MSA for the iRMX Operating System

RadiSys Corporation
5445 NE Dawson Creek Drive
Hillsboro, OR 97124
(503) 615-1100
FAX: (503) 615-1150
www.radisys.com
07-0703-01
December 1999

.

October 1999

Quick Contents

Notational Conventions

This manual uses the following conventions:

- All numbers are decimal unless otherwise stated. Hexadecimal numbers include the `H` radix character (for example, `0FFH`).
- Bit 0 is the low-order bit unless otherwise stated.
- `Computer output is printed like this.`
- `User input appears like this.`
- **System call names and command names appear in bold.**
- The logical name *:rmx:* in a pathname replaces */rmx386/* or */rmx286/* as required for the specific file being accessed.
- The notation <CR>, in upper- or lower-case, indicates a carriage return at the end of a line. An instruction to enter a value means press <CR> at the end of the line.
- Items shown within angle brackets (< >) represent variable information, either displayed by a command, or for which you enter the appropriate value.
- The notation <CR>, in upper- or lower-case, indicates a carriage return at the end of a line. An instruction to enter a value means press <CR> at the end of the line.
- The logical name *:rmx:* in a pathname replaces */rmx386/* or */rmx286/* as required for the specific file being accessed.
- The following operating system layer-specific abbreviations are used.

  AL Application Loader

  BIOS Basic I/O System

  EIOS Extended I/O System

  HI Human Interface

  UDIUniversal Development Interface

Note

Notes indicate important information.

CAUTION

Cautions indicate situations which may damage hardware or data.

Contents

Tables

Figures

Introduction 1

This manual explains how to set up boot systems Multibus II systems. A *boot system* is a set of files that enables bootstrap loading of the iRMX® III Operating System. This set of files includes a *target file* for each host and support files needed for bootstrap and for the operating system. A target file is also known as a *bootable image*.

The instructions in the *Installation and Startup* manual describe default, automatic bootstrap. They explain how to configure a default boot system for the I/O server and CPU boards. By using this manual you can learn how to do the following:

- Change the default boot process by modifying some bootstrap parameters
- Add multiple CPU boards to the Multibus II System and change the boot scenario to include them

The boot scenarios described in this manual are for the iRMX III Operating System and are compatible with Multibus II System Architecture (MSA). To understand this manual, you will need to refer its companion manuals.

See also: Installing the Software on Multibus II Systems, *Installation and Startup*;
MSA Bootstrap Models, *Multibus II System Architecture Bootstrap Specification*;
MSA firmware, *Firmware User's Guide for MSA Firmware*

The rest of this chapter describes:

- Why you might change the bootstrap process
- The four phases of initialization
- The MSA bootstrap process

Why You Might Change the Bootstrap Process

Chapter 3 of this manual provides six examples of MSA boot scenarios, each of which demonstrates something unique. The variety of bootstrap methods of Multibus II provides a level of flexibility not available in Multibus I or PC-based systems. These scenarios exploit the power of distributed computing and are examples of how to customize a system for maximum performance.

What Happens After Power-up or Cold Reset

MSA defines the sequence of events that occurs after power-up, which is the same as a cold reset. Figure 1-1 shows the four basic parts of initialization: reset, board initialization, system initialization, and bootstrap loading. Figure 1-2 on page 4 shows the sequence of events from a cold reset to booting.

- **Reset:** Initialization begins when power is applied to the system.
- **Board Initialization:** Each board verifies its own functions.
- **System Hardware Initialization:** One board assumes control of the Multibus II System and verifies that the system hardware is working. During this phase, you can change the bootstrap parameters.
- **Bootstrap Loading:** From its own non-volatile memory, each board invokes code that finds the device to boot from and loads the operating software into its memory.

The next section describes each of these phases in more detail.

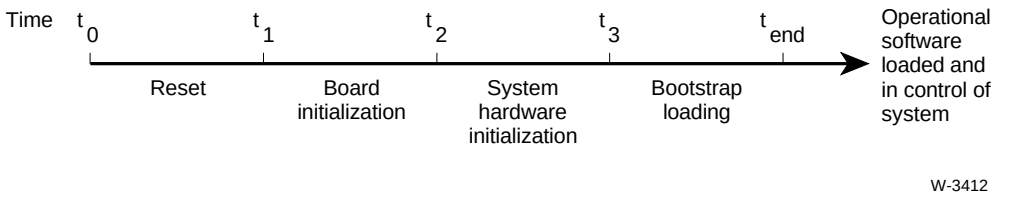

Figure 1-1. Four Phases of Initialization

Reset

At power-up, the system receives a cold reset, which resets all the hardware in the system.

The reset phase signals the beginning of initialization for the system. The Central Services Module (CSM), found in every Multibus II system, handles initialization procedures during this phase. The reset phase ends when the Reset signal (RST*) on the Parallel System Bus (PSB) is deasserted.

Board Initialization

Each board tests itself, first running initialization checks on its own interconnect controller. The interconnect controller is hardware that handles the interconnect address space. Then the board checks the processor.

If all the checks pass, the board runs Built-In Self Tests (BISTs) that verify the rest of the board. Each board reports a pass/fail result for the checks and BISTs in its interconnect space, where the results can be read by other boards during subsequent phases. This phase ends when the processor deasserts the Reset Not Complete signal (RSTNC*) on the PSB.

System Hardware Initialization

The independent boards then begin to communicate with each other through interconnect space. They use a standard method for selecting one of the boards to control the system tests. The selected board, called the master, instructs the other boards to run further tests on themselves and across the PSB. At the end of system hardware initialization, the master isolates any faulty board by putting it in a reset state; this prevents a faulty board from interfering with the bootstrap process.

While the master is in control of the system, you can request the master to run additional tests. Also, you can influence the booting process by changing bootstrap parameters or by attempting to boot. You can communicate with the system through a console connected to the master.

System hardware initialization ends when the master sets the BIST Complete Bit on all boards.

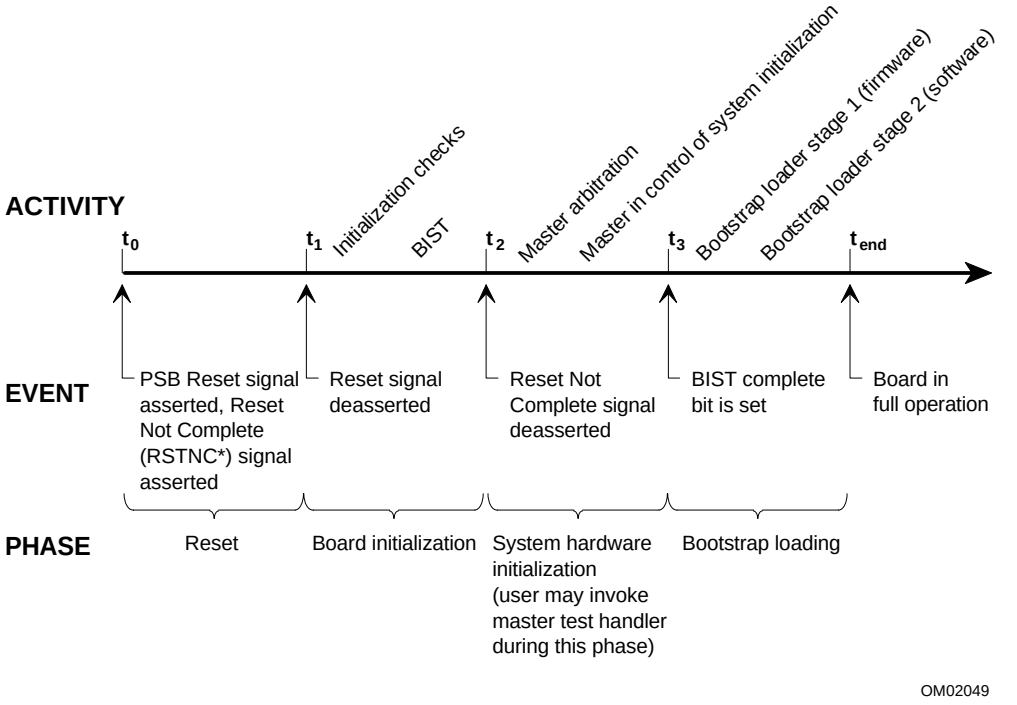

Figure 1-2. Sequence of Events from Cold Reset to Booting

Bootstrap Loading

In the final stage of initialization, the master passes control to the next piece of firmware, which is the first stage bootstrap loader. The first stage invokes the second stage, which loads the operating system.

First Stage Bootstrap Loader

When the first stage loader is invoked, the boards again become a group of independent computers, but at this stage they can communicate across the PSB. All the boards that are enabled for bootstrap invoke their own first stage bootstrap loader.

The first stage loader is in non-volatile memory on each of the boards and is independent of the operating system. The first stage loads the second stage from the mass storage device, if the second stage is present and valid.

Second Stage Bootstrap Loader

The **second stage** loads the operating system. The type of second stage loader depends on the operating system and the type of booting needed. The iRMX Operating System provides two types of second stage loaders:

- the disk second stage loader for independent and quasi-independent booting
- the dependent second stage loader for dependent booting

The disk second stage is installed on either the diskette or hard disk when it is formatted. Although the dependent second stage loader theoretically could be in any file that the boot server can access, the standard pathname for iRMX III systems is */msa32/stage2.rmx*.

The second stage bootstrap process can follow one of three methods: independent, quasi-independent, and dependent. The bootstrap loading phase ends with the system in full operation.

See also: page for definitions of the three bootstrap methods;
Testing a Board or System, *Firmware User's Guide for MSA Firmware*;
Initialization Phases, *Multibus II Initialization and Diagnostics Specification*

❑❑❑

Bootstrap Parameters 2

The operating system supplies two bootstrap loaders:

- **MSA Bootstrap Loader**. This loader can load the iRMX III Operating System on Multibus II systems. In general, Multibus II applications use this loader. The MSA bootstrap loader executes in Protected Virtual Address Mode (PVAM), and is described in this manual.
- **iRMX Real Mode Bootstrap Loader**. This loader can load the iRMX III Operating System on Multibus I systems.

  See also: *iRMX Bootstrap Loader Reference Manual* for more information on this loader

This chapter describes MSA bootstrap parameters, lists those used by the bootloaders and the MSA boot server, and tells how to modify the parameters. To use the booting examples in Chapter 3 you need to understand the basic concepts from this chapter.

What Are Bootstrap Parameters?

Bootstrap parameters are variables that control the bootstrap process. A bootstrap parameter consists of a parameter name, an equal (=) sign, and a parameter value. Various firmware and software modules read and set the bootstrap parameters at different times during the bootstrap process.

To set parameters, a firmware/software module sends a value to the Bootstrap Parameter String (BPS) Manager. The manager decides if the parameter is valid, then checks if the parameter is in RAM. If the parameter is already in RAM, the manager decides if the new value has higher precedence than the previous value. The order of precedence depends on the source of the new values, as follows:

1. Runtime
2. Operator-entry
3. BPS file (defined on page 9)
4. Non-volatile memory

If the new source has higher precedence, the manager puts the parameter in the BPS save area.

To read the parameters, the various firmware and software modules request them from the BPS Manager. The following firmware and software modules use or manage bootstrap parameters:

Firmware (EPROM)	**Software (Loaded from Mass Storage)**
MSA First Stage Bootloader	MSA Second Stage Bootloader
BPS Manager	Extended I/O System (EIOS)
Bootstrap device drivers	MSA boot server
	SPS Manager
	PCI Device Drivers
	ATCS Device Drivers
	Multibus II Downloader Job
	iNA 960 MIP Job

Bootstrap Parameters

This section describes the server and client parameters. You can enter bootstrap parameter names and their values in uppercase or lowercase.

The tables in this section use these terms:

BPS file	Bootstrap parameter string file, located on the hard disk. If you want to change the boot process, but don't want to type the changes after every reset, edit the BPS file. Its default filename is */msa/config/bps*.
BPS save area	Bootstrap parameter string save area, located in RAM. If you want to change the boot process temporarily, use the Master Test Handler (MTH) to write a new parameter to the BPS save area. When the Multibus II system is turned off or receives a cold reset, it loses these changes.
Where To Set	Lists where you can modify the parameter; in the BPS file or the BPS save area (using the MTH).
Default	The value used if you don't change the parameter.
Valid Values	The values to use if you change a bootstrap parameter, either in the BPS file or by using the MTH.
QI Master	Quasi-independent master. The QI master is the host responsible for running a configuration server.

See also: Bootstrap parameters, *Multibus II System Architecture Bootstrap Specification*

Server Parameters

Table 2-1 lists the server bootstrap parameters. You must enclose all server parameters within square brackets. For example, [bl_method = quasi]. If you enter more than one parameter, they must be separated by semicolons.

Note

You cannot change server parameters using the MTH.

Table 2-1. Server Parameters

Name	Where To Set	Read By	Valid Values	Default
[bl_host_id]	BPS file	Second Stage	valid host ID or global	none
	BPS file	Boot Server	valid host ID or global	none
[bl_method]	BPS file	Boot Server	dependent or quasi	none
[bl_second_stage]	BPS file	Boot Server	valid pathname	none

`bl_host_id`

Specifies the host to which the following parameters apply. For example, the following parameters all apply to host 3:

```
[bl_host_id = 3....]
       bl_debug = on;
       rq_sd = GSCW5_2;
```

The second stage and boot server use this value to select a set of parameters from the BPS file. If `bl_host_id` is set to `global`, the remaining parameters in the group apply to all hosts. However, if any global parameters are also defined for an individual host, the individual's parameter takes precedence.

`bl_method`

Controls how the boot server responds to the boot client. The choices are quasi-independent and dependent booting.

If set to quasi, the boot server responds to `locate config server` requests from the client. It offers a subset of the boot server called the configuration server.

If set to dependent, the boot server responds to `locate boot server` requests from the client.

If not defined, the boot server responds to either `locate boot server` or `locate config server` requests if the client making the request has a valid entry in the BPS file. A valid entry must be in the BPS file; it looks like this:

```
[bl_host_id = x;
             bl_method = method]
```

where

x	is the host ID of the client requesting service
method	is the bootstrap method for this client

Either the server or the client may control the method of bootstrap.

`bl_second_stage`

The pathname of the file containing the dependent second stage for the boot client, defined either globally or for a specific host. For example:

```
[bl_host_id = global;
       bl_second_stage = /msa32/stage2.rmx]
```

or

```
[bl_host_id = 3;
       bl_second_stage = /msa32/stage2.rmx]
```

This parameter must be defined in the BPS file.

Either the server or the client can control which second stage is used. If this parameter is not defined, the client parameter of the same name must be set on the boot client.

See also: Client parameter `bl_second_stage`, page 13

Client Parameters

Table 2-2 lists the client bootstrap parameters. You can enter client parameters in either the BPS file or the BPS save area.

To specify client parameters in the BPS file, separate them by semicolons and do not put them in brackets. To specify client parameters in the BPS save area, use the MTH.

