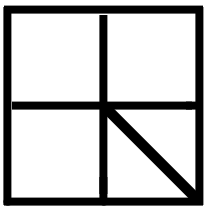
User Manual

Version 3.4B

PromICE™



Grammar Engine Inc.

921 Eastwind Drive Suite 122 Westerville Ohio 43081

(614) 899-7878 Voice
(614) 899-7888 Fax
Internet: www.gei.com

PromICE User Manual

Version 3.4B

PRINTED IN THE UNITED STATES OF AMERICA

WARRANTY

GEI provides a 30-day money back guarantee on its products. Within that period, if you are not fully satisfied within PromICE, it can be returned for a refund. The shipping and handling fees are non-refundable. All returned merchandise must be complete, in working order, and returned in the original packing with a GEI-supplied Return Material Authorization number.

GEI products are covered by a one year warranty. GEI warrants that the equipment is free of manufacturing defects (i.e. defects in material and workmanship under normal and proper use in their unmodified condition) for a period of one year from the date of delivery. GEI shall repair or replace any defective equipment during this period at its option. This warranty does not cover any damage resulting from accident, abuse, or any consequential damages as a result of the use of this product.

WARNING: All warranties are void if PromICE is opened!

REPAIR/REPLACEMENT

GEI shall perform all warranty repairs and re-ship the product via 3 day shipping at no charge. Handling charges apply for special shipping considerations. To return a product for repair or replacement, GEI shall provide a return material authorization number (RMA#) which should be clearly marked on the outside of the package. A copy of the invoice or packing slip must accompany the returned items. **Any static sensitive device returned to GEI must be shipped in a static shielding bag. The warranty is VOID if the product is not returned in this manner.**

1. INSTALLATION

1.1. Unpacking

When you take PromICE out of the static shielding you must be in a static safe environment or you could damage the unit. Replacing blown input buffers on the ROM interface is our most common repair. You can develop a long and productive relationship with your PromICE by following a few safety guidelines and handling precautions:

1. **Take the box PromICE came in to the work site. If the box must be opened elsewhere for inspection under no circumstance should you or anyone else remove PromICE unit from its static shielding bag.**
2. **Make sure that a static safe environment is maintained at the work site at all times. Ground yourself by touching a grounding strip before touching any device at the workstation.**
3. **Wear a grounded wrist strap when you handle your PromICE or target system, such as when setting up. Practice safe static handling by not touching any exposed metal parts or pins on PromICE or the target. We recommended you always wear a wrist strap when handling static sensitive devices.**
4. **If you must move PromICE, wear a grounded wrist strap and disconnect all cables from PromICE. Remove power from both the PromICE and target system before removing ROM and auxilliary cables. DISCONNECT ALL CABLES FROM PromICE END FIRST. Return PromICE to its static shielding bag before transporting. Never carry PromICE with the ROM cables still attached. You can damage the buffers easily and also damage the ROM cable pins.**
5. **When handling PromICE do not touch any exposed pins. Most of the pins are direct connections to inputs on buffers and are very sensitive to static damage. A damaged buffer can cause intermittent and permanent failures.**

1.1.1. Packing List

PromICE is shipped in a custom cardboard container with cutouts for various parts. **Do not remove PromICE unit from its static shielding bag until you get to section 2.1 "*Connecting the Hardware*":**

1. PromICE - in its static shielding bag
2. External power supply
3. 6' shielded serial cable (DB9)
4. DB9 to DB25 adapter for serial cable
5. 6' shielded parallel cable (DB25)
6. Reset mini clip
7. User manual - with Host software - LoadICE

PromICEs with model numbers containing “AI” will receive 3 mini clips.

1.2. Hardware Installation

Please read this entire section before attempting to connect or disconnect PromICE from the target. Grammar Engine is not responsible for damages incurred as a result of mishandling or misuse.

STEP 1: Turn power off to your target and disconnect power from PromICE. Failure to turn off power may damage PromICE and/or the target.

STEP 2: Disconnect the ROM adapter from PromICE side **FIRST**.

STEP 3: Connect or disconnect the cable from the target system.

STEP 4: Connect or disconnect the serial and/or parallel cables from PromICE

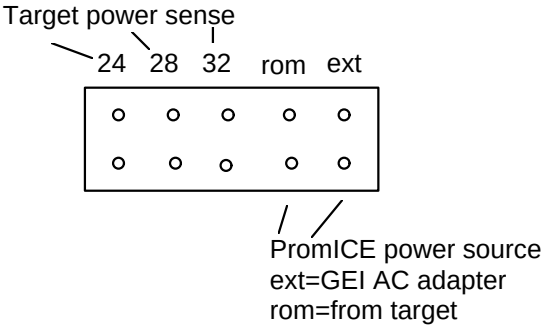
STEP 5: (Disconnecting only) Place PromICE in a static bag.

When you are ready to use PromICE, bring the unopened shipping box to your work area and wear a grounded wrist strap. Remove PromICE, cables and accessories from the box. DO NOT remove the unit from its static protective bag at this time.

Please remember any time you are handling connectors, whether from the target to PromICE or host to PromICE, always remove power first and wear a wrist ground strap. Never connect or disconnect anything without first turning off the power to PromICE and target systems (except the host).

1.2.1. Connecting PromICE to the target system

Power Source Selection for PromICE



Set one jumper on either the "ext" pins for external power (preferred) or the "rom" pins for parasitic power (When using parasitic power, PromICE takes its power directly from the target's ROM socket. No external power supply should be connected to PromICE).

Target Power Sense Selection

Place a jumper on “24” if you are using a 24 pin DIP cable or a jumper on “28” if you are using the 28 pin DIP cable. Place a jumper on “32” for all other cables and/or adapters.

Connecting ROM cables

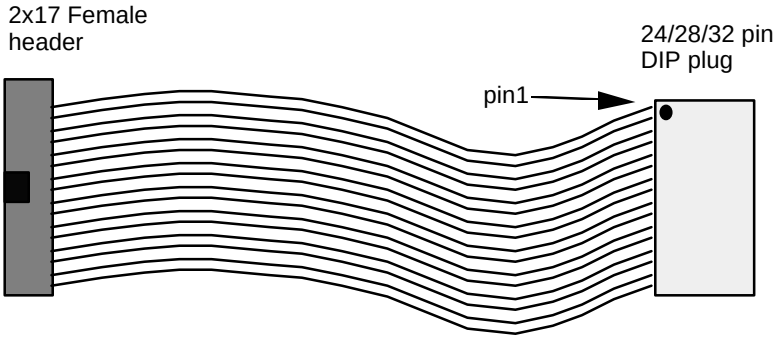
This section describes installation of the 28 and 32 pin DIP cables only. Refer to the included cable manual for installation instructions for other cable assemblies.

If you will be emulating multiple ROMs consider the byte order before connecting the ROM cables. You will have to describe to the LoadICE software which module is plugged into which ROM. Plug the cable(s) into the target system as instructed below:

All DIP cables come with a conditioning pod, labeled “NoNoise/NOZap”. This board contains signal conditioning and protection circuitry. Place this adapter between the PromICE and the DIP cable.

Carefully locate pin 1 of the ROM socket. It is the top left pin when the notch on the socket is on the top. If you do not understand how to locate

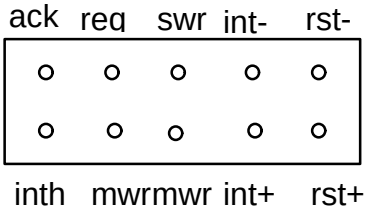
pin 1 of a given socket, then get help before proceeding further. Plugging this cable in backwards will damage PromICE and/or target. Connect the cable to the target ROM(s) with **pin 1** on the cable DIP aligned with pin 1 on the ROM socket. Then, connect the female end of the cable to the conditioning pod..



WARNING: CONNECTING THE CABLE INTO THE ROM SOCKET BACKWARDS WILL DAMAGE PromICE.

Carefully connect the other end of the cable(s) to PromICE via the keyed 34 pin female IDC connector. If you are using a 16 bit or larger system, Make sure the byte order is set correctly .

