is specified, defines the smallest increment by which the user can add additional memory to the board.

list\_range

Specifies a list or range of starting addresses of the memory. The ADDRESS statement is optional for memory if MEMTYPE is EXP (expanded) or OTH (other). The ADDRESS statement is required for system and virtual memory.

{**YES**|**NO**}

With the WRITEABLE statement, indicates whether memory is writeable. Specify YES for RAM (the default); NO for ROM.

{**SYS**|**EXP**|**VIR**|**OTH**}

Specifies the memory type, as follows:

|  |  |  |
| --- | --- | --- |
| **Constant** | **Memory Type** | **Description** |
| SYS | System memory | Base and extended memory (the default). |
| VIR | Virtual memory | Address space is used, but no physical memory occupies the address. Access to VIR memory does not generate an address on the EISA bus. |
| OTH | Other | Address space used for memory-mapped I/O or bank-switched memory. OTH is intended primarily for I/O devices, such as network adapters. Include an ADDRESS statement only if it resides in the physical address space. |

{**BYTE**|**WORD**|**DWORD**}

Specifies the memory size.

{**20**|**24**|**32**}

Specifies the number of address lines a memory expansion board decodes. The default is 32.

{**YES**|**NO** [| ident]}

With the CACHE statement, indicates whether the memory contents can be stored in cache memory. The default is NO.

With the SHARE statement, indicates the total amount of memory the CHOICE statement block specifies. This statement is required for a MEMORY block that can split its allocation between system memory, other memory, and expanded memory.

If you include an ident variable (up to 10 characters long), the function can share the memory address range only with a device that has a matching identifier.

## B.7    INIT Statements

INIT statements specify the initializations for alternative configurations. An INIT statement can initialize a DMA, IRQ, IOPORT, or MEMORY block. The ECU determines the initializations for the selected configuration and stores them in nonvolatile memory. The system ROM power-up routine performs the initializations.

### B.7.1    I/O Port INIT Statement

The I/O port INIT statement specifies an I/O port and the binary value to write to the port for configuration. You can specify the address of the port in an IOPORT statement or by a PORTADDR variable.

### Syntax

**INIT = IOPORT( i ) [ LOC( bitlist ) ] valuelist   
INIT = PORTADDR( address ) [ { BYTE | WORD | DWORD } ] list**

i

Identifies the address of the port to be initialized. You define the size of the port in a previous IOPORT statement for the same identifier i; do not define it in the INIT statement.

bitlist

Contains a list or range of bit positions to be initialized. The elements of the bitlist must be in order from most significant bit to least significant bit, with a space as a delimiter.

valuelist

Assigns the binary values to the port.

address

Specifies the address of the I/O port to be initialized. This form of the INIT statement provides a shorthand form for specifying I/O port values when no initialization block is needed.

{**BYTE** | **WORD** | **DWORD**}

Specifies the size of the I/O port. BYTE is the default size.

list

Specifies the values to be assigned to the I/O port. When you use this form, you must specify all bits as follows:

* 0 or 1 indicates that the bit is reserved and must be initialized to the specified value.
* r indicates that the bit value must be read from the port.

### B.7.2    Switch INIT Statement

The switch INIT statement specifies the switch positions and appropriate settings for the configuration.

### Syntax

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

i

Identifies the switch to be initialized.

switchlist

Contains a list or range of switch positions. The elements of the list must be in ascending order if REVERSE=YES, or descending order if REVERSE=NO in the corresponding switch identification block. You must use a space to delimit the elements of the list.

valuelist

Specifies the switch setting of each switch position in switchlist, but you must not use spaces to delimit the elements of the list. You can set a dip switch to 1 for ON or 0 for OFF. You can set a rotary or slide switch by placing a 1 in the appropriate bit position.

### B.7.3    Jumper INIT Statement

The jumper INIT statement defines the jumper positions and the appropriate setting for the configuration.