Table 2-2. Client Parameters

Name	Where To Set	Read By	Valid Values	Default
bl_boot_device	BPS save area only	First Stage	See Table 2-3	none
	BPS save area only	Second Stage	See Table 2-3	none
bl_boot_logical_part	BPS save area	Second Stage	1 or more	none
bl_boot_master_part	BPS save area	Second Stage	1 to 4	Active iRMX partition
bl_config_file	BPS save area	Second Stage	valid pathname	msa/config/bps
	BPS file or BPS save area	Boot Server	valid pathname	Error occurs if not set by 2nd stage.
bl_debug	BPS file or BPS save area	Second Stage	on off	off
bl_device_type	BPS save area	First Stage	DISK FLOPPY TAPE WINI NIL	none
bl_error_action	BPS save area	First Stage	INT1 INT3 HALT RESTART	INT1
bl_host_id	BPS save area (set by First Stage)	Second Stage	must be host slot ID	host slot ID (reserved, do not change)

continued

Table 2-2. Client Parameters (continued)

Name	Where To Set	Read By	Valid Values	Default
bl_location	BPS save area	First Stage	SCSI NIL	none
bl_method	BPS save area	First Stage	independent dependent quasi	none
bl_qi_master	BPS file or BPS save area	Boot Server	host ID of QI master	none
bl_quasi_server_id	BPS save area	First Stage	valid host ID 0FFFFH	search (0FFFFH)
bl_retry_count	BPS save area	First Stage	0 to 0FFFEH 0FFFFH	forever (0FFFFH)
bl_second_stage	BPS save area	First Stage	valid pathname	none
bl_server_id	BPS save area	First Stage	valid Host ID 0FFFFH	search (0FFFFH)
bl_target_file	BPS file or BPS save area	Second Stage	valid pathname	msa/boot/rmx
bl_unit	BPS save area	First Stage	valid unit number 0FFH	See page 18

`bl_boot_device`

The first stage bootstrap loader sets this parameter. After the second stage starts executing, the board that has the MTH prints the parameter value on the console screen, telling you which device it is using for bootstrap.

This value is also used as the system device if `rq_sd` is not set.

Table 2-3. Possible Parameters for bl_boot_device

Parameter	Device	Boot Method
IMM[1]	Factory test device	Independent
GSCW5_2	SCSI hard drive (SCSI ID 2 only)	Independent
SCW	SCSI hard drive	Independent
SCF	SCSI diskette drive	Independent
SCT	SCSI tape drive	Independent
SCD	Any SCSI device	Independent
PCD	PCI disk drive	Quasi-independent
PCW	PCI hard disk drive	Quasi-independent
PCF	PCI diskette drive	Quasi-independent
PCT	PCI tape drive	Quasi-independent
DEP	An MSA boot server supplied by another board	Dependent

1 For factory test only. Do not use.

To find the boot device, the first stage uses the `bl_boot_device` parameters in the order listed in Table 2-3; it boots from the first one it finds.

The first entry in the I/O server firmware is GSCW5_2. All other entries have `bl_unit` set to 0FFH meaning search for the first available unit. Because most Multibus II systems have one SCSI hard disk with a PCI unit number of 2, this entry was created and put at the beginning of the boot device table so that the first stage would first try to boot from this particular hard disk rather than searching for the first available device. Multibus II systems boot faster this way.

`bl_boot_logical_part`
`bl_boot_master_part`

The second stage bootstrap loader by default boots from the active partition. You can use the MTH to specify one or both of these parameters to indicate a different partition from which to boot. This applies to independent or quasi-independent boot. A partitioned iRMX volume contains a partition table which can have up to four master partition entries. Specify `bl_boot_master_part` to indicate one of the master partitions. The second through the fourth master entries can have logical extended partitions. Specify both `bl_boot_master_part` and `bl_boot_logical_part` to indicate one of the extended logical partitions. If the iRMX volume is a single partition, do not set these BPS parameters.

`bl_config_file`

assigned	the second stage uses it as the file name for the BPS file
unassigned	the second stage uses *msa/config/bps*, which is its default value. It contains values for the bootstrap parameters. If you enter a value, the new value overrides the default.

The boot server uses this value as the pathname for the BPS file. If the value is not set to a valid pathname when the boot server initializes, a fatal error occurs and the boot server is deleted.

Note

To boot a dependent board that hosts an MSA boot server, you must specify a `bl_config_file` parameter for the boot server. This parameter is required by the boot server and takes effect only after the boot server is running. There are two ways to specify the `bl_config_file` parameter:

- Set it using the MTH **mp** command
- Set it in the BPS file for the boot server that hosts the dependently booting board

`bl_debug` Causes a break to the debug monitor after loading a target file and before executing it.

on	the second stage tells the operating system to invoke the monitor before executing
off, or not set	the target file executes without breaking to the monitor

`bl_device_type`

DISK	Boot from either hard disk or diskette
FLOPPY	Boot from diskette
TAPE	Boot from tape
WINI	Boot from hard disk
NIL	Boot device not specified or, as for dependent boot, not applicable

`bl_error_action`

INT1	Bootstrap error routine uses interrupt 1
INT3	Bootstrap error routine uses interrupt 3 (default)
HALT	Bootstrap error routine causes a processor HALT
RESTART	Bootstrap error routine restarts boot process

`bl_host_id`

The first stage sets this parameter in the BPS save area. The value is always equal to the slot ID.

The second stage uses this parameter to read the host's bootstrap parameters from the BPS file.

If the user sets this parameter, it is ignored. You can use the MTH **dp** command to display this parameter. If you use the **mp** command to change the parameter, the first stage overwrites your entry.

`bl_location`

SCSI	Boot from SCSI interface
NIL	I/O interface not specified

`bl_method`

INDEPENDENT	Boot from local boot device
DEPENDENT	Boot from boot server
QUASI	Boot quasi-independently

If this parameter is set in the client's local BPS save area, it controls how the boot client will bootstrap. For example, if it is quasi, the boot client only tries a quasi-independent boot.

`bl_qi_master`

If set, the value is the Host ID of a host that acts as QI master for the quasi-independent boot process. The QI master bootstraps without needing a configuration server. It hosts a configuration server or a boot server for other boards to use for their booting.

When `bl_qi_master` is set, the boot server only responds to `locate config server` requests from the host with this Host ID (slot number). The boot server responds with the status message `e_qi_master`. This message tells the host to act as the QI master and go directly to the PCI server for bootstrap.

When not set, the default QI master is the system boot master.

`bl_quasi_server_id`

valid server ID	Host ID of I/O server (Quasi-independent bootstrap)
0FFFFH	Poll all hosts for an I/O server

`bl_retry_count`

0 to 0FFFEH	Number of times to try a boot device
0FFFFH	Try forever (default)

`bl_second_stage`

This is the file name of the second stage, used to request the second stage from the boot server. If the value is not set, the boot server looks in its BPS file for a second stage file name as a server parameter.

`bl_server_id`

valid Host ID	Host ID of boot server (for dependent boot)
0FFFFH	Poll all boot servers

`bl_target_file`

assigned	the second stage uses the value as the file name of target to load
unassigned	the second stage uses */msa/boot/rmx*, which is its default value. It must contain an iRMX target file.

`bl_unit` The unit ID depends on the type of booting you use. Independent booting uses the SCSI IDs shown below. Quasi-independent booting uses the PCI IDs.

unit number	Physical or logical unit number of device to boot from
0FFH	Poll all devices

Intel Systems Device	**SCSI Target:LUN**	**Unit # for Independent Booting**	**Unit # for Quasi-Independent Booting**
Diskette	0:0	0	0
	0:1	1	8
Hard Drive	2:0	10H	2
	3:0	18H	3
	4:0	20H	4
	5:0	28H	5
Tape Drive	6:0	30H	6

The SCSI:LUN values shown above correspond to the SCSI ANSI specification. The first digit is the SCSI target ID and the second digit is the Logical Unit Number (LUN) on that target.

The unit numbering convention for quasi-independent booting is described in the PCI driver description in the Interactive Configuration Utility (ICU) help screens.

SPS Parameters

Table 2-4 lists the system parameter string (SPS) parameters. These parameters are specific to the iRMX operating system . You can modify them in either the BPS file or the BPS save area.

To specify SPS parameters in the BPS save area, use the MTH. To specify SPS parameters in the BPS file, separate them by semicolons and do not put them in brackets, using the following format:

```
major_parameter = minor_param_1:value,...,minor_param_n:value;
```

Where:

`major parameter`
: Tells the operating system which device information table or set of configuration variables to change. The names of the major parameters for the PCI and ATCS device drivers are determined by the DEV parameter on the driver ICU screen.

`minor parameter`
: Tells the operating system which device information table field or configuration variable to change. Most Multibus II board names are entered into the interconnect registers in upper case characters using the `bnam` minor parameter. Most board name SPS entries need to be in upper case.

`value`
: The new value that replaces the device information table field or configuration variable.

For example, this example line from a BPS file defines the `rq_pci_a` major parameter by defining the `bnam` minor parameter:

```
rq_pci_a = bnam:486/166SE,bin:1,sin:0;
```

Table 2-4. SPS Parameters

Name	Where to Set	Used By	Valid Values	Default
rq_atcs_con rq_atcs_a rq_atcs_b rq_atcs_c rq_atcs_d	BPS file BPS save area	ATCS Driver	Board Name Board Instance Slot ID	ICU definition file value
rq_dlj	BPS file BPS save area	Multibus II Downloader	iRMX Device Name File Driver	ICU definition file value
rq_hscf	BPS file BPS save area	Human Interface	:config:r?init file replacement	*:config:r?init* file
rq_hterm	MTH BPS save area	Human Interface	:config:terminals file replacement	*:config:terminals* file
rq_mip[1]				
rq_mip_xx[2]	BPS file BPS save area	iNA 960 MIP Job	Board Name File Name iRMX Device Name Load Method	See page 24
rq_rnet_c	BPS file BPS save area	iRMX-NET	Board Name Board Instance	ICU definition file value
rq_rnet_s	BPS file BPS save area	iRMX-NET	Board Name Board Instance	ICU definition file value
rq_pci_a rq_pci_b	BPS file BPS save area	PCI Driver	Board Name Board Instance Server Instance	ICU definition file value
rq_sd	BPS file BPS save area	Second Stage	iRMX physical device name or remote device of System Device	bl_boot_device

1 Obsolete. Use rq_mip_xx to set MIP values; use rq_rnet_c and rq_rnet_s to set iRMX-NET values.
2 The value of XX denotes the board instance, ranging from 00 to 19. For more information, refer to the description of rq_mip_xx on page 23.

```
rq_atcs_con
rq_atcs_a
rq_atcs_b
rq_atcs_c
rq_atcs_d
```

The parameter names you use to change the ATCS driver device information fields. You can change the attributes of the serial controller the ATCS driver uses when the physical device is attached, without reconfiguring the operating system. If you have changed the DEV parameter or are not using a standard definition file, use the device name specified for the DEV parameter on the D410 ICU screen(s). The three minor parameters that you can change are as follows:

`bnam` Specifies the terminal controller board name. The related configuration option on the D410 ICU screen is BID.

Values: From 1 to 10 characters that must match the value specified in the board ID registers in the interconnect space on the terminal controller board. The case of the characters must match. For example:

String	**Board**
186/410	186/410 controller
486/150	486/150 board with an MPI 450 board
MIX486DX66	MIX 486DX66 baseboard with a MIX 450 controller module

`bin` Identifies a particular board in a system containing multiple boards with the same board name. The related configuration option on the D410 ICU screen is IN. The board in the lowest slot has a board instance of 1. The board of the same name in the next higher slot has a board instance of 2, and so on.

Value: [0-21] Use an H to indicate hex values. 0 means that this parameter is ignored and the slot parameter `sid` is used.

See the ATCS driver in the ICU help screens for more information on the default ATCS driver configuration in the standard ICU definition files.

`sid` Specifies the Multibus II slot ID in which the ATCS server board resides. It is used only if the `bin` parameter is 0.

Value: [0-20, 31] 31 means that the ATCS driver uses the ATCS server on the same host.

`rq_dlj`

This is the major parameter name you use to change the Multibus II Downloader Job configuration variables. You can change the attributes of the device from which the file */rmx386/config/dload.mb2* is loaded, without reconfiguring the operating system. The file name cannot be changed. This parameter applies only to the ICU-configurable downloader job and not the dload cusp. The two minor parameters you can change are as follows:

`dev` Specifies the physical device name (DUIB name) of the storage device containing the configuration file for the Downloader Job. The related configuration option on the DLJ ICU screen is SD.

Value: [1-14 Chars]

`fdvr` Specifies the file driver for the storage device specified by the `dev` parameter. The related option on the DLJ ICU screen is FD.

Value: Specify `remote` if it is a remote device, otherwise, specify `named` for this parameter.

See also: dload, downloader job, *System Configuration and Administration*

`rq_hscf`

Use this parameter to specify an alternative initialization file in place of *:config:r?init*. When you specify this parameter in a specific section of the BPS file, the HI on that board uses the alternative file for its initialization. This allows you to use different initialization sequences for different boards. For example, if you don't want to use the default *loadinfo* file for a board, use a replacement for *r?init* that submits a different file than *:config:loadinfo*. Use this syntax:

```
rq_hscf=file:filename;
```

`filename` The name of a file in the *:config:* directory. Do not specify *:config:* as part of the filename.

`rq_hterm`

Use this parameter to specify an alternative terminal initialization file in place of *:config:terminals*. You can use this parameter to recover from a situation where the HI will not initialize because the *terminals* file is invalid or has become corrupted. Prepare a replacement file in the *:config:* directory that contains only a generic terminal device name, such as `t0`. Then, if the HI fails to initialize using the *:config:terminals* file, you can enter the `rq_hterm` parameter in the Master Test Handler (MTH) during the boot sequence to point to your generic alternative file. At the MTH prompt, use this syntax:

```
rq_hterm=file:filename
```

`filename` The name of a file in the *:config:* directory. Do not specify *:config:* as part of the filename.

`rq_mip`

> **Note**
>
> This parameter is obsolete. Use the `rq_mip_xx` parameter to set values for the MIP job, and use the `rq_rnet_c` and `rq_rnet_s` parameters to set values for the iRMX-NET jobs.

`rq_mip_xx`

The name you use to change the iNA 960 MIP job configuration variables. This parameter allows you to change the attributes of the Ethernet controller your iNA 960 application uses, without reconfiguring the operating system. These parameters are valid only for the Multibus II iNA 960 COMMengine jobs, either configured into the iRMX target image or sysloaded as file *icemb2.job*. You can change the following minor parameters:

`00 - 19` Replace xx with a number that specifies the iNA 960 server instance in your Multibus II system. Server instances must be in ascending, consecutive order starting at 00. This value overrides the CBI parameter on the CEBI screen of the ICU.

`bnam` Specifies the Ethernet board name that resides in the system; overrides board names specified on the CEBN screen of the ICU.

Values: From 1 to 20 characters which must match the value specified in the board ID registers in interconnect space on the Ethernet board. The case of the characters must match. For example:

String	**Board**
186/530	186/530 board
486/166SE	486/166SE board
P5120ISE	P5120ISE board
SBCP5090	SBCP5090 board
MIX486DX66	MIX 486DX66 baseboard board

`file` Specifies the pathname of the iNA 960 load file for non-MSA communication boards; overrides the FN parameter on the CEBI screen of the ICU. This parameter is only relevant if the `load` minor parameter is set to `local` (in the ICU) or `load` (in the BPS file).