Optional Connections



The PromICE back panel has auxiliary signals that allow you to control the target system from the host, or allow additional features of PromICE to interact with the target. These signals are as follows:

rst- and rst+ : (outputs) These are reset signals that are driven by PromICE whenever the unit is in LOAD mode or is instructed by the reset command from LoadICE. Both polarities of the signal are provided and are driven by a 74HCT125 tri-state buffer. The signals are driven when

asserted and are tri-stated when not asserted. This allows these signals to be shared by other sources.

int- and int+ : (outputs) These are interrupt signals that are driven by PromICE whenever the HDA is set in the AI status register, when AI is used in transparent mode, or driven by a host request. Both polarities of the signal are provided and they are driven by a 74HCT125 tri-state buffer. The signals are driven when asserted and are tri-stated when not asserted. This allows these signals to be shared by other sources.

mwr / swr: (input) This is a low asserted input that usually comes from the system write line on the target. This allows the target to do write cycles to PromICE master unit (bottom ROM connector) The **swr** pin is used to do writes into the slave module of a PromICE (P2*xxx*). This allows byte writes to the ROMs, you must attach two separate mini-clips to the **mwr** and the **swr** signals.

inth: (input) This signal directly drives the CTS pin on the RSR232 interface on PromICE front panel. It allows the target to directly interrupt or alert the host.

req: (output) This signal is driven directly by PromICE micro-controller to request the target systems bus for PiCOM protocol use. Its polarity is programmable via LoadICE.

ack: (input) This is the input from the target responding to req above. Its sense polarity is programmable by LoadICE.

Reset Line

PromICE will control the target reset, either automatically, i.e. asserted while down-loading data and released when emulating, or it can be asserted by issuing direct command. The reset signal as driven by PromICE is driven only during the asserted state and is tri-stated (i.e. not driven) during the non-asserted state. This allows the connection to be shared. However, you must make sure that your target allows a shared reset. Connecting to unsharable reset on the target will damage the drivers in PromICE as well as on the target. Refer to your target hardware documentation.

If the target reset line is low asserted connect the reset line from the target to the **rst-** pin on PromICE. If the target reset line is high asserted connect the target reset line to the **rst+** pin on PromICE.

If you choose not to connect the reset line between PromICE and the target system, then you must boot your target either by pressing a reset button or by power-cycling the target system.

Interrupt Line

If the target interrupt line is low asserted connect the **int-** line from PromICE to the desired interrupt on the target. If the target interrupt line is high asserted connect the **int+** from PromICE to the desired target interrupt line. Usually, you will want to connect this line to the NMI line on the target system.

PromICE has the ability to interrupt the target system. This feature is used to return control to the debugger if your program runs away. Note which interrupt this line is connected to your target. Some debuggers will want to know which target interrupt line PromICE is connected to.

Write Line

The following describes how to connect the target's write line into the PromICE depending on the target's chip select/write configuration:

8 bit only

Connect the target's write line to either of the **mwr** pins on the PromICE unit.

16 bit (1 target write line)

If your target has only one write line and uses separate chip selects for each byte jumper the **swr** and **mwr** pins together and connect the target's write line.

16 bit (2 target write lines)

If your target has one ship select and uses the write line to determin which ROM is being written to remove the jumper between **mwr** and **swr**. Then, connect the write line that will be used for the bottom PromICE unit to one of the **mwr** pins on the PromICE. Connect the other write line to the **swr** pin on the PromICE.

32 bit (1 target write line)

This configuration requires two PromICE (P2*xxx*) units. Connect the target's write line to each of the PromICE unit's **mwr** lines. Leave the jumper between the other **mwr** and **swr** pins in place.

32 bit (4 target write lines)

This configuration requires two PromICE (P2*xxx*) units. Remove the jumpers on **mwr** and **swr** on each of the PromICE units. Connect one of the target's write lines to each of the pins on the PromICE units.

1.3.2. Connecting PromICE to the host PC

There are four ways to connect your PromICE units to the host system's ports. 1) serial only; 2) parallel only; 3) multiple units on the serial port 4) multiple units on the serial port with parallel for down-loading only.

1.3.2.1. Serial / Parallel connection

Connect the cable(s) to PromICE by attaching the supplied cables from PromICE to the host. Refer to the next chapter for software configuration.

1.3.2.2. Multiple PromICEs on the serial link

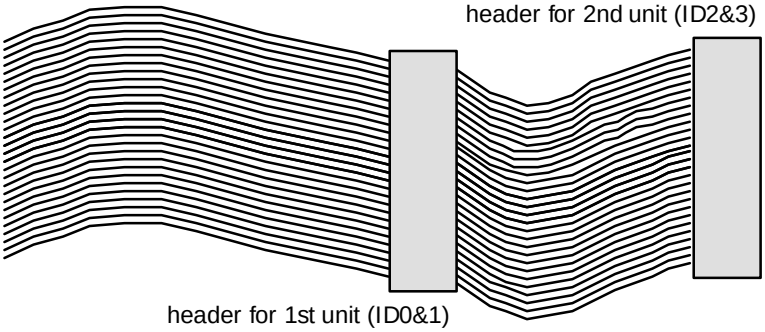
To connect multiple PromICEs to a single serial port, you must use the daisy-chain modules. A daisy chain module lets you create a loop with the transmit signal of one unit going in as the receive signal of the next. You can connect multiple daisy chain modules together to emulate a large configuration of ROMs. Up to 256 modules can be daisy chained from a single serial port.

The daisy chain module that is made up of DB9 connectors and DB9 adaptors.

With the center jack's opening facing you, connect the master (unit 0 on a simplex or units 0 and 1 on duplex units) PromICE to the left connector and the slave (unit 1 on a simplex or units 2 and 3 for duplex units) PromICE to the right connector (the jacks that go to PromICE will be facing away from you). Any number of PromICE units can be daisy chained in this way.

1.3.2.3. Multiple PromICEs The Parallel Bus Cable

When using PromICE in this configuration, the serial port is used for full communication with PromICEs and the parallel link is used to achieve fast down-load times. The serial daisy chain adapter determines the byte ordering, so byte ordering isn't important when connecting the parallel bus cable.



Connect the serial daisy chain adapter(s) as described in the previous section (1.3.2.3 *Multiple PromICEs on the serial link*). Connect the parallel bus cable in whatever way is most convenient.

This completes the connection of PromICE hardware. Your target system should now be connected to PromICE properly and PromICE should be connected to the proper host ports. You can now proceed with the setup of the host software for down-loading and emulating your code. It is best to check your setup by simply down-loading known working code and verifying that your target system and PromICE are functioning properly and that you can repeat the down-loading and restarting process.

1.3. Software Installation

The software you will need to accomplish most of your work with PromICE is the LoadICE application. It is distributed with a command line user interface for all PC and UNIX based systems. The software comes on a 3.5" floppy disk with the application and sources on it.

If the distribution disk has an executable LoadICE application for your system, then you do not need to install the sources on your machine. Otherwise install the sources and use the "make" command to create the LoadICE executable. You would most likely do this if you are using a UNIX system.

1.3.1. DOS / Windows 3.x / Windows 95

Insert the floppy disk in to drive A: or B: as appropriate

If you have a directory on your system for local commands or development tools then its is best to copy just the LoadICE executable to this directory. For example, assume that you keep a set of your non-DOS commands in a directory called CMD. This directory should also be in your PATH. Run the following to install LoadICE on your system:

```
copy a:\dos\loadice.exe c:\cmd
or
copy a:\win95\loadice.exe c:\cmd
```

Now you can execute LoadICE from anywhere in your system.

1.3.2. Windows NT 4.0

If you will only be using serial communications, follow the instructions in 1.3.1 to install LoadICE to copy LoadICE from the \winnt directory to your hard disk.

Follow these steps to installe the parallel port driver:

1. Make sure the port is in either Standard or EPP mode. This is set in the BIOS of the system.
2. While in BIOS setup, note the address of the port. This is very important, since some newer computers do not necessarily use standard addresses.
3. Reboot and login to Windows as administrator.

4. Edit the “PromICE.ini” file in the \WINNT\PromICE directory. Change the IoPortAddress to match the address of your parallel port.
5. If necessary, edit the PromICE.bat file. This should only be necessary if you installel NT into a directory other than “C:\winnt”.
6. Run PromICE.bat
7. Restart the computer.
8. From the Start Menu, select Settings->Control Panel.
9. In Devices, there should be a service called PromICE. It should be started and automatic.