### Syntax

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

i

Identifies the jumper to be initialized.

jumperlist

Contains a list of jumper positions. For paired and tripole jumpers, jumperlist contains a list or range of jumper positions. For inline jumpers, it indicates the connections between posts, using a caret. For example, LOC(1^2 3^4 5^6) shows that a jumper is set between posts 1 and 2, posts 3 and 4, and posts 5 and 6. The elements of the list must be in ascending order if REVERSE = YES, or descending order if REVERSE = NO in the corresponding jumper configuration block. A space delimits the elements of the list.

valuelist

Specifies the setting for each jumper in the same order as jumperlist, but with no space as a delimiter. You can set paired and inline jumpers to 1 for ON (jumper is installed) or 0 for OFF (jumper is not installed). You can set tripole jumpers to 1 for ON (jumper is installed in upper or right position), 0 for OFF (jumper is installed in lower or left position), or n (jumper is not installed).

### B.7.4    Software INIT Statement

The software INIT statement defines how command-line parameters are substituted in the text of a software initialization statement block.

### Syntax

**INIT = SOFTWARE( i ) parameter**

i

Identifies the software statement block to be initialized.

parameter

Specifies the text to be substituted.

## B.8    System Board Configuration

System boards require special configuration files that contain additional information not required by expansion boards. 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 board.

### B.8.1    System Board Identification Block

The board identification block for system boards uses the same syntax as an expansion board (see [Section B.1](#board_id_block)), but you must set the CATEGORY field to SYS and the SLOT field to EMB(0).

### Syntax

**BOARD ID = board\_id   
NAME = system\_board\_name   
MFR = manufacturer\_name   
CATEGORY = SYS   
SLOT = EMB(0)   
AMPERAGE = value**

### B.8.2    System Description Block

The system description block describes characteristics of the system such as the size of nonvolatile memory. It must immediately follow the board identification block.

### Syntax

**SYSTEM   
[ NONVOLATILE = memory ]   
[ AMPERAGE = amps ]   
[ SLOT(1) = { ISA8 | ISA16 | EISA | OTH | ,"type" } ]   
[ LENGTH = length ]   
[ SKIRT = { YES | NO } ]   
[ BUSMASTER = { YES | NO } ]   
  
.  
.  
.**

memory

Specifies the total number of bytes of EISA nonvolatile memory, not including 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.

Use the following equation to determine the minimum amount of EISA nonvolatile memory required:

**NVRAM = ( Slots + Sys + Edev + Vdev ) \* 340**

where:

* Slots is the number of expansion connections, and is a whole number between 1 and 15.
* Sys is the system board, EMB(0).
* Edev is the number of embedded devices on the system board, and is a whole number between 1 and 15 (the number of physical slots).
* Vdev is the number of system board virtual devices, if any.

For example, if you had a system with one system board, 8 physical devices, 1 embedded device, and 2 virtual devices, for a total of 12 devices, you would calculate the amount of nonvolatile memory as follows:

**NVRAM = 12 \* 340 = 4080 bytes**

amps

Specifies the total amount of 5-volt power (in milliamps) available to expansion devices installed on the expansion bus.

{**ISA8**|**ISA16**|**EISA**|**OTH**[**,"**type **"**]**...**}

Specifies an expansion slot. This statement does not apply to the system board, embedded devices, or virtual devices when included as part of the system description block.

You can include an optional text string (type) enclosed in quotation marks to describe the slot.

length

Accompanies a SLOT statement to specify the maximum length board that the user can install in the slot. System boards should include a LENGTH statement so that the ECU can optimize expansion board slot allocation. If omitted, the ECU assumes the maximum length of 341 millimeters and assigns slot numbers without regard to length.

{**YES**| **NO**}

With the SKIRT statement, specifies whether a slot supports a skirted expansion board. The default is YES.

With the BUSMASTER statement, specifies whether an EISA slot accepts a bus master expansion board. The default is YES.