Values: [1-80 Chars]

`dev` Specifies the name of the device on which the iNA 960 software resides; overrides the DN parameter on the CEBI screen of the ICU. For example, you might set the device to `SD` or `scw_2`. This parameter is only relevant if the `load` minor parameter is set to `local` (in the ICU) or `load` (in the BPS file).

Values: [1-14 Chars]

`load` Specifies how the COMMengine is loaded with the iNA 960 Transport Software. This value overrides the LD parameter on the CEBI screen of the ICU.

Values:

`load`	specifies that the iNA 960 software loads from the local disk storage (same as "local" when specified in the LD parameter of the ICU).
`noload`	specifies that the COMMengine is loaded by some mechanism other than iRMX-NET.
`prom`	specifies that the iNA 960 software is in PROM.

The default parameters for the preconfigured iNA 960 MIP Job are shown below.

Major Parameter	**Minor Parameters**
rq_mip_00	bnam=486/166SE load=NOLOAD
rq_mip_01	bnam=MIX486DX66 load=NOLOAD
rq_mip_02	bnam=MIX386/560 load=LOAD dev=GSCW5_2 file=/net/ina560n.32L
rq_mip_03	bnam=186/530 load=LOAD dev=GSCW5_2 file=/net/ina530n.32L

`rq_rnet_c`
`rq_rnet_s`

Use these to change the name of the iRMX-NET Client (`rq_rnet_c`) and Server (`rq_rnet_s`) jobs, either when configured as first-level jobs in the ICU or the loadable jobs *remotefd.job* (client) and *rnetserv.job* (server).

`bnam` The name of the Ethernet board from which the iRMX-NET client or server takes iNA 960 COMMengine services This overrides the CBN parameter from the RCJ (client) or RSJ (server) screen of the ICU.

Values: From 1 to 20 characters which must match the value specified in the board ID registers in interconnect space on the Ethernet board. The case of the characters must match. For example:

String	**Board**
186/530	186/530 board
486/166SE	486/166SE board
P5120ISE	P5120ISE board
SBCP5090	SBCP5090 board
MIX486DX66	MIX 486DX66 baseboard board

`bin` The board instance of the particular type specified in the `bnam` parameter. The first instance is number 1 and the second instance (in a higher slot number) is 2, etc. This overrides the CBI parameter from the RCJ (client) or RSJ (server) screen of the ICU.

Values: [0-20] Use an H to indicate hex values.

`rq_pci_a`
`rq_pci_b`

The names you use to change the PCI driver device information fields. These parameters allow you to change the attributes of the SCSI controller that the PCI driver uses when the physical device is attached. There are two sets of PCI driver parameters to access two different PCI Servers in all standard ICU definition files. If you have changed the DEV parameter or are not using a standard ICU definition file, use the device name specified for the DEV option on the DPCI ICU screen(s). The parameters apply only to the ICU-configured PCI server job and not the loadable PCIDRV driver. The minor parameters that you can change are as follows:

`bnam` Specifies the name of the I/O Server board in your system. The related configuration option on the DPCI ICU screen is BI.

Values: From 1 to 10 characters that match the values specified in the board ID registers in the interconnect space on the I/O Server board. The case of the characters must match. For example, the string "P5120ISE" specifies the P5120ISE board and the string "486/166SE" specifies the 486/166SE board.

`bin` Indicates if there are other boards in this system of the same board type. The related configuration option on the DPCI ICU screen is IN.

Values: [0-31] Use an H to indicate hex values.

`sin` Specifies the instance of the PCI Server on the controller. The related configuration option on the DPCI ICU screen is SIN. A single controller board may host multiple instances of PCI Servers; the `sin` parameter identifies the server instance within the board. For example, the 386/258D board has two PCI servers; one for the single-ended SCSI channel (`sin:0`) and one for the differential SCSI channel (`sin:1`).
Value: [1-32] Use an H to indicate hex values.

`rq_sd`

The physical device name or remote file server name used by the EIOS when it attaches the iRMX system device (if Automatic Bootstrap Device Recognition is enabled). The system device always has the logical name *:sd:*. The file driver associated with the system device cannot be changed.

assigned	the second stage passes this value to the target
unassigned	the second stage uses the value of `bl_boot_device` and passes it to the target

Using the MTH to Change Parameters

By using MTH commands, you can change the MSA bootstrap parameters. Only a few of the commands are discussed here. To access the MTH, use the following steps:

1. After turning on power or performing a cold reset, the system starts initializing. When you see the first characters on the screen, type:

   ```
   u
   ```

   You will see a screen similar to this:

```
MULTIBUS SYSTEMS ARCHITECTURE Master Test Handler  ID: 515723
Copyright 1991,  Intel Corporation
                                          Reset Type: COLD
SYSTEM CONFIGURATION AND BIST STATUS.........................PASS

  1 --> Run System Diagnostics
  2 --> Go to Operator Interface (Selected if no character entered)
  3 --> Go to Boot Phase

Enter number:  ?
```

2. At this point, type:

   ```
   2
   ```

   and you will see the MTH prompt:

   ```
   MTH [0]
   ```

   The [0] in the MTH prompt indicates that the master board for the initialization process is in slot 0. To see the help menu, type:

   ```
   MTH [0] H <CR>
   ```

Table 2-5 describes the commands which are useful for changing bootstrap parameters. The default slot tells the MTH which board to use when it carries out the command. Changing the default slot does not change the prompt. You may type the commands in uppercase or lowercase.

See also: MTH commands, *Firmware User's Guide for MSA Firmware*

Table 2-5. MTH Commands Used for Changing Bootstrap Parameters

Command	Short Form	Description
Slot	S	Displays the default slot number is. Most commands act upon the board in the default slot.
Slot#	S#	Changes the default slot to the number entered. (For example, entering S3 <CR> sets the default slot number to 3.)
InitbP	IP	Initializes the bootstrap parameters for the board in the default slot. It clears all of the operator-supplied parameters. It asks you to verify the action before initializing the parameters.
DispP	DP	Displays the operator-supplied bootstrap parameters of the board in the default slot. After power-up or cold reset, the parameters are initialized. If no new parameters are entered during the initialization phase, the parameters are taken from preconfigured sources (a BPS file, the boot parameter string in non-volatile memory, or the bootstrap loader's run-time parameters). Because there are no default parameters in non-volatile memory on the boards, this command displays no values for the BPS unless you enter them. If no parameters are entered, the code uses the default values. After a warm reset, this command displays whatever was put in the Bootstrap Parameter String.
ModbP	MP	Asks you for new parameter names and new values. Be careful to distinguish between new parameter and new value. Whatever you enter at the new value prompt is assigned to the parameter that was most recently displayed. If you don't want to change its value, type <CR>. The subcommands are: # deletes the parameter . ends the entry process <CR> goes to the next item
Bphase	B	Executes the first valid entry in the Program Table, which is the bootstrap loader. (The Program Table is a list of pointers to modules in the firmware.)

❑❑❑

Bootstrap Scenarios 3

This chapter provides several examples of the three type of MSA bootstrapping: independent, quasi-independent, and dependent. This chapter explains the three methods and then describes the scenarios for each one.

Overview of Examples and Terms

Using the examples in this chapter, you can learn MSA bootstrapping concepts and understand how MSA bootstrapping works.

Try as many of the examples in this chapter as your hardware configuration can support. While the examples assume specific hardware configurations, you can modify them to use other boards and configurations: select the appropriate BPS file (see the list of directories and files on page 33 and Table 3-1 on page 34) and the appropriate target files. The following examples are provided:

Example 1	Independent booting using an I/O server board.
Example 2	Quasi-independent booting of an I/O server board and a CPU board.
Example 3	Dependent booting an I/O server board and a CPU board.
Example 4	A combination of quasi-independent and dependent booting using an I/O server board, multiple CPU boards, and an Ethernet controller board.
Example 5	Quasi-independent booting using an I/O server board, one CPU board, and two hard drives on separate SCSI buses.

See also: Multibus II standard definition files, *Installation and Startup*

Bootstrap Methods

As stated in Chapter 1, the second stage bootstrap process can follow one of three methods: independent, quasi-independent, and dependent. These methods are defined as follows:

Independent The boards loads its own bootstrap software from a locally-attached device, such as a disk. Boards that boot independent can become servers to assist other boards in booting dependently.

A board without a locally attached device must boot across the PSB, either quasi-independently or dependently, and must broadcast its need for boot service.

Quasi-independent The CPU board boots across the PSB using a boot server running on a different host. In a Multibus II system, either the SBCP5090, P5120ISE, 386/258 or the 486/166SE boards can act as the boot server, supporting quasi-independent booting with a block-level protocol. PCI is the peripheral controller interface used on the I/O Server and CPU boards. PCI is built on top of the Multibus II Transport Protocol; it is a block-level protocol for message passing.

Dependent The CPU board relies on the help of a boot server from a board that booted independently or quasi-independently. When a boot server responds, the host uses boot server protocol (a high-level, file protocol) to bootstrap.

Host Configurations

The boot scenario examples in this chapter use the following terms to describe host configurations:

Diskfull describes a host that, after booting, executes from a local (non-network) system device.

Diskless describes a host that, after booting, executes from a remote (network) system device.

System Configuration

iRMX® Configuration Files

MSA boot systems require three files that reflect the desired system configuration:

BPS File	Contains the parameters needed to bootstrap Multibus II systems.
:sd:net/data	Defines the network names of the processor boards. A network can be entirely inside a single Multibus II system or connected to other systems. This file is required only if a network is needed. Networking is required if a board is booted dependently and has a remote system device.
:config:terminals	Associates terminal names with processor boards on the basis of the network name. (The network name is defined in the *:sd:net/data* file.)

See also: Bootstrap Parameters, Chapter 2;
:sd:net/data file, *Network User's Guide and Reference*;
:config:terminals file, *System Configuration and Administration*

Selecting Terminal Device Names

Two sample versions of *:config:terminals* are contained in the *:config:default* directory. The one you select depends on what device you use for the system console. If your system uses a 279A graphics module, select the *terminals.279* file. If your system uses a serial terminal, select the *terminals.arc* file. For example, to select the *terminals.arc* file, enter:

```
copy :config:default/terminals.arc over :config:terminals <CR>
```

Use the **skim** command to examine both of these files. In general, use the physical device name `t279_x` is used to communicate with the `atcs279` server and use the physical device name `atcs_con_0` is used to communicate with the `atcs279` server or the `arc` server. Refer to the ICU help screens for a discussion of the ATCS driver configuration.

Based on this discussion, you can now select the appropriate *terminals* file for your system.

Generating Configuration Files and Submit Files

The ICU uses a definition file to make iRMX configuration files and a Submit file for a specific configuration of the operating system. The Submit file assembles and compiles the configuration files. It binds and builds them with the iRMX libraries to make a target file. The target file is also called a bootable image. During bootstrap loading, the second stage loader loads the bootable image and transfers control to it.

This chapter refers to files with a *.bck* extension. These files are Standard Definition files for the operating system. Definition files contain all the configuration information for the operating system, including information about hardware in your system, desired user jobs, configured device drivers, and desired parts of the iRMX Operating System.

See also: Multibus II standard definition files, *ICU User's Guide and Quick Reference*

Making New Boot Systems

Following is the directory structure for the target files and the BPS files. If you have a 386/258 board in slot 0 or slot 1, the BPS files in */msa32/386258* apply to your boot system. If you have a 486/166SE board in slot 0, the BPS files in */msa32/486133* apply to your boot system. It is assumed that the 486/166SE, SBCP5090, and P5120ISE boards use a CSM/002, not a CSM/001 (no *bps1** files are supplied for such a configuration). All boot files are contained in the */msa32/boot* directory. The file */msa32/readme.txt* contains a summary of the BPS parameters.

Directories and *Files*

msa32
- 38625
  - bps.258
  - bps.cpu
  - bps.d
  - bps.dep
  - bps.dkt
  - bps.dl
  - bps.qi
  - bps.rmx
  - bps1.dl
  - bps1.qi
  - bps1.rmx
  - bpsmdp.dkt
  - bpsmdp.qi
  - bpsmdp.rmx

msa32
- 486133
  - bps.133
  - bps.166
  - bps.cpu
  - bps.dep
  - bps.dkt
  - bps.dsq
  - bps.dl
  - bps.qi
  - bps.rmx

msa32
- p5090
  - bpsise.90
  - bpsise.120
  - bps.90
  - bps.cpu
  - bps.dep
  - bps.dkt
  - bps.dsq
  - bps.dl
  - bps.qi
  - bps.rmx

Table 3-1 shows the relationship between the various boot scenarios and the BPS files.

Table 3-1. Bootstrap Method and Related Files

Boot Method	CSM Configuration	Default Bootstrap Parameter String File
Independent	CSM/002 CSM/001	*bps.cpu* *bps.cpu*
Quasi-independent	CSM/002 CSM/001	*bps.rmx* *bps1.rmx*
Dependent	CSM/001 CSM/002	*bps.dep* *bps.dep*
Quasi-independent and Dependent	CSM/002 CSM/001	*bps.qi* *bps1.qi*
Dual SCSI Quasi-independent	CSM/002 CSM/001	*bps.dsq* *bps.dsq*

Do not modify the default BPS files listed in Table 3-1. Instead, copy the default file that most closely matches your requirements to another file name. Then edit the newly created file. You should change the default boot system defined in the */msa/config/bps* file only when you are confident that the modified BPS file correctly boots your system.

Typically, in more complex systems you will need entries from several BPS files. You should initially use the file that fits your application most closely and remove entries from other BPS files. Be very careful to ensure entries from other BPS files point to the correct I/O server boards and correct server instance.

Parameters for Other I/O Server Boards

The example boot scenarios in this chapter use a P5120ISE and 486/166SE as an I/O server board.

Configuration Files for a Networking System

Each host must have a name that is unique, not just within the system, but within the entire local area network. Before connecting an Ethernet transceiver cable to the Multibus II system, choose new names for each of the host boards in the system. A useful convention is to choose one name and append the slot ID, creating a different name for each host.

If the default iRMX and iRMX-NET configurations have been used so far, the software does not have to be reconfigured. Instead, modify the following three files:

:sd:net/data	Change the names of the file server and the diskless hosts to the new names chosen. The file server entry is typically the first entry and does not have a specified address. The diskless host entries are of type=PT0005H and are by default slotnn.
:config:terminals	Specify new names of diskless hosts. The names must match those listed in the *:sd:net/data* file.
BPS file	Specify the name of the file server as the system device (`rq_sd`) for each of the diskless hosts in the BPS file you are using. The name listed here must match that listed in the *:sd:net/data* file.

See also: *:sd:net/data* file, *Network Concepts and Network Programmer's Reference*;
:config:terminals file, *System Configuration and Administration*;
Bootstrap Parameters, Chapter 2

The file server boot file does not need to be reconfigured. The diskless host boot files do not have to be reconfigured unless they specify that iRMX-NET attaches the file server. The default is to have the EIOS attach the file server. If iRMX-NET attaches the file server, the name of the file server must be configured in the master UDF device (ND) parameter of the user definition file (UDF) screen and in the CDF device (CD) parameter of the client definition file (CDF) screen in the ICU.