1.3.3 UNIX

If you have a 3.5" floppy drive on your workstation, then the software can be installed from a DOS diskette as follows (This example is taken from a SUN workstation):

Insert the floppy disk in the drive and do the following commands:

```
mkdir LoadICE
cd LoadICE
mount /pcfs
cp /pcfs/source/*.* .
dos2unix makefile.unx makefile

(if you plan to edit any of the files you can
translate all the files to UNIX format: example using
shell:
sh
for file in *
>do
>echo $file
>dos2unix $file $file
>done

make
eject
```

This will make a LoadICE executable. Ignore warnings during compile time that are caused by multiple includes in the UNIX header files. You need only convert the makefile from DOS to UNIX format. You also have other makefiles that are specific to other types of UNIX, i.e. there is a `makefile.hp` for Hewlett Packard and also `makefile.unx` as a generic make file.

2. Software Configuration

If you haven't already done so, go back and read the hardware installation instructions before continuing with this section. This section details the software configuration only.

Fastport Users: A serial connection is required for communication with the Fastport. The parallel port is optional, but allows for faster downlad speeds.

1. The LoadICE software uses the LoadICE.ini file to initialize PromICE to match the ROM that is being emulated. The LoadICE software is case sensitive. All entries should be lower case unless otherwise specified. For more information on any of the following LoadICE.ini instructions, refer to Chapter 3, *Command Reference* of this manual.

2. If PromICE is using the serial port for communications, the first line in the LoadICE.ini file should read:

```
output=comX:abc
```

Where:

`X`	is the serial port number being used.
`:abc`	(optional) Specify only if the port address is not the default address.

3. If PromICE is connected via the parallel port only (Fastport and daisy chain users cannot use this command, goto step 4 below), the specification should be:

```
pponly=lptX:abc
```

Where:

`X`	is the parallel port number being used.
`:abc`	(optional) Specify only if the port address is not the default address.

4. If PromICE is connected via the parallel port in addition to the serial, the specification should be:

```
ppbus=lptX:abc
```

Where:

`X`	is the parallel port number being used.

`:abc` (optional) Specify only if the port address is not the default address.

You may also have a ppmode command and number statement to use bus mode of parallel port.

```
ppbus lpt2
number 4
```

The statement "`number = 4`" tells LoadICE that there are four PromICE units attached to the parallel port "bus". Using the parallel port bus also requires the serial daisy chain module be used, so the output and baud statements will also be needed.

5. Specify the socket size of the ROM to be emulated. The socket size is the largest ROM that the target can use. For example: If you are trying to emulate a 27010 (1Mbit) part, but you can put a 27040 (4Mbit) part in your target without rewiring it, specify the socket size as:

```
socket=27040
```

The socket size should equal the largest ROM that can be used by the target. This will ensure that both LoadICE and the target 'see' the same address space within PromICE units. This is critical when you are emulating a ROM smaller than the amount of memory PromICE unit has. This will ensure that if you are using a debugger with PromICE, it will communicate via the AI option correctly.

NOTE: ALL UNUSED PINS ON THE ROM SOCKET SHOULD BE TIED HIGH. PROMICE MAY NOT EMULATE PROPERLY IF THERE ARE ANY FLOATING LINES ON THE ROM SOCKET.

6. Specify the ROM size to be emulated. This size should not exceed the size specified in the socket statement above. Place the following line in the LoadICE.ini file after the baud or pponly statement:

`rom=27512` Specify the part number of the ROM (i.e. 27512), or
`rom=64k` Specify the ROM size in bytes.

If you are using a 40 pin DIP or a 44 pin PLCC set the ROM statement to emulate half the size of the ROM you are emulating. PromICE master and slave modules will add up to the size you want to emulate.

7. Specify the word size (and byte order if necessary). The word size is the width of the target bus in bytes. Place the line as follows:

`word=8`	for an 8 bit word
`word=16 0 1`	For a 16 bit word with the first byte going to the "0" (master) PromICE unit and the second byte going to the "1" (slave) unit. To reverse the byte order, swap the "0" and "1".

Unless another order is specified, the default order is 0, 1, 2,... The byte order allows you to hook your PromICE modules in the order most convenient for your target.

If you are using a 40 pin DIP adapter or a 44 pin PLCC adapter set the word size to 16.

8. The file specification is used to specify the file and where the data is to be loaded. To load multiple files, simply duplicate the following statement for each of the files to be loaded. LoadICE currently supports the fololowing file formats: Intel 8 and 16 bit hex, Motorola S record format, Tektronix standard and extended hex, Mostek and RCA formats. In HEX files each record contains the address where the data must be loaded. The following specs lets you map the hex file to desired location in ROM:

```
file=filename fileaddr=[id:] romaddr
```

Where:

`filename`	The file that is to be loaded
`fileaddr`	The file's starting address (i.e. the hex records in the file indicate to start loading the data at this address).
`id:`	(optional) PromICE unit ID number that the file is to be loaded into. This is an optional specification. If PromICE is a duplex unit or more than one unit is being used. The bottom unit in the duplex PromICE is ID 0, or lower ID. Do not use IDs unless you are using per ROM files. Improper use can override your word size without your realizing it.
`romaddr`	The location in PromICE where the code is to be loaded.

If a binary image is to be used the line will appear as: (since there is no explicit address information in a binary file, the specs allow you to locate it anywhere in the ROM):

`image=filename skipcount=[id:]romaddr`

`skipcount` Allows you to skip any data (usually load header) from the beginning of the file. There are no other differences in syntax between the two file specifications.

To load multiple files or images, repeat the file/image statement for each file/image to be loaded:

file = file1.hex 8000=0
file = file2.hex 10000=2000
…

Make sure that you are not loading your files over top of one another. You can do a compare ‘c’ in LaodICE dialog mode to see if any of the files are overlapping.

At this point the LoadICE.ini file should appear like one of the following:

```
output=com1
baud=57600
socket=27040
rom=27010
word=16 0 1
file=mad.hex fe000=0:0   *Sets the  file's address that starts at
                         * fe000 to position 0  in the ROM

pponly=lpt1              *This tells LoadICE the parallel port
                         is
socket=27080             *being used as a bi-directional link
rom=27040
word=8
file=mad.hex fe000=0     *The "0:" specification is not
                         necessary *in an 8 bit configuration

output=com1              *Use the output, baud and parallel
                         *specifications if you want to use
                         parallel
```

```
baud=19200              port downloading and the serial port
                        for
ppbus=lpt1              *communications
socket=27010
rom=27256
word=8
file=mad.hex fe000=0
```

9. Apply power to PromICE and the target system. The RUN light on PromICE should come on.

10. Execute LoadICE from the directory where the LoadICE.ini file is located. The RUN light will blink as LoadICE connects. If you are using the serial link, you will also see the activity on RxD and TxD lights.

11. Cycle power to the target or press the reset button (This is not necessary if the reset line is connected to the target). The LOAD light on PromICE should have already been out. If you boot your target by power up then the LOAD light should go OFF when target power is turned on.

You now have a working configuration for ROM emulation. If you are using PromICE AI (debugger users) proceed to Chapter 6 "*Analysis Interface Configuration*" section of this manual. The next chapters describes how PromICE interface operates and shows examples of basic PromICE functions.

If the target doesn't run or the LOAD light won't go out or you are experiencing some other failure double check your previous steps. If there is any sign of problems with power, such as dim or flickering lights on PromICE or any target LEDs, immediately shut off power to target system and double check all connections.

If you find nothing wrong, check your ROM configurations, file mapping and target booting process. Consult Chapter 4: *Troubleshooting* for common configuration problems before calling for technical support.

3. LoadICE Command Reference

3.1. OVERVIEW

LoadICE commands fall into four major categories which are summarized in this chapter. You should be able to find a particular command within a specific category and then look it up on the following reference pages. If you are interested in upgrading your LoadICE version you may download it free from either our **WEB** or **ftp** site. If you don't have access to either one of these services contact your Sales Representative to obtain a copy.

Our WEB site is located at www.gei.com.