Example 1: Independent Boot Method

You can use independent bootstrapping for any board with a local SCSI subsystem, provided you have an I/O server board, such as a SBC P5120ISE, SBCP5090, 486/166SE, or a 386/258 board. In this example, the SBC P5120ISE board uses the SCSI protocol to communicate with the local disk. This allows the iRMX Operating System to be booted independently on the P5120ISE board.

When the P5120ISE board is booted this way, it forms a single-board Multibus II system that contains SCSI I/O software (PCI) and iRMX-NET networking software. Figure 3-1 shows the independent boot model.

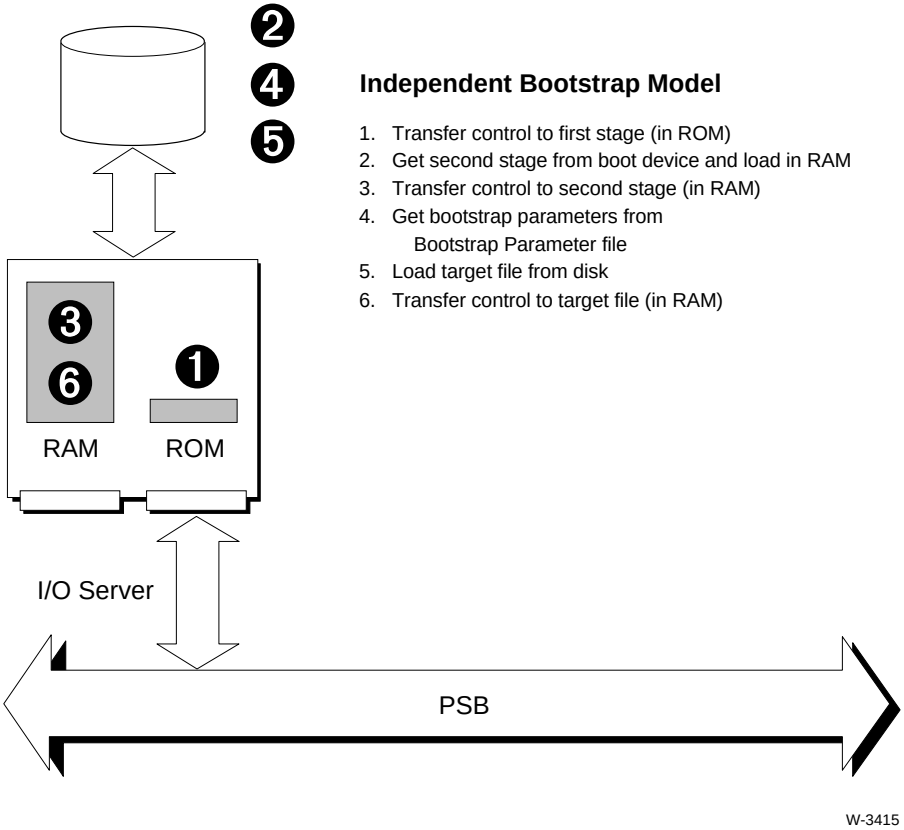

Figure 3-1. Independent Bootstrap Model

Hardware Configuration

Figure 3-2 shows the hardware configuration for this example. This P5120ISE-based single-board system provides the same functionality as a system using a 386/258 SCSI peripheral controller, a 486/150 CPU board, and a MIX486DX66 with a MIX560 Ethernet controller.

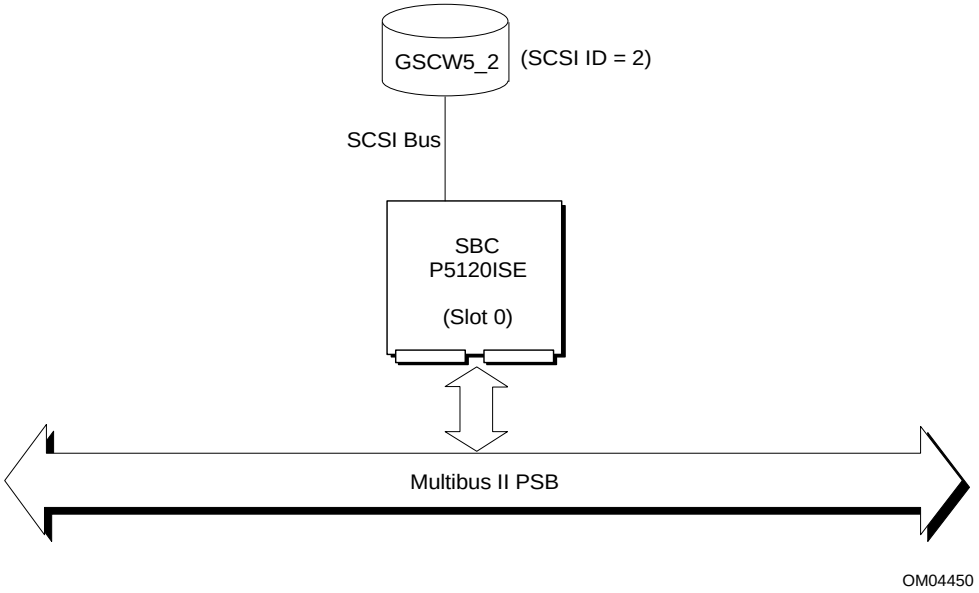

Figure 3-2. Hardware Configuration for Independent Boot Scenario

Software Configuration

This section describes the three types of files required for this example: the boot image file, the BPS file, and the *:config:terminals* file.

Boot Image Files

The file *p90scp.bck* is used to create the */msa32/boot/p90scp* target file. The *p90scp* target file causes the board to act as an independent host. It contains the human interface (HI), application loader (AL), and iRMX-NET networking software. The slot 0 board runs the PCI server and the *atcs/279/arc* server.

Before trying this boot scenario, verify the existence and status of the required target file.

1. Look for the required target files for this configuration using the command:

   ```
   find p90scp.out / <CR>
   ```

   If the file is present, the **find** command displays the pathname of the file and you can proceed to step 0. If the file is not found, create the target files, as described in the next step.

2. Create the target files using the following command:

   ```
   mksys p90scp <CR>
   ```

   The boot file produced by this command is */msa32/boot/p90scp*.

3. Use the **grep** and **skim** commands to display any error messages in the *p90scp.out* file:

   ```
   grep error p90scp.out >error <CR>  ;write errors to file
   skim error <CR>  ;display the lines with error messages
   delete error <CR>  ;delete the error message grep file
   ```

BPS.CPU File

The following is a listing of */msa32/p5090/bps.cpu*, the BPS file used for this boot scenario.

```
#
#   *-*-*   BPS.CPU   *-*-*
#
#   iRMX III MSA Bootstrap Parameter String configuration file for
#   use on Multibus II Microcomputers when an iSBC P5120ISE board
#   boots Independently from a diskette on its local SCSI I/O
#   subsystem using the default Multibus II boot file pathname
#   /msa/boot/rmx.
#
#   The iSBC P5120ISE may be in slot 0 or in slot 1.
#
#   Refer to /msa32/readme.txt for a description of the BPS and SPS
#   parameters.

[bl_host_id = 0]
       bl_target_file = /msa/boot/rmx;
       rq_pci_a = bnam:P5120ISE,bin:1,sin:0;
       rq_atcs_con = bnam:P5120ISE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:P5120CPU,bin:0,sid:31;
       rq_dlj = dev:GSCW5_2,fdvr:named;
       rq_sd = GSCW5_2

[bl_host_id = 1]
       bl_target_file = /msa/boot/rmx;
       rq_pci_a = bnam:P5120ISE,bin:1,sin:0;
       rq_atcs_con = bnam:P5120ISE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:P5120CPU,bin:0,sid:31;
       rq_dlj = dev:GSCW5_2,fdvr:named;
       rq_sd = GSCW5_2
```

The `bl_target_file` parameter has the value */msa32/boot/rmx*. This is the name of the default target file for an independently booting host. Using this name assures that your default target file is not overwritten when you create new target files with the ICU. There is no standard definition file that creates a target file with this name. Using this convention allows you to experiment with configurations but not risk losing the ability to easily load a functional boot system. The file */msa32/boot/p90scp* must be copied over */msa32/boot/rmx* to use the default */msa32/p5090/bps.cpu* file.

The :config:terminals File

The first device in *:config:terminals* is used by the P5120ISE board in this scenario. The *:config:terminals* file should be as follows:

```
2
atcs_con_0,,,ANY
t82530_0,,,ANY
.
.
.
```

Using `atcs_con_0` (rather than `t279_0`) will bring up a user whether you are using an iSBX– 279A or a serial terminal.

Testing the Boot Scenario

The */msa32/boot/rmx* target file is the default for this configuration. When booting the system in this configuration, you don't need to specify a `bl_target_file` or `bl_config_file` parameter from the MTH. To boot this configuration, use the steps described below.

1. Shutdown the system by typing:

   ```
   sh <CR>
   ```

2. Reset the system.

3. When you see the first characters on the screen, type:

   ```
   u
   ```

4. Invoke the boot phase by typing:

   ```
   Enter number:  ? 3
   ```

   As the system boots, you will see a screen display similar to the following:

```
INITIALIZATION PHASE SUMMARY                              Proceed to
  Slot    Product Code     Test Summary     Status     Boot Phase?
   0      P5120ISE        Passed           Active     Yes
System now entering Boot Phase...
MSA Bootstrap Loader

Leaving First Stage

Booting from SCW_2

Loading boot parameters from /msa/config/bps

Loading target file /msa32/boot/rmx
```

An HI version of the operating system boots on the slot 0 board.

First, the date and time are set. Then you will see an iRMX bannerhead for the slot 0 board. The host ID is printed in square brackets ([]) before the LOGON prompt. The bannerhead looks similar to this:

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

           iRMX* III.x.y  Operating System

 * iRMX is a registered trademark of RadiSys Corp

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

  [0]Logon:
```

5. Logon as *Super* and enter the default password (in lower case) as follows:

```
[0] Logon:  super <CR>
Password:  passme <CR>
```

Example 2: Quasi-Independent Method

In this boot scenario, the 486/166SE board boots the operating system independently from its local hard disk. The 486/150 board boots the operating system quasi-independently. The quasi-independent scenario is similar to a traditional boot process: the 486/150 board is the MSA boot master and treats the 486/166SE board like an I/O controller. The HI executes on the 486/150 board only.

During the boot phase, the 486/166SE board firmware uses the SCSI protocol to boot independently; it then hosts the PCI server. The 486/150 board firmware uses the PCI protocol to boot quasi-independently.

After the boot phase, the 486/166SE board provides the PCI server, which is part of its iRMX software. The 486/150 board uses a PCI driver, part of its iRMX software to communicate with the PCI Server on the 486/166SE board and to access data on the disk.

Figure 3-3 on page 44 illustrates the quasi-independent boot scenario.

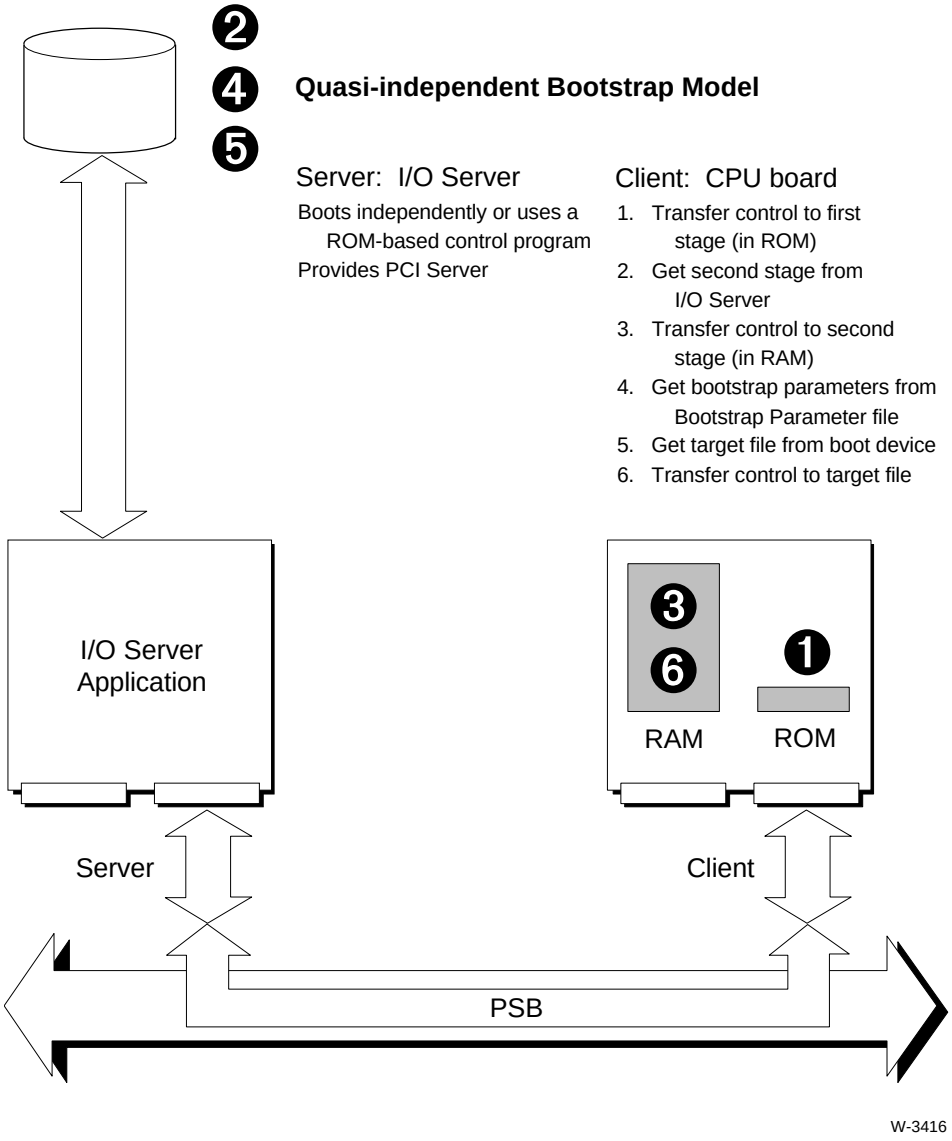

Figure 3-3. Quasi-Independent Bootstrap Model

Hardware Configuration

Figure 3-4 shows the hardware configuration for this example.

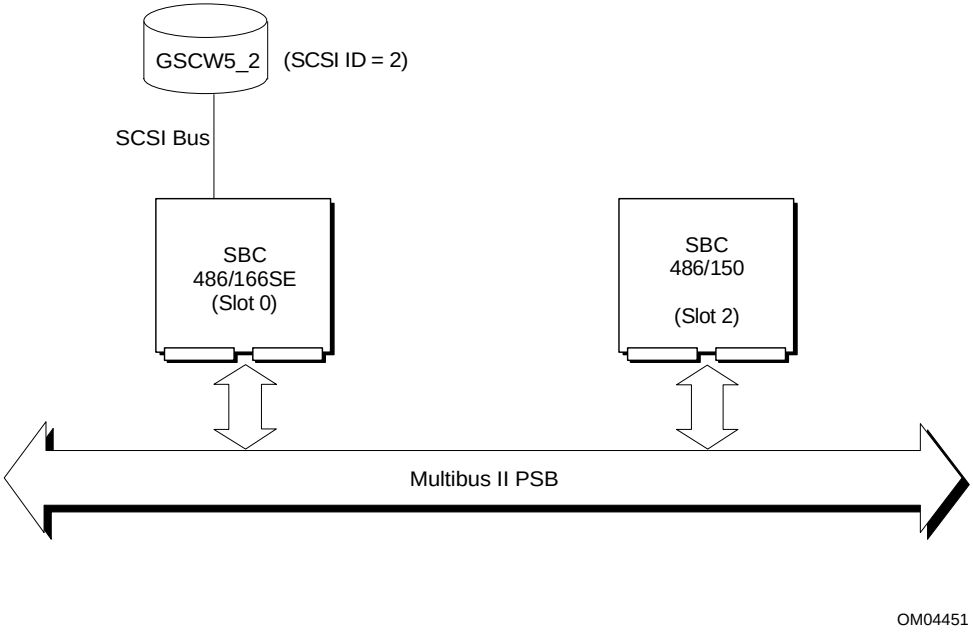

Figure 3-4. Hardware Configuration for Quasi-Independent Boot Scenario

Software Configuration

This section describes the three types of files required for this example: the *:config:terminals* file, the boot image files, and the BPS file.

The :config:terminals File

The first device in *:config:terminals* is used by the 486/150 board in this scenario. Because the 486/166SE board has no HI, it requires no *:config:terminals* entries. Refer to page 40 for a listing of a *:config:terminals* file that will work for this boot scenario.

Boot Image Files

Use the *433io.bck* and *486150.bck* files (or *486150net.bck* for a networking boot system) to create boot images for this example. These files contain the following:

433io.bck — This file configures the 486/166SE board as an I/O controller. It has no HI or AL. This version of the operating system is loaded from the hard disk, but does not have the system device (*:sd:*). The 486/166SE board runs the PCI server. This leaves the disk free for use by the 486/150 board. The target file created, */msa32/boot/433io*, must be copied over */msa32/boot/rmxio* to set up the default boot system.

486150.bck or *486150net.bck* — The target file created by this file includes the HI. The 486/150 board owns the disk that is attached to the 486/166SE board; the server on the 486/166SE board does not. The 486/150 board runs the PCI driver. The file */msa32/boot/486150* must be copied over */msa32/boot/rmx* to set up the default boot system.

Before trying this boot scenario, verify the existence and status of the required target files.

1. Look for the required target files for this configuration using the command:

   ```
   find 433io.out / <CR>
   ```

   If the file is present, the **find** command displays the pathname of the file and you can proceed to step 0. If the file is not found, create the target files, as described in the next step.

2. Create the target files using the following command:

   ```
   mksys 433io <CR>
   ```

   The Boot File produced by the commands above is */msa32/boot/433io*.

3. Use the **grep** and **skim** commands to display any error messages in the *433io.out* file:

   ```
   grep error 433io.out >error <CR>  ;write errors to file
   skim error <CR>  ;display the lines with error messages
   delete error <CR>  ;delete the error message grep file
   ```

Repeat these steps for the *486150.out* or *486150net.out* file, depending on which of the two you are using. If you use the **mksys** command to create the other target files for this scenario, the boot file is */msa32/boot/486150* or */msa32/boot/486150net*.

BPS.RMX File

The following is a listing of */msa32/486133/bps.rmx*, the BPS file used for this boot scenario.