To obtain new software over Internet from an anonymous ftp site: (loadice executable is for SUN sparc stations, for other systems you must do make):

```
ftp -i ftp.gei.com
login:anonymous
password: <your e-mail address>
cd /gei/loadice <latest LoadICE version number>
ftp> bin
ftp> mget * (or get loadice)
```

3.1.1. Host to PromICE communication

These commands allow you to specify the link between the host and PromICE unit(s). You may be using both the serial and the parallel link. These commands allow you to completely specify your communication configuration:

output	serial device name
baud	baud rate
fast	Adjusts parallel port timing
number	number of PromICE units on daisy chained (UNIX only)
ppbus	Connect multiple PromICEs for parallel download
ppmode	Sets parallel port's communications mode
pponly	parallel bi-directional device name
fastport	hostname of FastPort when using PromICE on Ethernet
resetfp	reset the FastPort before connecting to PromICE

3.1.2. ROM specifications and ROM operations

These commands allow you to describe your ROM configuration. The number, size and arrangement of ROMs must be specified and any peculiarities of the target addressing , such as mismatch between sockets and ROMs, must be specified for proper operation. Finally, all the operations that you may perform on the ROM data are specified:

rom	Set size of rom to be emulated
word	Set word size for roms being emulated
socket	Target rom socket size
checksum	Rom checksum specifications
fill	Rom fill specification
dump	Dump rom data
edit	Edit rom data
move	Move rom data
search	Search for ASCII data in rom space
find	Search for binary data in rom space

3.1.3. File specifications and file operations

These commands allow you to specify the data files on the host system by name, type and configuration information such as word size of the data they contain, or special mapping of the data to ROM space. Various loading and processing options can also be specified:

bank	Bank emulated memory
file	Specify hex data files
image	Specify binary data files
load	Down-load the data files
compare	Compare files with rom contents
noaddrerr	Ignore data that falls out of rom space
map	Turn off or on the display of data areas being loaded
save	to save rom contents to a file

3.1.4. Miscellaneous specifications

dialog	Enter dialog mode on startup
display	Change output level detail
stop	Stop PromICE units form emulating
go	Turn on PromICE emulation
help	On line help
log	Record all LoadICE command traffic to a log file
reset	Set default reset time length
ver	Report LoadICE and PromICE micro-code versions

hso	Define the operation and polarity of the interrupt signal
config	Display configuration data in use
notimer	Disable PromICE internal timer
fkey & altfkey	Assign commands to function keys
!system	Escape commands to the host shell
exit	Exit LoadICE dialog mode
delay	Change the time out period used by LoadICE
status	Display target status (power on; executing)
sleep	Used for waiting for something to get done in batch mode
stop	Turn off PromICE emulation
test	Test emulation memory

3.2. .

Bypass the LoadICE.ini file.

3.2.1. Command Forms

. Command line

3.2.2. Syntax

{**.**} [*options*]

3.2.3. Use

. Specifies to bypass the LoadICE.ini file.
options Any command line options available to LoadICE.

3.2.4. Default

When this directive is used, LoadICE will try to connect PromICE using the default configuration:

```
output=com1
baud=19200
word=8
socket=max_size
rom=max_size
```

where:

max_size The maximum memory size of your PromICE unit.

3.2.5. Description

LoadICE automatically looks for a configuration file called *LoadICE.ini* in the current directory. Specifying the "." on the command line bypasses this default.

3.2.6. Notes

N/A.

3.2.7. Examples

```
loadice . -q
```

Bypass the LoadICE.ini file. Connect to PromICE using the parallel port *LPT1* only (refer to pponly in this section for more information on the parallel port option).

3.3. ! system

Escape command to DOS or UNIX shell.

3.3.1. Command Forms

! Dialog mode

3.3.2. Syntax

{**!**} {*command* | *"string"*}

3.3.3. Use

command The command word.

"string" The command string to be executed by the operating system shell. After the command completes control is returned to LoadICE.

3.3.4. Default

None.

3.3.5. Description

This command allows you to execute arbitrary operating system commands without leaving LoadICE.

3.3.6. Notes

This command does not work on Macintosh or VMS systems.

3.3.7. Examples

```
!dir
```

Show the files in current directory.

```
!edit myfile
```

Run the editor and edit 'myfile'. When you exit the editor you will be back at LoadICE: prompt.

3.4. *@filename*

Use LoadICE configuration file *filename*.

3.4.1. Command Forms

@filename Command line

3.4.2. Syntax

{@} [*path_name*] {*file_name*}

3.4.3. Use

path_name The directory path in which the configuration file is located.

file_name The name of the configuration file to use.

3.4.4. Default

LoadICE will automatically look for the file *LoadICE.ini* in the current directory. When the @ directive is used, LoadICE will use the file as specified.

3.4.5. Description

This command allows you to use multiple configuration files for LoadICE without the necessity of keeping each configuration in a different location.

3.4.6. Notes

There should be no spaces between the "@" symbol and the first character of the path.

3.4.7. Examples

```
loadice @c:\mycfg\myini.ini
```

Tells LoadICE to look for a configuration file called "`myini.ini`" in directory "`c:\mycfg\`"

3.5. afn

Allows assigning hot keys to LoadICE or host commands.

3.5.1. Command Forms

afn#	LoadICE.ini file
-afn#	Command line
afn#	Dialog mode

3.5.2. Syntax

{**afn#**=*word* | *"string"*}

3.5.3. Use

#	(decimal number) alt-function key number
word	A LoadICE command.
	OR
"string"	System shell command.

3.5.4. Default

No keys are assigned.

3.5.5. Description

Allows you to assign LoadICE commands or system commands to function keys. You can, for example, edit and compile your source file without exiting LoadICE by assigning command strings to function keys.

3.5.6. Notes

Assign regular word commands directly but enclose strings in double-quotes. Operating system commands must be preceded by a '!'. Strings on command line require that \" be used.

3.5.7. Examples

```
afn12=restart
```
issues the restart command to LoadICE when ALT key and F12 key are pressed.

```
afn1="!edit test.c"
```
When the alt-function1 key is invoked, LoadICE will automatically execute the operating system command "edit" to edit "test.c". When you exit the editor control will return to LoadICE.

3.6. bank

Allows the emulation space to be broken in to a number of banks.

3.6.1. Command Forms

bank	LoadICE.ini file
-ba	Command line
bank	Dialog mode

3.6.2. Syntax

{**bank** | **-ba**} {#}

3.6.3. Use

specifies the number of banks to divide the ROM space into.
after first specification it chooses the bank # you want to use

3.6.4. Default

Banking is disabled.

3.6.5. Description

This command is used when PromICE emulates ROM divided into banks. The first occurrence of the bank statements defines the number of banks. The subsequent statements select the bank number to use for the current operation.

3.6.6. Notes

When using this command the emulation space is effectively reduced to the size of a given bank. Only files that will fit within the bank can be loaded.

3.6.7. Examples

This example shows loading a single file into multiple bank. from LoadICE.ini file:

```
file=mon.hex      ; the file to be loaded in multiple banks
bank 4        ; divide ROM into 4 banks
bank 1        ; select bank one
.load             ; load files here
bank 3        ; select another bank
load          ; load files here
.file = app.hex ; new file
bank 2        ; select bank
load          ; load file
```

3.7. baud

Specify baud rate for serial communication between PromICE and the host. (See *output*)

3.7.1. Command Forms

baud	LoadICE.ini file
-b	Command line

3.7.2. Syntax

{**baud** | **-b**} {*baud_rate*}

3.7.3. Use

{baud_rate}	(integer) A valid baud rate (1200, 2400, 4800, 9600, 19200, 57600)

3.7.4. Default

19200 baud

3.7.5. Description

This command specifies the baud rate a particular serial port.

3.7.6. Notes

The 38400 baud rate is not supported, however, 57600 is supported.

3.7.7. Examples

```
-b 57600
baud 57600
```

Sets the baud rate of the specified device to 57600.

3.8. checksum

Perform checksum on ROM and store in PromICE memory.

3.8.1. Command Forms

checksum	LoadICE.ini file
-k	Command line
k	Dialog mode

3.8.2. Syntax

{**checksum** | **-k** | **k**} [*id* {:}] [*start*] [*end*] [*store*] [*function*] [*sum_size*] [*order*]

3.8.3. Use

id	(integer) A valid PromICE unit ID number (0-255).
start	(hex) An address in PromICE where the checksum should start.
end	(hex) An address in PromICE where the checksum should stop.
function	(character) Indicates the preferred checksum function (*x* , X or *a, A*). An '*x*' indicates exclusive OR function will be performed on data in all locations within the selected address range. An '*a*' indicates addition function will be performed on data in all locations in selected address range. Capital X causes a 1's complement of the result to be stored where as small x stores the result as is. A capital A causes a 1's complement of the addition be stored, a lower case a causes a 2's complement to be stored.
sum_size	(integer) The size of checksum. This must be an integral multiple of 8 and cannot be larger than the data bus width emulated by the total number of daisy chained PromICE units.
order	(0|1) checksums, longer than a byte, are stored high byte first, unless this argument is supplied and is '1', then the low byte is stored first.

3.8.4. Default

You must specify the start, end and store arguments. Default for ID is 0, default for function is '*x*' and sum size is 8.

3.8.5. Description

Compute and store a 8, 16 or 32 bit checksum in the ROM. The checksum is performed on a "per ROM" basis or for the entire configuration. The checksum is computed inclusive of the start and end addresses, then the results are stored in the given store address. The checksum is displayed.

If an ID is specified, then perform a checksum on that particular ROM, otherwise, checksum all ROMs. If it is used from dialog mode then checksum the current configuration.

You can specify a specific checksum method for each ROM, in the ini file. This allows you to mix and match custom checksums of your ROM space.

3.8.6. Notes

Checksum is performed by uploading the data from the ROM and then stored back. Larger ROMs take longer to compute the checksum. You can specify separate checksums for each ROM.

3.8.7. Examples

```
k 0 fffb fffc a 32
```

Compute the checksum on 0 through FFFB and store the resulting checksum (2's complement thereof) as a 32 bit number at (FFFC-FFFF). High byte is stored first.

3.9. compare

Compares data loaded in PromICE against files on the host.

3.9.1. Command Forms

c Dialog mode

3.9.2. Syntax

{**c**}

3.9.3. Use

c Compares files instead of down-loading them

3.9.4. Description

This command will process the data files and upload PromICE contents and compare it with the data files. It is an explicit verification of file data. Differences are displayed on the screen and there is an option to continue displaying the differences or canceling the compare.

3.9.5. Notes

If a compare operation fails, it is most likely due to overlapping data areas in different files. If that is not the case then it may be overlapping data areas in the same file. If a 'write' line is connected to the target and the target was emulating then it could also have 'written' to location to the emulated space and thus cause the compare failures.

If the unit is failing to compare after loading and none of the above problems are the cause, there may be problems with the battery backup or memory in PromICE unit.

3.9.6. Examples

c

Compare now.

3.10. config

Display current PromICE configuration.

3.10.1. Command Forms

config	LoadICE.ini file
C	Dialog mode

3.10.2. Syntax

{**config** | **C**} [*link* | *rom* | *file* | *all*]

3.10.3. Use

link	Displays diagnostic information about the communication link (serial or parallel)
rom	Displays diagnostic information about the ROM configuration
file	Displays diagnostic information about the list of files loaded into PromICE
all	Displays all diagnostic information

3.10.4. Default

all

3.10.5. Description

Displays the current LoadICE configuration data. This command will display just about all the information relevant for diagnosing any setup problem. It will display information regarding communication links, both serial and parallel as well as various operating mode.

3.10.6. Notes

Use this command when you need to determine the word size, emulation size and/or file information that is being used by LoadICE

3.10.7. Examples

```
C all
```

Display the entire configuration.

3.11. delay

Change the value of time period when LoadICE is waiting on response from PromICE.

3.11.1. Command Forms

delay	LoadICE.ini file
delay	Command line

3.11.2. Syntax

{delay} [#]

3.11.3. Use

#	(decimal number) multiplier results in #x4 seconds of delay.
0	no delay (wait indefinitely).
<no arg>	restores default.

3.11.4. Default

Delay 4 seconds.

3.11.5. Description

This command allows control over the delay and keeps LoadICE from timing out. The nominal delay is 4 seconds. This command supplies a multiplier for this period.

3.11.6. Notes

While executing certain commands where response may take variable amount of time, LoadICE internally shuts off the delay, such as when executing 'test' command to test PromICE memory.

3.11.7. Examples

```
delay 8
```

Wait 32 seconds before timing out.

3.12. dialog

Enter LoadICE dialog mode.

3.12.1. Command Forms

dialog	LoadICE.ini file
-d	Command line

3.12.2. Syntax

{**dialog** | **-d**}

3.12.3. Description

This command will force LoadICE to enter interactive dialog mode.

3.12.4. Notes

Normally LoadICE will process and down-load user data files according to the configuration specified. However, if no data files are specified LoadICE will go into an interactive 'dialog' mode also. In this mode any commands, files and operations can be specified and done interactively.

3.12.5. Examples

```
loadice -d
```

Invoke LoadICE application and enter the dialog mode.

3.13. display

Change the output display level of LoadICE.

3.13.1. Command Forms

display	LoadICE.ini file
-D	Command line
D	Dialog mode

3.13.2. Syntax

{**display** | **-D** | **D**} [*level*]

3.13.3. Use

level (hex) A valid number (0 - FF) indicating the level of diagnostic display desired. Bits are as follows:

0x80 - displays prompts
0x40 - displays progress
0x20 - displays command parser data
0x10 - displays config data, disk i/o and buffer transfer
0x08 - displays hex record processing
0x04 - displays abbreviated commands and responses to/from PromICE
0x02 - displays full commands and responses to/from PromICE
0x01 - displays actual data bytes going over the link

3.13.4. Default

0xC0 - only displays main prompt and command progress and results.

3.13.5. Description

This command allows the display of more or less information about a command during it's processing.

3.13.6. Notes

Setting the display level to 0x00 will shut off everything except the command results. Setting the level to 0xFF can cause data overflow over the serial link.

3.13.7. Examples

```
-D fe
```

Display everything but the actual data going over the link.

3.14. dump

Display PromICE memory contents on the screen.

3.14.1. Command Forms

d Dialog mode

3.14.2. Syntax

{**d**} [[*id* {:}] [*start*] [*end*]]

3.14.3. Use

id	(integer) A valid PromICE unit ID number (0-255)
start	(hex) The first PromICE address to display
end	(hex) The last PromICE address to display

3.14.4. Default

The ID is zero or use in the current configuration. The default for start address is 0 and the end is start+64 (or the last valid address in ROM).

3.14.5. Description

The dump command will display PromICE data in hex and ASCII with 16 bytes per line. Data is displayed in the configuration that is active, i.e. you can view data as 8, 16, 32 (up to 64) bits at a time with the proper configuration setup. A <cr> will simply repeat the command with next range of arguments.

3.14.6. Notes

None.

3.14.7. Examples

```
d 0 ff
```

Dump data from 0x0 to 0xFF address range.

```
d 1:fe ff
```

Dump data from unit ID-1 from 0xFE to 0xFF (two bytes).

3.15. edit

Modify PromICE emulation memory (edit ROM space).

3.15.1. Command Forms

edit	LoadICE.ini file
-e	Command line
edit	Dialog mode (also **e**)

3.15.2. Syntax

{**edit** | **-e** | **e**} [[*id* {:}] [*address*] [[*value*] ...]]

3.15.3. Use

id	(integer) A valid PromICE unit ID number (0-255).
address	(hex) Address in PromICE to start memory modification.
value	(hex) A value or list of values to be written to memory starting at '*address*'. The width of individual values can be any integral multiple of 8, but less than or equal to the data bus width emulated by the total number of daisy chained PromICE units.

3.15.4. Default

Start editing at address 0 of the current configuration.

3.15.5. Description

Edit changes one or more data bytes in one or more ROMs using the current configuration. When edit is in the ini file or the command line, bytes are edited after loading any specified files. In dialog mode, edit will enter interactive mode if no data values are specified.

3.15.6. Notes

The use of edit in the ini file or command line is intended for patching ROM data after the files have been downloaded. To stop editing in dialog mode, type a period followed by enter. This will return you to the LoadICE prompt.

3.15.7. Examples

```
edit 1:500 ab cd
```

Will edit location 0x500 and 0x501 in unit 1 with values 0xAB and 0xCD.

3.16. exit

Exit LoadICE when in dialog mode.

3.16.1. Command Forms

exit Dialog mode (also x or `quit`)

3.16.2. Syntax

{**exit** | **x** | **quit**} [#]

3.16.3. Use

x exit LoadICE
also set the exit code to this number

3.16.4. Default

Exit code is 0 if all goes well, if error is encountered then exit code is 1.

3.16.5. Description

Exit the LoadICE application.

3.16.6. Notes

The exit code is useful when running LoadICE in batch mode. It can be used to determine if an error occurred .

3.16.7. Examples

```
exit
```

Exit LoadICE application.

3.17. fast

Lengthens the strobe on the parallel port on a host with a high performance host.

3.17.1. Command Forms

fast LoadICE.ini file

3.17.2. Syntax

{**fast**} [#]

3.17.3. Use

To lengthen the parallel port strobes, by a factor of

3.17.4. Default

The strobe is generated by turning it ON and then OFF. If this command is specified then a delay loop is inserted between ON and OFF. The default value for the loop count is 10.