```
#
#   *-*-*   BPS.RMX   *-*-*
#
#   iRMX III MSA Bootstrap Parameter String configuration file for
#   use on Multibus II Microcomputers when:
#      1) an iSBC 486/166SE board boots Independently from its local
#         SCSI I/O subsystem using the default I/O controller boot
#         file pathname /msa32/boot/rmxio and executes in an I/O
#         controller-only mode.
#      2) a single CPU board boots Quasi-independently from that
#         iSBC 486/166SE board using the default Multibus II boot
#         file pathname /msa32/boot/rmx.
#
#   Refer to /msa32/readme.txt for a description of the BPS and SPS
#   parameters.
#
#   The iSBC 486/166SE must be in slot 0; the QI-master CPU must be
#   in slot 2. BPS Parameters for an iSBC 486/166SE board booting
#   Independently.

[bl_host_id = 0]
       bl_target_file = /msa32/boot/rmxio;
       bl_qi_master = 2

#   BPS Parameters for a CPU board booting Quasi-independently in
#   slot 2.

[bl_host_id = 2;
       bl_method = quasi]
       bl_target_file = /msa32/boot/rmx;
       rq_pci_a = bnam:486/166SE,bin:1,sin:0;
       rq_atcs_con = bnam:486/166SE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:486/150,bin:0,sid:31;
       rq_dlj = dev:GSCW5_2,fdvr:named;
       rq_mip_00 = bnam:486/166SE,load:noload;
```

```
rq_rnet_c = bnam:486/166SE,bin:1;
rq_rnet_s = bnam:486/166SE,bin:1;
rq_sd = GSCW5_2
```

The */msa32/433io/bps.rmx* file contains the required boot parameters. In this example, override the default target file name (*/msa32/boot/rmx*) with a target file name */msa32/boot/433io* created earlier.

The `bl_qi_master` flag for the 486/166SE board is set to 2. This tells the 486/166SE board not to go to the disk to get the BPS file for the 486/150 board, which leaves the disk free for the 486/150 board to own the disk.

Description of the Boot Scenario

Both boards boot from the same disk. The 486/166SE board boots independently from its SCSI hard disk, but requires no further access of the hard disk. The 486/150 board hosts an HI and boots quasi-independently, which requires exclusive use (ownership) of the hard disk. The sequence of the quasi-independent boot of the 486/150 board is as follows:

1. The 486/150 board broadcasts its need for an MSA configuration server.
2. The 486/166SE board becomes a limited configuration server and tells the 486/150 board to become the QI master.
3. Then, the 486/150 board becomes the QI master, bootstraps itself, and takes control of the system device.

This scenario can be called diskfull because the operating system on the 486/150 board has full control of the disk.

Note the following items concerning this boot scenario:

- As in the independent boot example, default boot system file names can be used. The default name of the I/O controller-only version of the operating system for the 486/166SE board is */msa32/boot/rmxio*. The default name of the HI version of the operating system for the 486/150 board is */msa32/boot/rmx*. Using this convention allows you to experiment with configurations but not risk losing the ability to easily load a functional boot system. The default */msa32/486133/bps.rmx* file contains these parameters.
- The target file */msa32/boot/rmx* can include iRMX-NET for remote file access.

Testing the Boot Scenario

Use the MTH to modify several parameters so that you can boot the 486/150 board quasi-independently.

1. Shutdown the system by typing:

   ```
   sh <CR>
   ```

2. Reset the system.

3. When you see the first characters on the screen, type:

   ```
   u
   ```

4. Request the MTH by typing:

   ```
   2
   ```

5. At the MTH [0] prompt, enter the following commands to change the `bl_config_file` parameter values for the slot 0 and slot 2 boards:

   ```
   MTH [0] mp
   bl_target_file = /msa32/boot/433io <CR>
   new parameter
   bl_config_file = /msa32/486133/bps.rmx <CR>
   new parameter
   <CR>
   save changes ([y] or n)
   <CR>
   MTH [0] slot 2 <CR>
    Default Slot is 2.
   MTH [0] mp <CR>
    Modify Boot Parameters for slot   2:
   Store Bootstrap Parameter
   new parameter
   bl_target_file = /msa32/boot/486150 <CR>
   new parameter
   bl_config_file = /msa32/486133/bps.rmx <CR>
   new parameter
   <CR>
   save changes ([y] or n)
   <CR>
   MTH [0]
   ```

6. Tell the MTH to invoke the boot phase by typing:

   ```
   MTH [0] b <CR>
   ```

   An I/O controller version of the operating system boots on the 486/166SE board. An HI version of the operating system boots on the 486/150 board.

   First, the date and time are set. Then you will see an iRMX bannerhead for host 2, the 486/150 board, because it contains the HI. The host ID is printed in square brackets ([]) before the LOGON prompt. The bannerhead looks similar to this:

   ```
          *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-

                 iRMX* III.x.y  Operating System


             * iRMX is a registered trademark of RadiSys Corp


      *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*


   [2]Logon:
   ```

7. If the system booted successfully, you can now try out the next scenario.

8. If you want to establish this boot scenario as the default, type:

   ```
   COPY /MSA32/BOOT/433io OVER /MSA32/BOOT/RMXIO <CR>
   COPY /MSA32/BOOT/486150 OVER /MSA32/BOOT/RMX <CR>
   COPY /MSA32/486133/BPS.RMX OVER /MSA/CONFIG/BPS <CR>
   ```

9. To establish a networking boot system, type:

   ```
   COPY /MSA32/BOOT/486150NET OVER /MSA32/BOOT/RMX <CR>
   ```

   Be certain you have selected the appropriate *terminals* file.

Example 3: Dependent Method

Figure 3-5, on page 52, shows the dependent scenario. The 486/166SE board boots independently from the SCSI hard disk. The 486/150 board boots dependent with help from the MSA boot server on the 486/166SE board. This scenario is called diskless because the 486/150 board communicates indirectly with the disk. It uses the iRMX-NET remote file server and the Ethernet controller (on the 486/166SE board) to access files on the disk.

Because it uses networking, the dependent scenario allows user files to be shared between the 486/150 board and the 486/166SE board.

During the boot phase, the 486/166SE board firmware uses the SCSI protocol to boot independently; it then hosts the MSA boot server. The 486/150 board firmware, which is the client, uses the MSA boot server protocol to boot dependently.

After the boot phase, the 486/166SE board provides the iRMX-NET Remote File Server, which is part of its iRMX software. The 486/150 board uses the iRMX-NET client, part of its iRMX configuration, to access files on the disk attached to the 486/166SE board.

Note

A Multibus II system containing an Ethernet controller board is a small local area network, even without a cable connection to an outside net. Refer to page 35 for information on how to set up the required files for a networking system. Until you have set up these files, you should not connect an Ethernet cable to the system.

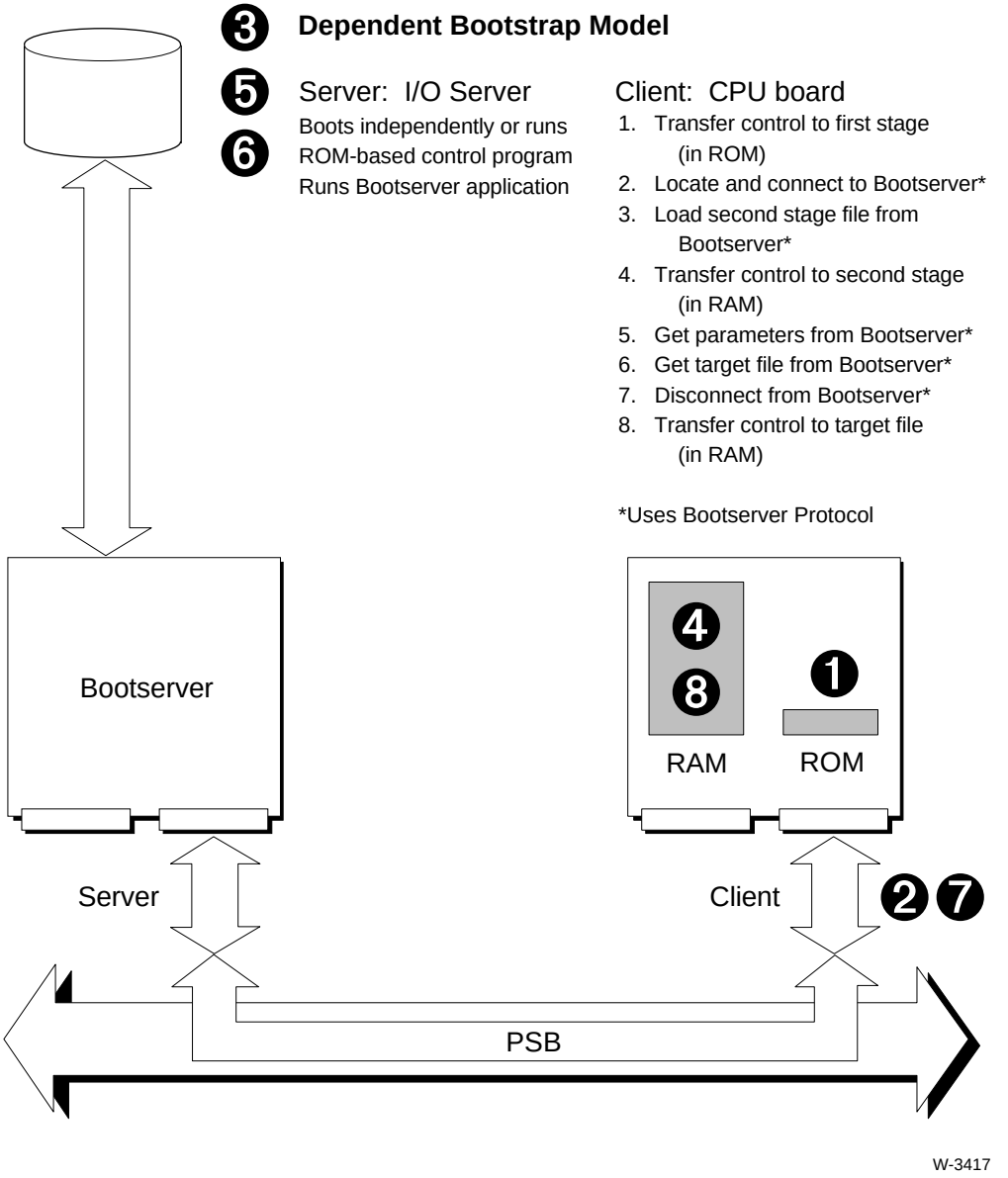

Figure 3-5. Dependent Bootstrap Model

Hardware Configuration

Figure 3-6 shows the hardware configuration for this example.

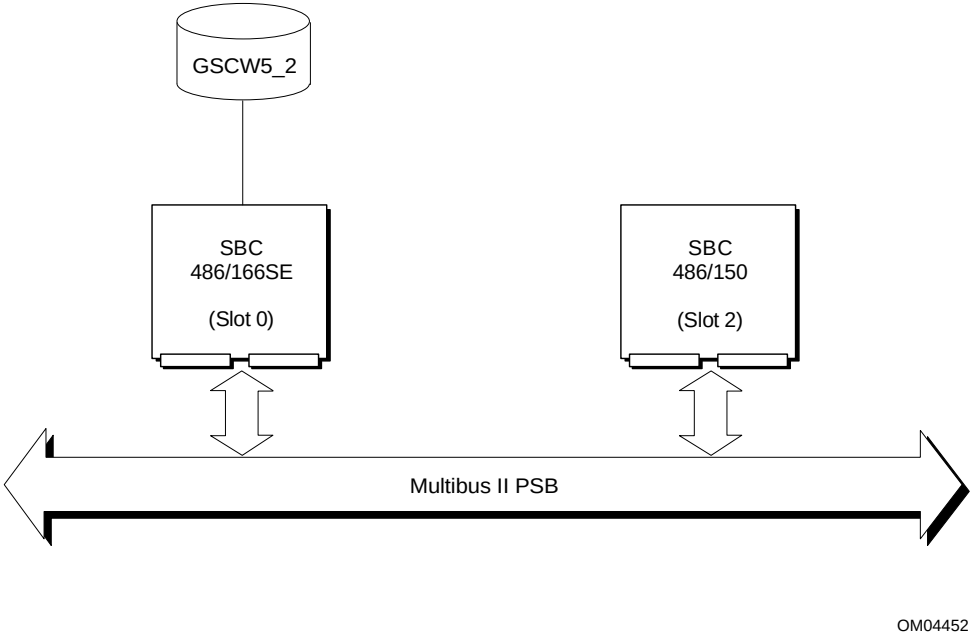

Figure 3-6. Hardware Configuration for Dependent Boot Scenario

Software Configuration

This section describes the four types of files required for this example: the boot image file, the BPS file, the *:config:terminals* file, and the */net/data* file.

Boot Image Files

Use the *433scp.bck* and *486150rsd.bck* files to create boot images for this example. The definition files contain the following:

433scp.bck	causes the slot 3 board to act as an independent host. It contains an HI, AL, PCI server, and iRMX-NET networking software. This version of the operating system is loaded from the hard disk, but because the BPS file tells the board to boot quasi-independent, it becomes the QI master. The slot 3 board has a hard disk as the system device (*:sd:*).
486150rsd.bck	Contains the iRMX-NET client and an HI. It is configured to use a remote device as its system disk.

The MSA boot server is in software on the 486/166SE board and the boot client is in firmware on the 486/150 board.

Before trying this boot scenario, verify the existence and status of the required target files.

1. Look for the required target files for this configuration using the command:

   ```
   find 433scp.out / <CR>
   ```

   If the file is present, the **find** command displays the pathname of the file and you can proceed to step 3. If the file is not found, create the target files, as described in the next step.

2. Create the target files using the following command:

   ```
   mksys 433scp <CR>
   ```

   The Boot File produced by this command is */msa32/boot/433scp.*

3. Use the **grep** and **skim** commands to display any error messages in the *433scp.out* file:

   ```
   grep error 433scp.out >error <CR>  ;write errors to file
   skim error <CR>  ;display the lines with error messages
   delete error <CR>  ;delete the error message grep file
   ```

Repeat these steps for the *486150rsd.out* file. If you use the **mksys** command to create the other target files for this scenario, the boot file is */msa32/boot/486150rsd.*

BPS.DEP File

The following is a listing of */msa32/486133/bps.dep*, the BPS file used for this boot scenario. This configuration file can also be used if you are using a CSM/001 in slot 0 and a 486/166SE board in slot 1.