3.17.5. Description

This command allows you to lengthen the strobe by specifying a count for a delay loop. This delay loop is only inserted if this command is present, otherwise the length of the strobe depends on how long it takes to execute the instructions.

3.17.6. Notes

If parallel port transfers fail, use this command to lengthen the strobe. The strobe is lengthened by inserting an idle loop between the assertion and removal of strobe signals.

3.17.7. Examples

```
fast 20
```

makes it work on a 100MHZ RISC machine.

3.18. fastport

Specifies the host name for FastPort on the network to LoadICE.

3.18.1. Command Forms

fastport	LoadICE.ini file
-fp	Command line

3.18.2. Syntax

{**fastport** | **-fp**} {*hostname*}

3.18.3. Use

hostname	The name of the host for the fastport on the network. In UNIX it is generally kept in the /etc/hosts file.

3.18.4. Default

N/A

3.18.5. Description

To access PromICE system including the AI option over a network, the FastPort print server may be utilized. FastPort supports serial and parallel printers on Ethernet networks. PromICE units are attached to the FastPort printer ports and then LoadICE software is used to access PromICEs over the network.

3.18.6. Notes

The FastPort is a product made by Milan Technologies. Follow the installations instructions that come standard with the FastPort. Do not install the printer server software. Follow the instructions for configuring the FastPort for use with PromICE from the installation instructions supplied by Grammar Engine.

3.18.7. Examples

`fastport=pint`	Communicate to FastPort "pint" in LoadICE.ini file.
`-fp pint`	Use FastPort "pint" using LoadICE command Line.

3.19. file

Specify hex record file information for downloading or compare operation.

3.19.1. Command Forms

file	LoadICE.ini file
<no arg>	Command line
file	Dialog mode

3.19.2. Syntax

{**file** | <no arg>} [[*file_name*] [*link_address* =] [*id* {:}] [*ROM_offset*] [*data_width*] [[*byte_n*] ...]] [(a1,a2)]

3.19.3. Use

file_name (string) The name of the hex record file to be loaded.

link_address (hex) The absolute starting address of the file '*file_name*' produced by the linker, that is stored incrementally in each hex record of the file '*file_name*'. This address is normally used by hex record decoding programs to determine where in memory to place the data.

id (integer) A valid PromICE unit ID number (0-255)

ROM_offset (hex) An offset value within PromICE emulated ROM space. The start of PromICE emulated ROM space is assigned to offset address 0. This offset is used to determine the location within PromICE emulated memory where the '*link_address*' should be assigned.

data_width (integer) The data bus width associated with the width of data objects in file '*file_name*'. It must be an integral multiple of 8 and can not be larger than the data bus width emulated by the total number of daisy chained PromICE units.

byte_n (integer) The nth byte in a data bus that has a width that is an integral multiple of 8 bits. Bytes are assigned to successive PromICE units (0 - 255) based on the selected order of *byte_n* specifications.

(a1,a2) (hex addresses) transfer data only from address a1 through address a2, inclusive. This specification allows partial loading of data files.

3.19.4. Default

Load the file in current configuration starting at address zero.

3.19.5. Description

This specification allows you to specify the file for down-loading to PromICE. The file type can be specified if it is binary, else it is assumed to be hex. The command allows for provision to specify whether the data in the file needs to be mapped or relocated within the ROM space. The word width of the data in the file as well as byte order within the word can be specified. Partial loading of a data file can be specified.

File formats supported: Intel 8 and 16 bit, Motorola S record, Tektronix standard and extended hex, Mostek and RCA.

3.19.6. Notes

Only one set of partial loading addresses can be specified.

You need not specify the ID unless you are loading files "per ROM" (i.e. a file gets loaded into one ROM and another file gets loaded into the other ROM). If you have a word size of 16 or larger then specifying the ID says to load the file into a bank where the given ID is the first ID. If such a bank is not found then the file is treated as an 8 bit file for that particular ROM. This inadvertent side affect can be avoided by careful consideration of ID specification.

To load mutiple files in the LoadICE.ini file, add a new file statement for each file to be loaded.

3.19.7. Examples

```
file=myfile.hex 400000=0 16 1 0
```

Load data from '*myfile.hex*' with addresses starting at 0x400000 in the file to be mapped to 0x0 in PromICE emulation memory and that the file contains 16 bit data with the first byte (even) to be written to PromICE unit ID-1 and the second byte (odd) to be written to PromICE unit ID-0.

3.20. fill

Fill PromICE memory with repeating data (fill pattern).

3.20.1. Command Forms

fill	LoadICE.ini file
-f	Command line
f	Dialog mode

3.20.2. Syntax

{**fill** | **f** | **-f**} [[*id* {:}] [*start*] [*end*] [*data*] [*data2*] [*size*]]

3.20.3. Use

id	(integer) A valid PromICE unit ID number (0-255)
start	(hex) The first PromICE address to fill.
end	(hex) The last PromICE address to fill.
data	(hex) The data value with which to fill the address range. The width of the data value can be up to four bytes.
data2	(hex) If longer then 32 bytes of pattern are to be used then the lower long word is specified.
size	(decimal) The length of the pattern in bytes when it is not same as the current word size.

3.20.4. Default

Fill all of the ROM space with the default fill character (0x0FF).

3.20.5. Description

Used for filling all or part of ROM space with a fill character. Individual ROM or entire configurations can be filled. The fill character matches 8, 16, 32 or 64 bit word configuration (64 bit data is specified as two 32 bit items). You can load patterns 1 to 8 bytes long in any configuration.

3.20.6. Notes

Filling is done prior to loading the file data, when specified in the LoadICE.ini file. You can specify unique fill parameters for each ROM.

3.20.7. Examples

```
f 200 300 ab
```

Fill from 0x200 to 0x300 with data value 0xAB

3.21. find

Find binary data patterns in PromICE memory.

3.21.1. Command Forms

F Dialog mode

3.21.2. Syntax

{**find** | **F**} [[*id* {:}] {*start*} {*end*} {*size* } {*data bytes*}

3.21.3. Use

id (integer) A valid PromICE unit ID number (0-255)

start (hex) The first PromICE address to start search

end (hex) The last PromICE address to search

size (integer) The number of bytes to find, max 32

data bytes (hex) The data values to look for. The sequence of bytes to search for in the specified address range. The data must be specified as hex bytes separated by spaces.

3.21.3. Default

All information except for the ID must be specified.

3.21.4. Description

Find allows you to search PromICE memory for binary data patterns. You can search individual ROMs or ROM configurations.

3.21.5. Notes

None.

3.21.6. Examples

```
F 0 1ffff 4 de ad fe ed
```

Looks for the pattern 0xDEADFEED in a 128kB ROM space (0x-0x1FFFF).

3.22. fn

Allows assigning hot keys to LoadICE or host commands.

3.22.1. Command Forms

fn#	LoadICE.ini file
-fn#	Command line
fn#	Dialog mode

3.22.2. Syntax

{**fn#=**} {*word* | *"string"*}

3.22.3. Use

#	(decimal number) function key number (1-12)
word | *"string"*	LoadICE or system shell command. Precede the system commands with a '!' so that LoadICE will pass them to the DOS or UNIX shell.

3.22.4. Default

N/A

3.22.5. Description

Allows you to assign LoadICE commands or operating system commands to function keys

3.22.6. Notes

Assign commands directly but enclose strings in double-quotes. Operating system commands must be preceded by a '!'. On the command line the " must be specified as \".

3.22.7. Examples

```
fn12=restart
```

This will issue the restart command to LoadICE when F12 key is pressed. You can then request LoadICE to restart the link with PromICE after a time-out error.

```
fn1="!edit test.c"
```

When the F1 key is invoked LoadICE will automatically execute the operating system command '*edit*' to edit the file '*test.c*'. When you exit the editor you will return to the LoadICE prompt.

3.23. go

Instruct PromICE to go into emulation mode.

3.23.1. Command Forms

go	Dialog mode
go	LoadICE.ini file

3.23.2. Syntax

{**go**}

3.23.3. Description

Allows you to start emulation from the dialog mode. The status of PromICE emulation state is reflected in the load light on the front panel. When the go command is executed the load light should go out (unless the target is not powered up).

3.23.4. Notes

You can also use the [ESC] key to switch from load to emulate and vice-versa.

3.23.5. Examples

<ESC>

PromICE will toggle between LOAD and EMULATE.

3.24. help

Obtain help about a LoadICE command.

3.24.1. Command Forms

?	Command line
help	Dialog mode

3.24.2. Syntax

{**help** | **?**} [*command*]

3.24.3. Use

command (string) Any valid LoadICE command

3.24.4. Default

Display the list of commands for which help is available.

3.24.5. Description

Help will give you on-line help for any command. When invoked with a '?' as the only argument on the command line, it will give you help on all the commands that can be specified in the LoadICE.ini file. When invoked as '?' or 'help' in dialog mode, it will display a list of all the commands available. Further help then can be obtained on each individual command.

3.24.6. Notes

Only the most common and useful commands are included in on-line help. Full documentation is available only in this manual. Or you can print out PISCRIPT.H file for all command scripts.

3.24.7. Examples

```
help find
```

In dialog mode this will give information on how to use the 'find' command.

3.25. hso

Program the interrupt signal to the target (on PromICE back panel).

3.25.1. Command Forms

hso	LoadICE.ini file
-l	Dialog mode

3.25.2. Syntax

{**hso** | **-l**} [#]

3.25.3. Use

This number specifies the polarity of the interrupt signal and allows you to toggle it when LoadICE connects with PromICE:

0 - Interrupt is low asserted, it is raised at this time

1 - Interrupt is high asserted, it is lowered at this time

2 - Interrupt is high asserted and at startup lower-raise-lower interrupt line.

5 - Interrupt is low asserted and at startup raise-lower-raise the interrupt line

A - Interrupt is high asserted. At start up lower it, raise it for 1 second, then lower the interrupt line.

D - Interrupt is low asserted. At startup raise

3.25.4. Default

The signal is low asserted.

3.25.5. Description

This allows the host system to alert the target that LoadICE is talking to PromICE.

3.25.6. Notes

On current units the polarity of the signal is selectable on the back panel of PromICE unit. Specify this statement only when needing to toggle the interrupt line at startup.

3.25.7. Examples

```
hso 1
```

Defines the interrupt to the target to be high asserted.

3.26. image

Specify binary file information for down-load or compare operation.

3.26.1. Command Forms

image	LoadICE.ini file
-i	Command line
image	Dialog mode

3.26.2. Syntax

{**image** | **-i** | **i**} [[*file_name*] [*skip count*] [*id* {:}] [*ROM_offset*] [*data_width*] [[*byte_n*] ...]

3.26.3. Use

file_name (string) Name of binary file to be loaded etc.

skip_count (hex) Any data to be skipped from the beginning of the file. Occasionally there is a 14 byte header that may need to be skipped.

id (integer) A valid PromICE unit ID number (0-255).

ROM_offset (hex) An offset value within PromICE emulated ROM space. The start of PromICE emulated ROM space is assigned to be offset address 0. This offset is used to determine the location within PromICE emulated memory where the first data byte of file '*file_name*' should be assigned.

data_width (integer) The data bus width associated with the width of data objects in file '*file_name*'. Must be an integral multiple of 8 and can not be larger than the data bus width emulated by the total number of daisy chained PromICE units.

byte_n (integer) The nth byte in a data bus that has a width that is an integral multiple of 8 bits. Bytes are assigned to successive PromICE units (0 - 255) based on the selected order of *byte_n* specifications.

3.26.4. Default

Load the file in its entirety starting at 0 in PromICE configuration.

3.26.5. Description

This command allows you to specify a binary data file to be down-loaded to PromICE. Furthermore you can specify where the file should be loaded exactly and if any bytes need to be skipped from the start of the file. The word size and byte order in the file can also be specified.

3.26.6. Notes

When loading multiple binary files you must specify where they are written to in the emulated space or else they may get loaded on top of each other.

You need not specify the ID unless you are using files to be loaded "per ROM". If you have a word size of 16 or larger then specifying the ID says to load the file into a bank where the given ID is the first ID. If such a bank is not found then the file is treated as an 8 bit file for that particular ROM. This inadvertent side affect can be avoided by careful consideration of ID specification.

To load mutiple images in the LoadICE.ini file, add a new file statement for each file to be loaded.

3.26.7. Examples

```
image=myfile.bin 0=10000 16 0 1
```

Load the file 'myfile.bin' at location 0x10000. The file contains 16 bit data and the first byte will be loaded in unit-0 and the second byte in unit-1.

3.27. load

File loading associated with LoadICE dialog mode.

3.27.1. Command Forms

load	LoadICE.ini
-l	Command line (hex file load)
-li	Command line (binary file load)
l	Dialog mode (hex file load)
li	Dialog mode (binary file load)

3.27.2. Syntax

{**load** | **-l** | **l**} [*filename etc.*]

3.27.3. Use

filename	If you wish to load a specific file (in dialog mode only).

3.27.4. Default

Load the pre-specified file list.

3.27.5. Description

'load' allows you to down-load your files on demand. All specified files are processed and down-loaded.

When 'load' appears in the LoadICE.ini file or '-l' on the command line, it implies that LoadICE should down-load the files before entering the dialog mode.

3.27.6. Notes

'dialog' mode is entered only if LoadICE is instructed to do so with 'dialog' in the LoadICE.ini file or '-d' on the command line. When specified in the ini file or the command line, LoadICE down-loads the file before entering the dialog mode.

3.27.7. Examples

```
l
```

Load the files now.

3.28. log

Record all LoadICE command traffic to a log file in real-time.

3.28.1. Command Forms

log	LoadICE.ini file
-log	Command line
log	Dialog mode

3.28.2. Syntax

{**log** | **-log**} {*filename*}
log #

3.28.3. Use

filename	specify the file to use
#	0 - turn logging off
	1 - turn logging on
no args	Toggle on off

3.28.4. Default

No logfile or log is kept.

3.28.5. Description

This command allows the LoadICE/PromICE traffic to be collected in a file for later viewing. This is an alternate to changing the display level in LoadICE (refer to the *display* command). This command can write the log with much less overhead and will not loose the information or the condition of the error. The log can be viewed by using the application 'vlog', included on the LoadICE distribution disk.

3.28.6. Notes

If the log command is used in dialog mode without having previously specified name then the file 'loadice.log' is created to store the log. This is also the default filename for 'vlog' application.

3.28.7. Examples

`log logfile.log`	Log data to file 'logfile.log' from either the LoadICE.ini file or LoadICE dialog mode.

3.29. map

Control the display of the address range being loaded during file processing.

3.29.1. Command Forms

map	LoadICE.ini file
map	Dialog mode

3.29.2. Syntax

{**map**} [#]

3.29.3. Use

#	0 - turn the display of the map information off 1 - turn the display of the map information on

3.29.4. Default

On.

3.29.5. Description

LoadICE will display the range of addresses where data is being loaded. This command lets you turn that feature off.

3.29.6. Notes

When the data file has hex records that have fragmented data then there may be a large number of address ranges displayed. You can use this command to turn that display off.

3.29.7. Examples

```
map 0
```

Don't display the map information.

3.30. move

Copy bytes of memory in PromICE.

3.30.1. Command Forms

move	LoadICE.ini file
m	Dialog mode

3.30.2. Syntax

{**move** | **m**} [[*id* {:}] [*start*] [*end*] [*destination*]]

3.30.3. Use

id	(integer) A valid PromICE unit ID number (0-255).
start	(hex) The starting address of source data block where data is copied from.
end	(hex) The ending address of source data block where data is copied to.
destination	(hex) The starting address of destination location where source data block is to be copied to.

3.30.4. Default

All three parameters must be specified.

3.30.5. Description

Allows you to move data (copy) within PromICE memory.

3.30.6. Notes

Data is up-loaded and then down-loaded to a new location. However, it is uploaded in 4096 bytes chunks, therefore overlapping moves may have unexpected results.

3.30.7. Examples

```
m 100 120 300
```

Move data from 0x100 to 0x120 to 0x300 (to 0x320).

3.31. noaddrerr

Ignore address out of range errors during file loading.

3.31.1. Command Forms

noaddrerr	LoadICE.ini file
-z	Command line
noaddrerr | z	Dialog mode

3.31.2. Syntax

{**noaddrerr** | **-z** | **z**} [1]

3.31.3. Use

1	View records in hex file that have address values out of range.

3.31.4. Default

Stop down-loading data when address error is encountered.

3.31.5. Description

If an attempt is made to load a data byte to an address outside the specified emulated space (i.e. the address is larger than the ROM size) LoadICE will stop processing data and report this error message. You can specify to skip the offending records and continue processing.