```
#
#   *-*-*   BPS.DEP   *-*-*
#
#   iRMX III MSA Bootstrap Parameter String configuration file for
#   use on Multibus II Microcomputers when:
#      1) an iSBC 486/166SE board boots Independently from its local
#         SCSI I/O subsystem and hosts the MSA Bootserver.
#      2) From 0 to 5 CPU boards boot Dependently from the
#         iSBC 486/166SE which hosts the MSA Bootserver.
#
#   Refer to /msa32/readme.txt for a description of the BPS and SPS
#   parameters.

#   The iSBC 486/166SE may be in slot 0 or in slot 1.
#   BPS Parameters for the iSBC 486/166SE board booting
#   Independently.

[bl_host_id = 0]
       bl_target_file = /msa32/boot/433scp;
       rq_pci_a = bnam:486/166SE,bin:1,sin:0;
       rq_atcs_con = bnam:486/166SE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:486/150,bin:0,sid:31;
       rq_dlj = dev:GSCW5_2,fdvr:named;
       rq_sd = GSCW5_2

[bl_host_id = 1]
       bl_target_file = /msa32/boot/433scp;
       rq_pci_a = bnam:486/166SE,bin:1,sin:0;
       rq_atcs_con = bnam:486/166SE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:486/150,bin:0,sid:31;
       rq_dlj = dev:GSCW5_2,fdvr:named;
       rq_sd = GSCW5_2
```

```
#   BPS Parameters for CPU boards booting Dependently.

[bl_host_id = 2;
       bl_second_stage = /msa32/stage2.rmx;
       bl_method = dependent]
       bl_target_file = /msa32/boot/486150rsd;
       rq_atcs_con = bnam:486/166SE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:486/150,bin:0,sid:31;
       rq_mip_00 = bnam:486/166SE,load:noload;
       rq_rnet_c = bnam:486/166SE,bin:1;
       rq_rnet_s = bnam:486/166SE,bin:1;
       rq_sd = FSERVER

[bl_host_id = 3;
       bl_second_stage = /msa32/stage2.rmx;
       bl_method = dependent]
       bl_target_file = /msa32/boot/486150rsd;
       rq_atcs_con = bnam:486/166SE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:486/150,bin:0,sid:31;
       rq_mip_00 = bnam:486/166SE,load:noload;
       rq_rnet_c = bnam:486/166SE,bin:1;
       rq_rnet_s = bnam:486/166SE,bin:1;
       rq_sd = FSERVER
```

```
[bl_host_id = 4;
       bl_second_stage = /msa32/stage2.rmx;
       bl_method = dependent]
       bl_target_file = /msa32/boot/486150rsd;
       rq_atcs_con = bnam:486/166SE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:486/150,bin:0,sid:31;
       rq_mip_00 = bnam:486/166SE,load:noload;
       rq_rnet_c = bnam:486/166SE,bin:1;
       rq_rnet_s = bnam:486/166SE,bin:1;
       rq_sd = FSERVER

[bl_host_id = 5;
       bl_second_stage = /msa32/stage2.rmx;
       bl_method = dependent]
       bl_target_file = /msa32/boot/486150rsd;
       rq_atcs_con = bnam:486/166SE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:486/150,bin:0,sid:31;
       rq_mip_00 = bnam:486/166SE,load:noload;
       rq_rnet_c = bnam:486/166SE,bin:1;
       rq_rnet_s = bnam:486/166SE,bin:1;
       rq_sd = FSERVER

[bl_host_id = 6;
       bl_second_stage = /msa32/stage2.rmx;
       bl_method = dependent]
       bl_target_file = /msa32/boot/486150rsd;
       rq_atcs_con = bnam:486/166SE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:486/150,bin:0,sid:31;
       rq_mip_00 = bnam:486/166SE,load:noload;
       rq_rnet_c = bnam:486/166SE,bin:1;
       rq_rnet_s = bnam:486/166SE,bin:1;
       rq_sd = FSERVER
```

The :config:terminals File

The first device in *:config:terminals* is used by the 486/166SE board in this scenario. Refer to page 40 for a listing of a *:config:terminals* file that will work for this boot scenario.

The /net/data File

The 486/166SE board needs a correct *:sd:net/data* file to provide remote system device services to other hosts in the system. This section tells how to modify the *:sd:net/data* file so that the 486/150 board has a remote system device.

See also: Loading Objects from the :sd:net/data File, *Network User's Guide and Reference*

1. Make sure your terminal type is set up for AEDIT on your Multibus II system. To enable the line editor of the iRMX command line interpreter (CLI), type:

   ```
   set terminal=<your terminal type> <CR>
   ```

   See also: *AEDIT Text Editor User's Guide for iRMX Systems* for more information on using AEDIT

2. To get the address of the Ethernet board in the system, type:

   ```
   getaddr <CR>
   ```

3. Write down the address.

4. Prepare to modify the *:sd:net/data* file by copying a sample file to *:sd:net/data*, as follows:

   ```
   af /net <CR>
   copy data.ex to data <CR>
   aedit data <CR>
   ```

5. The file contains extra lines, which are examples. Delete all of them except the following:

   ```
   local_name1/nfs:    TYPE=rmx:  ADDRESS=;
   slot2:           TYPE=PT0005:  ADDRESS=ssss############02;
   slot3:           TYPE=PT0005:  ADDRESS=ssss############03;
   slot4:           TYPE=PT0005:  ADDRESS=ssss############04;
   slot5:           TYPE=PT0005:  ADDRESS=ssss############05;
   slot6:           TYPE=PT0005:  ADDRESS=ssss############06;
   ```

6. Replace `local_name` with the actual name of the server as specified by `rq_sd` in the BPS file. Substitute `ssss` with the subnet ID that applies. Substitute `############` with the Ethernet (MAC) address. Delete any comments that follow the semicolons.

   This example substitutes the Ethernet address 00AA00912345 for the number symbols and uses subnet ID 0001 (the default subnet ID for the Ethernet in all iNA 960 jobs):

```
FSERVER/nfs:    TYPE=rmx;     ADDRESS=;
slot2           TYPE=PT005    ADDRESS=000100AA0091234502;
slot3           TYPE=PT005    ADDRESS=000100AA0091234503;
slot4           TYPE=PT005    ADDRESS=000100AA0091234504;
slot5           TYPE=PT005    ADDRESS=000100AA0091234505;
slot6           TYPE=PT005    ADDRESS=000100AA0091234506;
```

   By using the default names, you avoid having to make additional changes, but you cannot use these files to access systems outside of the Multibus II chassis.

7. Quit the AEDIT editor while saving the file, by using these commands:

```
<ESC>
q
e
```

   This writes a new version of the */net/data* file to the disk.

Description of the Dependent Boot Scenario

In this scenario, the 486/166SE board boots independently from the SCSI device attached to it and hosts the MSA boot server. When the 486/150 board broadcasts its need for bootstrap, the boot server on the 486/166SE board reads the second stage bootstrap loader from the disk and sends it to the 486/150 board. The second stage begins running on the 486/150 board and asks the MSA boot server to send it the target file that contains the operating system. The second stage then loads the target file into the memory of the 486/150 board and transfers control to it.

The 486/166SE board must finish bootstrapping before it can provide the MSA boot server for the 486/150 board.

The client sets the `rq_sd` parameter to FSERVER, the name of the iRMX host used as a file server by the diskless host. (The name, FSERVER, is defined in the *:sd:net/data* file.)

Testing the Boot Scenario

1. Ensure the *:config:terminals* file describes your terminals with the correct network names for each host CPU board. The default version of *:config:terminals* contains the names, slot2, slot3, slot4, slot5, and slot6 to demonstrate these MSA boot scenarios.

   The network names are defined in the *:sd:net/data* file, and the terminal names are iRMX DUIB (device unit interface block) names defined in definition files.

   To view the *:config:terminals* file, type:

   ```
   skim :config:terminals <CR>
   ```

2. To shutdown the system, type:

   ```
   sh <CR>
   ```

3. Reset the system.

4. When you see the first characters on the screen, type:

   ```
   u
   ```

5. Request the MTH by typing:

   ```
   2
   ```

6. At the MTH [0] prompt, enter the following commands to change the `bl_config_file` parameter value and to enable booting:

```
MTH [0] mp
new parameter
bl_config_file=/msa32/486133/bps.dep <CR>
new parameter
<CR>
save changes ([y] or n)
<CR>
MTH [0] b <CR>
```

First you will see an iRMX bannerhead for the 486/166SE board, because it is the first board to boot. Then, an iRMX bannerhead for the 486/150 board covers most of the first screen. A bannerhead looks like this:

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

           iRMX* III.x.y  Operating System

 * iRMX is a registered trademark of RadiSys Corp

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

  [0]Logon:
```

See also: ARC Server in *System Configuration and Administration* for information on switching between hosts using a character-based terminal;
Appendix D in *Command Reference* for information on switching between hosts using an iSBX 279A

Example 4: Quasi-Independent and Dependent Booting

This scenario is a combination of the quasi-independent scenario and the dependent scenario described in previous sections. In this example, the P5120ISE board in slot 2 boots quasi-independently and the remaining CPU boards boot dependently.

The slot 0 board and the slot 2 board boot off the same disk. The slot 0 board boots independently from its SCSI hard disk but operates as an I/O server and Ethernet controller and requires no further access of the hard disk. The slot 2 board boots quasi-independently. The slot 2 board offers an HI which requires exclusive use (ownership) of the hard disk. The P5120CPU boards boot dependently from the MSA boot server on the slot 2 board.

During the boot phase, the slot 0 board firmware uses the SCSI protocol to boot independently; it then hosts the PCI server. The firmware on the slot 2 board uses the PCI protocol to boot quasi-independently. It then hosts the MSA boot server. The P5120CPU boards use the MSA boot server protocol to boot dependently.

After the boot phase, the slot 0 board provides the PCI server, which is part of its iRMX software. The slot 2 board uses a PCI driver, part of its iRMX software, to communicate with the slot 0 board and to access data on the disk. The slot 2 board also provides the iRMX-NET remote file server. The P5120CPU boards use the iRMX-NET client, part of their iRMX configuration, to access files on the disk owned by the slot 2 board.

Hardware Configuration

Figure 3-7 shows the hardware configuration for this example.

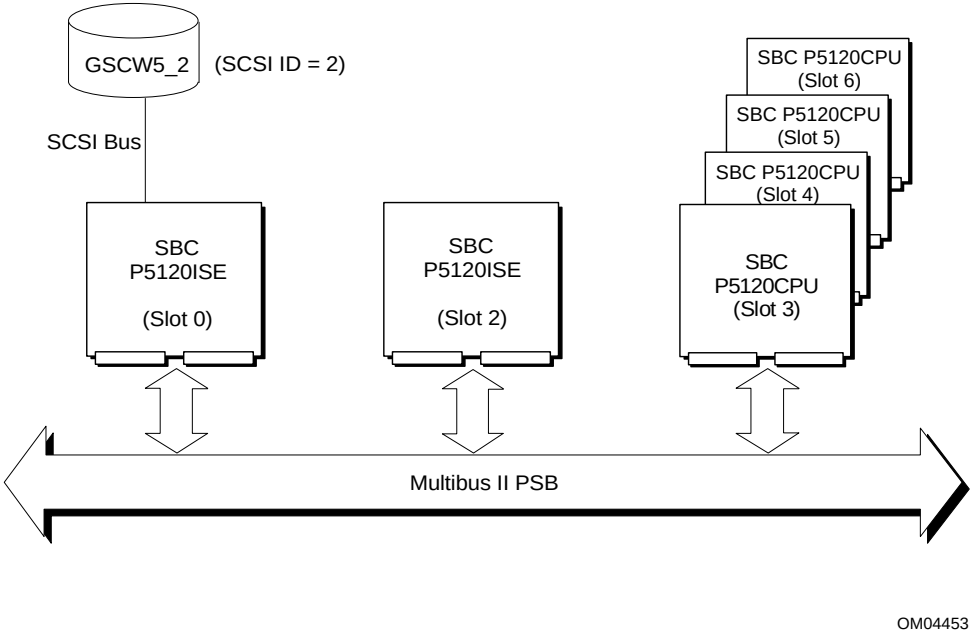

Figure 3-7. Hardware Configuration for Quasi-Independent and Dependent Boot Scenario

Software Configuration

This section describes the four types of files required for this example: the boot image files, the BPS file, the *:config:terminals* file, and the */net/data* file.

Boot Image Files

Use the *p90io.bck*, *p90net.bck*, and *p90rsd.bck* files to create boot systems for this example. These files contain the following:

p90io.bck	The configuration causes the slot 0 board to act like an I/O and Ethernet controller. It has no HI or AL. This version of the operating system is loaded from the hard disk, but does not have the system device (*:sd:*). This leaves the disk free for use by the slot 2 board. The slot 0 board runs the PCI server, the *atcs/279/arc* server, and iTP4.
p90cp.bck	This file includes the HI. The slot 2 board runs diskfull. The slot 2 board owns the disk that is attached to the slot 0 board; the server on the slot 0 board does not. The *p90cp.bck* also includes the iRMX-NET file server which is required to provide remote file access to the P5120CPU boards in the system.
p90rsd.bck	This file includes the HI and the iRMX-NET client. It is configured to use a remote device as the system disk. This board runs diskless.

Before trying this boot scenario, verify the existence and status of the required target files.

1. Look for the required target files for this configuration using the command:

   ```
   find p90io.out / <CR>
   ```

   If the file is present, the **find** command displays the pathname of the file and you can proceed to step 0. If the file is not found, create the target files, as described in the next step.

2. Create the target files using the following command:

   ```
   mksys p90io <CR>
   ```

   The boot file produced by this command is */msa32/boot/p90io*.

3. Use the **grep** and **skim** commands to display any error messages in the *p90io.out* file:

   ```
   grep error p90io.out >error <CR>  ;write errors to file
   skim error <CR>  ;display the lines with error messages
   delete error <CR>  ;delete the error message grep file
   ```

Repeat these steps for the *p90net.out,* and *p90rsd.out* files. If you use the **mksys** command to create the other target files for this scenario, the boot files are */msa32/boot/p90net* and */msa32/boot/p90rsd*, respectively.

BPS.QI File

The following is a listing of */msa32/p5090/bps.qi*, the BPS file used for this boot scenario.

```
#
#   *-*-*   BPS.QI   *-*-*
#
#   iRMX III MSA Bootstrap Parameter String configuration file for
#   use on Multibus II Microcomputers when:
#      1) An iSBC P5120ISE board boots Independently from its local
#         SCSI I/O subsystem and executes in an I/O and Ethernet
#         controller-only mode.
#      2) A single CPU board boots Quasi-independently from that
#         iSBC P5120ISE board and hosts the MSA Bootserver.
#      3) From 0 to 4 P5120CPU boards boot Dependently from the
#         CPU which hosts the MSA Bootserver.
#
#   Refer to /msa32/readme.txt for a description of the BPS and SPS
#   parameters.
#
#   The iSBC P5120ISE must be in slot 0; the QI-master CPU must be
#   in slot 2. BPS Parameters for an iSBC P5120ISE board booting
#   Independently.

[bl_host_id = 0]
       bl_target_file = /msa32/boot/p90io;
       bl_qi_master = 2

#   BPS Parameters for a CPU board booting Quasi-independently in
#   slot 2.

[bl_host_id = 2;
       bl_method = quasi]
       bl_target_file = /msa32/boot/p90net;
       rq_pci_a = bnam:P5120ISE,bin:1,sin:0;
       rq_atcs_con = bnam:P5120ISE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:P5120CPU,bin:0,sid:31;
       rq_dlj = dev:GSCW5_2,fdvr:named;
       rq_mip_00 = bnam:P5120ISE,load:noload;
```

```
       rq_rnet_c = bnam:P5120ISE,bin:1;
       rq_rnet_s = bnam:P5120ISE,bin:1;
       rq_sd = GSCW5_2

#   BPS Parameters for CPU boards booting Dependently in slots
#   3, 4, 5 and 6.

[bl_host_id = 3;
       bl_second_stage = /msa32/stage2.rmx;
       bl_method = dependent]
       bl_target_file = /msa32/boot/p90rsd;
       rq_atcs_con = bnam:P5120ISE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:P5120CPU,bin:0,sid:31;
       rq_mip_00 = bnam:P5120ISE,load:noload;
       rq_rnet_c = bnam:P5120ISE,bin:1;
       rq_rnet_s = bnam:P5120ISE,bin:1;
       rq_sd = FSERVER

[bl_host_id = 4;
       bl_second_stage = /msa32/stage2.rmx;
       bl_method = dependent]
       bl_target_file = /msa32/boot/p90rsd;
       rq_atcs_con = bnam:P5120ISE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:P5120CPU,bin:0,sid:31;
       rq_mip_00 = bnam:P5120ISE,load:noload;
       rq_rnet_c = bnam:P5120ISE,bin:1;
       rq_rnet_s = bnam:P5120ISE,bin:1;
       rq_sd = FSERVER

[bl_host_id = 5;
       bl_second_stage = /msa32/stage2.rmx;
       bl_method = dependent]
       bl_target_file = /msa32/boot/p90rsd;
       rq_atcs_con = bnam:P5120ISE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
```

```
       rq_atcs_d = bnam:P5120CPU,bin:0,sid:31;
       rq_mip_00 = bnam:P5120ISE,load:noload;
       rq_rnet_c = bnam:P5120ISE,bin:1;
       rq_rnet_s = bnam:P5120ISE,bin:1;
       rq_sd = FSERVER

[bl_host_id = 6;
       bl_second_stage = /msa32/stage2.rmx;
       bl_method = dependent]
       bl_target_file = /msa32/boot/p90rsd;
       rq_atcs_con = bnam:P5120ISE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:P5120CPU,bin:0,sid:31;
       rq_mip_00 = bnam:P5120ISE,load:noload;
       rq_rnet_c = bnam:P5120ISE,bin:1;
       rq_rnet_s = bnam:P5120ISE,bin:1;
       rq_sd = FSERVER
```

The :config:terminals File

The first device in *:config:terminals* is used by the slot 2 board in this scenario. The slot 0 board does not host an HI and, therefore, does not require a *:config:terminals* file. Refer to page 40 for a listing of a *:config:terminals* file that will work for this boot scenario.

The /net/data File

The slot 2 board needs a correct *:sd:net/data* file to provide remote system device services to other hosts in the system. This section tells how to modify the *:sd:net/data* file.

See also: Loading Objects from the :sd:net/data File, *Network User's Guide and Reference*

1. Make sure your terminal type is set up for AEDIT on your Multibus II system. To enable the line editor of the iRMX command line interpreter (CLI), type:

   ```
   set terminal=<your terminal type> <CR>
   ```

   See also: *AEDIT Text Editor User's Guide for iRMX Systems* for more information on using AEDIT

2. To get the address of the Ethernet board (the P5120ISE board in slot 0), type:

   ```
   getaddr <CR>
   ```

3. Write down the address.

4. Prepare to modify the *:sd:net/data* file by copying a sample file to *:sd:net/data*, as follows:

   ```
   af /net <CR>
   copy data.ex to data <CR>
   aedit data <CR>
   ```

5. The file contains extra lines, which are examples. Delete all of them except the following:

   ```
   local_name1/nfs:    TYPE=rmx:  ADDRESS=;
   slot2:           TYPE=PT0005:  ADDRESS=ssss############02;
   slot3:           TYPE=PT0005:  ADDRESS=ssss############03;
   slot4:           TYPE=PT0005:  ADDRESS=ssss############04;
   slot5:           TYPE=PT0005:  ADDRESS=ssss############05;
   slot6:           TYPE=PT0005:  ADDRESS=ssss############06;
   ```

6. Replace `local_name` with the actual name of the server as specified by `rq_sd` in the BPS file. Substitute `ssss` with the subnet ID that applies Substitute `############` with the Ethernet (MAC) address. Delete any comments that follow the semicolons.

   This example substitutes the Ethernet address 00AA00912345 for the number symbols and uses subnet ID 0001 (the default subnet ID for the Ethernet in all iNA 960 jobs):

```
FSERVER/nfs:    TYPE=rmx;     ADDRESS=;
slot2           TYPE=PT005    ADDRESS=000100AA0091234502;
slot3           TYPE=PT005    ADDRESS=000100AA0091234503;
slot4           TYPE=PT005    ADDRESS=000100AA0091234504;
slot5           TYPE=PT005    ADDRESS=000010AA0091234505;
slot6           TYPE=PT005    ADDRESS=000100AA0091234506;
```

   By using the default names, you avoid having to make additional changes, but you cannot use these files to access systems outside of the Multibus II chassis.

7. Quit the AEDIT editor while saving the file, by using these commands:

```
<ESC>
q
e
```

   This writes a new version of the */net/data* file to the disk.

Description of the Boot Scenario

In this scenario, the slot 0 board boots independently from its local hard disk. The slot 2 board boots quasi-independently. The remaining boards boot dependently.

1. The slot 2 board broadcasts its need for an MSA configuration server.
2. The slot 0 board becomes a limited configuration server and tells the slot 2 board that it should become the QI master.
3. Then, the slot 2 board becomes the QI master, bootstraps itself, takes control of the system device, and hosts the MSA boot server.
4. When the remaining boards broadcast their need for bootstrap, the boot server on the slot 2 board reads the second stage bootstrap loader from its disk and sends it to each of the client P5120CPU boards. The second stage begins running on each of the remaining boards and requests the MSA boot server to send it the appropriate target file. The second stage then loads the target file into the memory of the client boards and transfers control to them.

   The slot 2 board must finish bootstrapping before it can provide the MSA boot server for the remaining boards.

The client sets the `rq_sd` parameter to FSERVER, the name of the iRMX host used as a file server by the diskless host. (The name, FSERVER, is defined in the *:sd:net/data* file.)

The `bl_qi_master` flag for the slot 0 board is set to 2. This tells the slot 0 board not to go to the disk to get the BPS file for the slot 2 board, which leaves the disk free for the slot 2 board to own the disk. If the slot 2 board were in slot 3, you would set the `bl_qi_master` flag to 3.

Testing the Boot Scenario

1. Ensure that the *:config:terminals* file describes your terminals with the correct network names for each host CPU board. The default version of *:config:terminals* contains the names slot2, slot3, slot4, slot5, and slot6 to demonstrate these MSA boot scenarios.

   The network names are defined in the *:sd:net/data* file, and the terminal names are iRMX DUIB names defined in definition files.

   To view the *:config:terminals* file, type:

   ```
   skim :config:terminals <CR>
   ```

   This file contains the lines of the example *terminals* file you selected earlier.

2. Shutdown the system by typing:

   ```
   sh <CR>
   ```

3. Reset the system.

4. When you see the first characters on the screen, type:

   ```
   u
   ```

5. Request the MTH by typing:

   ```
   2
   ```

6. Set the `bl_config_file` parameter for the slot 0 board using the following commands:

   ```
   MTH [0] mp
   new parameter
   bl_config_file=/msa32/486133/bps.qi <CR>
   new parameter
   <CR>
   save changes ([y] or n)
   <CR>
   MTH [0]
   ```

7. Set the `bl_config_file` parameter for the slot 2 board using the following commands:

```
MTH [0] slot 2 <CR>
 Default slot is 2.
MTH [0] mp <CR>
 Modify Boot Parameters for slot   2:
Store Bootstrap Parameter
new parameter
bl_config_file=/msa32/486133/bps.qi <CR>
new parameter
<CR>
save changes ([y] or n)
<CR>
MTH [0]
```

8. Tell the MTH to invoke the boot phase by typing:

```
MTH [0] b <CR>
```

First you will see an iRMX bannerhead for host 2, the P5120ISE board in slot 2, because it is the first board to boot. The host ID is printed in square brackets ([]) before the LOGON prompt. Next, the iRMX bannerheads for the remaining P5120CPU boards cover most of the first screen. A bannerhead looks like this:

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

            iRMX* III.x.y  Operating System

* iRMX is a registered trademark of RadiSys Corp

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

[2]Logon:
```

See also: ARC Server in *System Configuration and Administration* for information on switching between hosts using a character-based terminal;
Appendix D in *Command Reference* for information on switching between hosts using an iSBX 279A

Example 5: Dual SCSI Bus Quasi-Independent Booting

This scenario is similar to the quasi-independent boot scenario described in Example 2, but with several differences in the hardware configuration. Here is an overview:

- The slot 0 board boots independent as an I/O controller using GSCW5_2.
- The slot 3 board boots quasi-independently using GSCW5_2. This board is diskfull.
- A second hard drive on a separate SCSI bus is available for access from both boards.

Hardware Configuration

Figure 3-8 shows the hardware configuration for this example. The main difference between this example and the previous examples is the addition of a second SCSI bus. This hardware configuration is also used in Example 6.

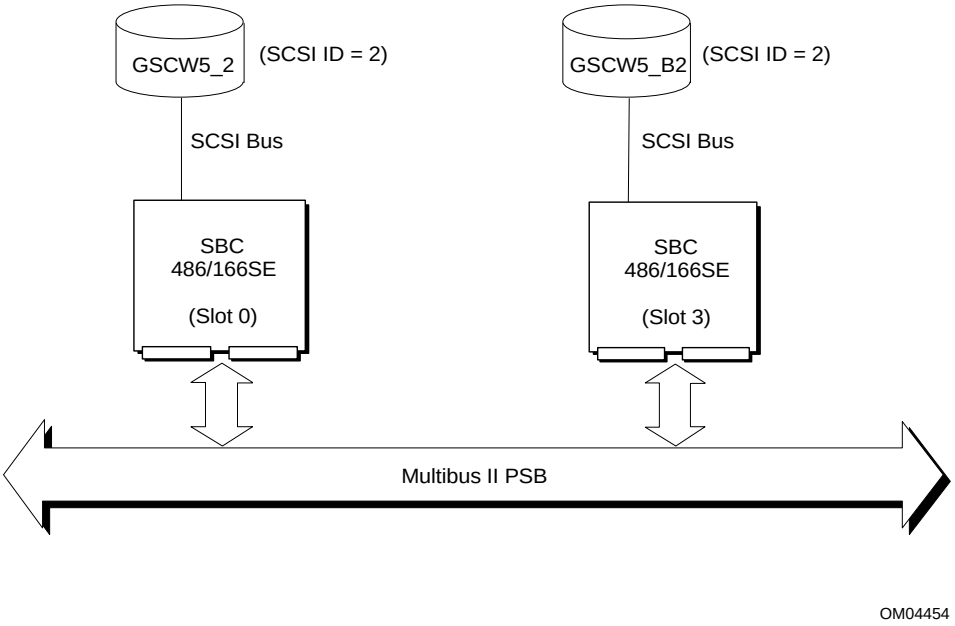

Figure 3-8. Hardware Configuration for Dual SCSI Bus Quasi-Independent Boot Scenario

Software Configuration

This section describes the three types of files required for this example: the boot image files, the BPS file, and the *:config:terminals* file.

Boot Image Files

This scenario uses the *433io.bck* and *433scp.bck* files to create boot images. These files contain the following:

433io.bck causes the slot 0 board to act as an I/O controller. It has no HI or AL. This version of the operating system is loaded from the hard disk, but does not have the system device (*:sd:*). This leaves the disk free for use by the QI master. The slot 0 board runs the PCI server and the *atcs/279/arc* server.

433scp.bck causes the slot 3 board to act as an independent host. It contains an HI, AL, PCI server, and iRMX-NET networking software. This version of the operating system is loaded from the hard disk, but because the BPS file tells the board to boot quasi-independent, it becomes the QI master. The slot 3 board has a hard disk as the system device (*:sd:*).

Before trying this boot scenario, verify the existence and status of the required target files, beginning with *433io.out*.

1. Look for the required target files for this configuration using the command:

   ```
   find 433io.out / <CR>
   ```

   If the file is present, the **find** command displays the pathname of the file and you can proceed to step 0. If the file is not found, create the target files, as described in the next step.

2. Create the target files using the following command:

   ```
   mksys 433io <CR>
   ```

   The Boot File produced by this command is */msa32/boot/433io*.

3. Use the **grep** and **skim** commands to display any error messages in the *433io.out* file:

   ```
   grep error 433io.out >error <CR>  ;write errors to file
   skim error <CR>  ;display the lines with error messages
   delete error <CR>  ;delete the error message grep file
   ```

Repeat these steps for the *433scp.out* file. If you use the **mksys** command to create the other target files for this scenario, the boot files are */msa32/boot/433scp*.

BPS.DSQ File

The following is a listing of */msa32/486133/bps.dsq*, the BPS file used for this boot scenario.

```
#                        *-*-*   BPS.DSQ   *-*-*
#
#  iRMX III MSA Bootstrap Parameter String configuration file for
#  use on dual SCSI bus Multibus II Microcomputers with the
#  following configuration.
#     1) An iSBC 486/166SE in slot 0 boots Independently from its
#        local SCSI I/O subsystem and executes in an I/O controller
        and ATCS server mode.
#     2) An iSBC 486/166SE in slot 3 boots Quasi-independently from
#        the SCSI I/O subsystem on the slot 0 board, contains a
#        Human Interface, and acts as a SCSI (PCI) controller and an
#        Ethernet controller.
#
#  Refer to /msa32/readme.txt for a description of the BPS and SPS
#  parameters.
#
#  BPS Parameters for an iSBC 486/166SE in slot 0 booting
#  Independently.
[bl_host_id = 0]
       bl_target_file = /msa32/boot/433io;
       bl_qi_master = 3

#  BPS Parameters for an iSBC 486/166SE in slot 3 booting Quasi-
#  independently from the first disk on the SCSI bus controlled by
#  the iSBC 486/166SE in slot 0.
#
[bl_host_id = 3;
       bl_method = quasi]
       bl_target_file = /msa32/boot/433scp;
       rq_pci_a = bnam:486/166SE,bin:1,sin:0;
       rq_pci_b = bnam:486/166SE,bin:2,sin:0;
       rq_atcs_con = bnam:486/166SE,bin:1;
       rq_atcs_a = bnam:186/410,bin:1;
       rq_atcs_b = bnam:186/450,bin:1;
       rq_atcs_c = bnam:MIX486DX66,bin:1;
       rq_atcs_d = bnam:486/150,bin:0,sid:31;
       rq_dlj = dev:GSCW5_2,fdvr:named;
       rq_sd = GSCW5_2
```

There are three parameters for host ID 3 that should be examined:

`bl_method` Before booting with *bps.dsq* file, you must specify a `bl_method` of `quasi` for board 3 from the MTH. Although it may seem redundant to specify the `bl_method` when the parameter is included in the BPS file, it is necessary. The reason is that an I/O controller board, by default, tries to boot across its own SCSI bus. Specifying the `bl_method` from the MTH insures that the slot 3 board boots quasi-independently, as required by this scenario.

`rq_pci_b` This parameter indicates a second instance of an I/O controller in the system (in this case, a second 486/166SE board). Having this parameter in the file allows access to a device with a `b` in its DUIB. For this configuration, the device is GSCW5_B2, the hard drive on the second SCSI bus.

The :config:terminals File

The first device in *:config:terminals* is used by the slot 3 board in this scenario. Refer to page 40 for a listing of a *:config:terminals* file that will work for this boot scenario.

Description of the Boot Scenario

In this boot scenario, the slot 0 board boots independently from its local hard disk. The slot 3 board boots quasi-independently:

1. The slot 3 board broadcasts its need for an MSA configuration server.
2. The slot 0 board becomes a limited configuration server and tells the slot 3 board that it should become the QI master.
3. Then, the slot 3 board becomes the QI master, bootstraps itself, and takes control of GSCW5_2, the system device, which is attached to the board in slot 0.

Testing the Boot Scenario

1. Shutdown the system by typing:

   ```
   sh <CR>
   ```

2. Reset the system.

3. When you see the first characters on the screen, type:

   ```
   u
   ```

4. Request the MTH by typing:

   ```
   2
   ```

5. Set the `bl_config_file` parameter value for the slot 0 board using the following commands:

   ```
   MTH [0] mp <CR>
   new parameter
   bl_config_file = /msa32/486133/bps.dsq <CR>
   new parameter
   <CR>
   save changes ([y] or n)
   <CR>
   MTH [0]
   ```

6. Set the `bl_config_file` and `bl_method` parameters for the slot 3 board using the following commands:

   ```
   MTH [0] slot 3 <CR>
    Default Slot is 3.
   MTH [0] mp <CR>
    Modify Boot Parameters for slot   3:
   Store Bootstrap Parameter
   new parameter
   bl_config_file = /msa32/486133/bps.dsq <CR>
   new parameter
   bl_method = quasi <CR>
   new parameter
   <CR>
   save changes ([y] or n)
   <CR>
   MTH [0]
   ```

7. Tell the MTH to invoke the boot phase by typing:

   ```
   MTH [0] b <CR>
   ```

   An I/O controller version of the operating system boots on the slot 0 board. An HI version of the operating system boots on the slot 3 board.

   First, the date and time are set. Then you will see an iRMX bannerhead for the slot 3 board because it contains the HI. The host ID is printed in square brackets ([]) before the LOGON prompt. The bannerhead looks similar to this:

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

             iRMX* III.x.y  Operating System

* iRMX is a registered trademark of RadiSys Corp

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

[3]Logon:
```

8. Logon as Super and enter the default password (in lower case) as follows:

   ```
   [3] Logon:  super <CR>
   Password:  passme <CR>
   ```

After booting the system using the *bps.dsq* file, prepare the hard disk attached to board 3. These steps assume a 700 Mbyte or larger hard disk.

See also: Determining disk parameters, "Step 4. Determining Disk Parameters," Chapter 4 of *Installation and Startup*

1. Attach the device using the following command:

   ```
   attachdevice gfcw5_b2 as :w: <CR>
   ```

2. Format the hard drive using the following command:

   ```
   format :w:rmxIII IL=1 files=64000 msaboot <CR>
   ```

   This command formats the drive with the logical name rmxIII at an interleave factor of 1 with a maximum number of 64,000 files and writes an MSA second stage loader to the disk. The format operation takes about 30 minutes to complete.

   See also: Second Stage Bootstrap Loader, page

3. Copy all files and subdirectories from the hard drive attached to the slot 0 board to the newly-formatted hard drive attached to the slot 3 board using the following command:

   ```
   copydir :sd: over :w: <CR>
   ```

   This operation takes about 30 minutes, but will vary depending on the number and size of files on the system device. After finishing the copy operation, you can access the hard disk attached to the slot 3 board. The generic DUIB of the device is GSCW5_B2.

□□□

Troubleshooting 4

SDM Recovery

When an error occurs during the first or second stage of booting, the System Debug Monitor (SDM monitor) is invoked. If the error occurs on the board that is master during booting, the board displays an error message on the screen. A slave board does not display a message. Instead, the board invokes the SDM monitor.

When the monitor is invoked after a bootstrap error, it displays this message:

```
Not a 286 Task State Segment SDM[n].
```

To exit the monitor during bootstrap you need to reset the system.

MSA Boot Error Messages

This section discusses various MSA boot error messages and suggests solutions for these errors.

Boot master error message: `Not a 286 Task State Segment SDM[n].`

Cause	Solution
The bootstrap target filename was not found.	Verify that the bl_target_file entry contains the correct pathname and that the target file exists.
The target file being downloaded is corrupted because the network interface adapter (NIA) board was not correctly downloaded.	Verify that the BNAM entry of the rq_mip_xx parameter matches the string returned by the ic -c agents command. Verify that the board instance (BIN) and server instance (SIN) are correct. Verify that the iNA 960 file being downloaded to the NIA board (FILE) is the correct one for this NIA board.

Boot master message: `Error Opening target file.`

Cause	Solution
The target file was not found.	Check that the pathname for the target file, either the default name or the name entered with the MP (Modify Parameter) command, is correct. Check that the file exists. Check that the file contains an iRMX target file.

iRMX-NET error during boot: `Comm Board not found`

Cause	Solution
The network interface adapter board being downloaded was not correctly specified in the BPS file's rq_mip_xx parameters.	Verity that the BNAM entry of the rq_mip_xx parameter matches the string returned by the ic -c agents command. Verify that xx, in rq_mip_xx refers correctly to the board instance. Verify that the iNA 960 file being downloaded to the NIA board is the correct one for this NIA board.

iRMX-NET error during boot: `No response from the Comm Board`

Cause	Solution
The network interface adapter board being downloaded was not correctly specified in the BPS file's rq_mip_xx parameters.	Verity that the BNAM entry of the rq_mip_xx parameter matches the string returned by the ic -c agents command. Verify that xx, in rq_mip_xx refers correctly to the board instance. Verify that the iNA 960 file being downloaded to the NIA board is the correct one for this NIA board.

Boot failure symptom: No logon session is created for a diskless CPU board.

Cause	Solution
The BPS file does not contain an entry for this CPU board.	Verify that the BPS file contains an entry beginning with the correct bl_host_id for this board.
The network name for the diskless CPU boards are not consistent among the BPS file, the */net/data* file, and the *:config:terminals* file (on the appropriate hard drives).	Verify that the network names for all CPU boards are consistent among these three files.
The rq_pci_a entry in the BPS file is incorrect for this CPU board.	Verify that the rq_pci_a bnam entry points to the board ID as displayed by the ic -c agents command, and that the board instance (BIN) and server instance (SIN) are correct.
A diskless target file was specified with an rq_sd pointing to a network name that could not be found by the diskless CPU board.	Verify that non-remote system device (RSD) target files are not specified if rq_sd points to a hard drive. Also, see the discussion above about coordinating network names among the BPS, */net/data*, and *:config:terminals* files.
The rq_atcs_con entry does not correctly point at the CPU board with the iSBX 279 module or the system terminal.	Verify that the board name (BNAM) parameter of the rq_atcs_con entry contains the exact characters returned by the ic -c agents command. Verify that the board instance (BIN) parameter is correct.

iRMX-NET error during boot: `Could not attach :sd:sys386 as system:`

Cause	Solution
The CPU board was not able to find the specified system device or the SYS386 directory on that device.	Verify that the specified system device exists. Verify that the specified system directory has an iRMX file system with the correct directory structure to serve as a system device.
The network interface adapter (NIA) board is not functioning properly.	Verity that the BNAM entry of the rq_mip_xx parameter matches the string returned by the ic -c agents command. Verify that xx, in rq_mip_xx refers correctly to the board instance. Verify that the iNA 960 file being downloaded to the NIA board is the correct one for this NIA board.

iRMX-NET error during boot: `0021H: iRMX error while loading iNA 960 961`

Cause	Solution
The specified iNA 960 file to be downloaded to the network interface adapter (NIA) board does not exist.	Verity that the BNAM entry of the rq_mip_xx parameter matches the string returned by the ic -c agents command. Verify that xx, in rq_mip_xx refers correctly to the board instance. Verify that the iNA 960 file being downloaded to the NIA board is the correct one for this NIA board.

Boot failure symptom: No logon session for the second diskfull CPU board

Cause	Solution
BPS file indicates wrong boot scenario.	Verify that there is only one boot master with a bl_method of quasi, and that the second diskfull CPU board has a bl_method of dependent with the proper bl_second_stage parameter.
BPS file indicates incorrect system device.	Verify that the rq_sd entry for this CPU board does not collide with another CPU board's rq_sd entry.
The :config:terminals entry for the second diskfull CPU board collides with a port/window already open by another board.	Verify that the *:config:terminals* file on the second diskfull CPU board's hard drive assumes that this board is the first entry in the file, and the entries do not collide with any entries in the *:config:terminals* file on the first disk-full CPU board's hard drive.

Boot failure symptom: No logon sessions on iSBC 186/410 ports

Cause	Solution
The iSBC 186/410 was not downloaded.	Verify that the :*config:dload.mb2* file is correct for your system. An example of this file is *:config:default/dload.mb2*.
The rq_atcs_b parameters in the BPS file are incorrect.	Verify that the BNAM and BIN entries in the BPS file do not collide with the entries for another board.

Causing a Warm Reset

A warm reset can be caused in any of these ways:

- Using the HI **ic** (read or write to interconnect space) command
- Using the **warmreset** command, which is an aliased Multibus II specific CLI command
- Using the front-panel keyswitch

These three methods are discussed here.

Using the HI ic Command

If the board you are resetting has a CSM/002, you can issue an **ic** command that instructs the CSM/002 to cause a warm reset. The examples below assume you want to cause a warm reset on a 486/133SE board in slot 0 with a CSM/002 installed. The commands are:

```
ic -c set 0 151 2 <CR>
```

or

```
icwrite 0 151 2 <CR>
```

Either of these commands writes a value of 2 to the CSM command register, thereby causing a warm reset. To use these commands, you must know the offset of the CSM command register. A simpler method is to use a form of the **ic** command that directly specifies a reset to desired board. For example, to cause a warm reset of the board in slot 0, type:

```
- ic -c reset 0 warm <CR>
```

You are prompted:

```
RESET COMMAND - Are you sure (Y/[N]) ?
```

Type:

```
y <CR>
```

Using the Aliased Warmreset Command

The file *:prog:alias.csd* contains an alias for the warmreset command. To cause a warm reset, type:

```
- warmreset <CR>
```

You are prompted:

```
RESET COMMAND - Are you sure (Y/[N]) ?
```

Type:

```
y <CR>
```

The default setting of the alias causes the warm reset to applied to the board in slot 0. `Warmreset` is an alias for the following command:

```
- ic -c reset 0 warm
```

You can change the alias to reset a different board.

Warm Reset from the Front Panel

In some Multibus II systems, turning the front panel keyswitch to the reset position causes a cold reset. It is possible to change the default reset type to a warm reset. In order to do this, you need to remove the cover on the system to get access to the P2 backplane for slot 0. On the P2 backplane there is a jumper which selects the type of reset. Because it is easier to cause a warm reset using the methods described above, it is not recommended that you change this jumper. If you want to change the jumper, refer to the hardware manuals that were shipped with your Multibus II system.

□□□

Index

C

D

E

F

H

I

M

O

P

Q

R

S

T

U

W