3.31.6. Notes

If you have properly mapped your data files, this error is usually caused by initialized data intended for RAM. This option will allow you to skip these records.

3.31.7. Examples

```
noaddrerr 1
```

Skip records that have an address out of emulation space and display them.

3.32. notimer

Turn off the internal timer interrupt in the PromICE.

3.32.1. Command Forms

notimer	LoadICE.ini file
-T	Command line

3.32.2. Syntax

{**notimer** | **-T**}

3.32.3. Use

notimer | -T Turn off the timer.

3.32.4. Default

The timer is on and interrupts every 8.9 milli-seconds

3.32.5. Description

This command is useful when you are using multiple PromICEs with synchronized targets. Since the timer interrupts are asynchronous, the different PromICE units can loose lock-step operation.

3.32.6. Notes

When using the AI circuit on fault tolerant systems, this option helps keep the units synchronized. If the 'reset' command is used the timer is turned on to drive the reset signal. It is then turned off if this option is in effect.

This option will automatically turn off the 'light' option if it is enabled.

3.32.7. Examples

```
notimer
```

Turn off the timer.

3.33. number

(UNIX only) Specify the number of PromICE modules.

3.33.1. Command Forms

number	LoadICE.ini file
-n	Command line

3.33.2. Syntax

{**number** | **-n**} [*total_modules*]

3.33.3. Use

total_modules (integer) The total number of PromICE master and slave modules daisy chained together (0-255).

3.33.4. Default

Number is 1.

3.33.5. Description

This option is needed only for UNIX systems. This allows the LoadICE application to send out enough auto baud characters to ensure that all the units go through the auto baud sequence and some of the characters would return to the host before a read is issued. If a read were to be issued before any characters have arrived, then the read will loop (forever).

3.33.6. Notes

Even though the serial line is opened with option to not wait on the read, (i.e. return if nothing to read) the UNIX manual states that on a communication line the first read will block if no characters are available to be read. This command allows PromICE to get around this problem.

3.33.7. Examples

```
number 3
```

Allows enough auto-baud characters to be transmitted for three units to auto-baud completely.

3.34. output

Specify serial output device for connection with PromICE.

3.34.1. Command Forms

output	LoadICE.ini file
-o	Command line

3.34.2. Syntax

{**output** | **-o**} {*dev_name*} [*address*]

3.34.3. Use

dev_name (string) The standard operating system name for the desired serial port. Paths may be required.

address (hex) On an IBM PC or compatibles, LoadICE will lookup the port address in the BIOS built table in low memory. If you have a nonstandard serial port, then you may explicitly give its address to LoadICE.

3.34.4. Default

COM1	PC
/dev/ttyb	UNIX

3.34.5. Description

This command specifies the serial link for LoadICE that is used to communicate with PromICE. Optionally, the port address may be specified for PC systems.

3.34.6. Notes

When using the AI in transparent mode (ailoc command) this option specifies the link to use for host communication after transparent mode is established.

3.34.7. Examples

```
-o com3
```

Use COM3.

3.35. ppbus

Specifies that the parallel port be used as a down load only bus for PromICE.

3.35.1. Command Forms

ppbus	LoadICE.ini file
-pb	Command line

3.35.2. Syntax

{**ppbus** | **-pb**} {*parallel_device_name*}

3.35.3. Use

parallel_device_name	The LPT number or the device name (UNIX) where PromICE units are attached.

3.35.4. Default

ppbus is not enabled.

3.35.5. Description

This option is used when attaching more than one PromICE unit to the parallel port for fast down-load, use the parallel port bus cable and this command. The serial daisy-chain adapter(s) are also used to support this option.

3.35.6. Notes

LoadICE uses the serial port to connect with and control PromICE units, and the parallel port is used to down-loading code. This makes it possible to have large configurations down-loaded fast. This option also works with the FastPort.

You must also specify the 'number' command when using the Milan FastPort.

3.35.7. Examples

`ppbus lpt1`	Specifies the ppbus port as 'lpt1' in the LoadICE.ini file.
`-pb lpt1`	Specifies the ppbus port as 'lpt1' from the LoadICE command line.

3.36. ppmode

Sets the parallel port's communication mode.

3.36.1. Command Forms

ppmode	LoadICE.ini file
-P	Command line

3.36.2. Syntax

{**ppmode** | **-P**} [0 | 1 | 2]

3.36.3. Use

0	Standard mode, this is the oldest implementation and the slowest, download speed is around 30Kbytes per second
1	Fast mode, this is the newest and much faster, achieving download speed of about 60KBytes per second
2	Turbo mode, it uses the fast mode with no verify and special code in the host to download at the fastest possible of about 90Kbytes per second.

3.36.4. Default

0 - standard mode for units with micro-code version 5 or older.
1 - fast mode for all units with micro-code 6 or greater
By default, the mode is determined at connect time. LoadICE connects to PromICE and automatically determines the parallel port mode by looking at the micro code version.

3.36.5. Description

This option allows you to select the transfer mode suitable for your application. Once you have been able to connect and communicate reliably with other modes you may want to try turbo mode.

3.36.6. Notes

Turbo mode is selectable only on units that support fast mode, i.e. micro-code versions 6 and above.

3.36.7. Examples

`-P 1`	Select fast transfer from LoadICE command line.

3.37. pponly

Specify parallel port in bi-directional mode (Not available to FastPort Users).

3.37.1. Command Forms

pponly	LoadICE.ini file
-q	Command line

3.37.2. Syntax

{**pponly** | **-q**} {*dev_name*} [*address*]

3.37.3. Use

dev_name (string) The standard operating system name for the desired parallel port.

address (hex) Non-standard port address.

3.37.4. Default

LPT1

3.37.5. Description

The host will communicate to the PromICE via the parallel port only.

3.37.6. Notes

There should note be anything between the host parallel port and the PromICE. Certain A/B switch boxes can interfere with the proper operation of the PromICE.

You cannot daisy chain multiple PromICE units over the parallel port alone. You must use the serial daisy chain adapter with the parallel bus cable and add "ppbus" and "output" to your LoadICE.ini file. Refer the installation section for more information about daisy chaining.

PPONLY cannot be used with ethernet adapters like the FastPort. You must use the serial and parallel ports and “ppbus”.

3.37.7. Examples

```
pponly lpt2
```

Use LPT2 as bi-directional parallel port to communicate with PromICE.

3.38. reset

Specify duration of target reset signal (RST).

3.38.1. Command Forms

reset	LoadICE.ini file
-r	Command line

3.38.2. Syntax

{**reset** | **-r**} [*milliseconds*]

3.38.3. Use

milliseconds (integer) The number of milliseconds (0-3000) to activate target reset signal (RST) from PromICE during the next reset event.

3.38.4. Default

500 milliseconds.

3.38.5. Description

Normally, LoadICE and PromICE will operate in auto-reset mode. This means that whenever PromICE is in 'load' mode the target reset signal is asserted and whenever PromICE is in 'emulate' mode then the reset is released (goes tri-state). However, reset may be asserted by the user by specifying this command.

3.38.6. Notes

Specifying 'reset' with a time of '0' will disable auto-reset. The target must be reset by issuing an explicit reset command.

3.38.7. Examples

```
r 500
```

Reset the target with a 500 milli-second pulse.

3.39. resetfp

Resets the FastPort before connecting with PromICE.

3.39.1. Command Forms

resetfp LoadICE.ini file
-rfp Command line

3.39.2. Syntax

{**resetfp** | **-rfp**} [0 | 1]

3.39.3. Use

0 Disables the FastPort reset.

1 Enables the FastPort reset.

3.39.4. Default

The FastPort is not reset.

3.39.5. Description

If the FastPort/PromICE link hangs and LoadICE fails to connect with the FastPort. Invoking LoadICE with this option enabled resolves this situation. LoadICE will connect with the FastPort built-in monitor and reset the unit. After the reset is completed, LoadICE then connects to PromICE unit(s).

3.39.6. Notes

Whenever a reset command is used, it will take approximately 8 or 9 seconds for LoadICE to connect with PromICE, instead of 1 or 2 it takes when no reset is to be done. To break the transparent link if you are using the AI option in transparent mode, the FastPort must be reset every time LoadICE connects. This will be remedied in a future model of the FastPort.

3.39.7. Examples

```
resetfp
```
OR
```
resetfp 1
```
Reset the FastPort from the LoadICE.ini file.

```
-rfp 0
```
Disable FastPort reset from the LoadICE command line.

3.40. restart

Restart the LoadICE/PromICE interface.

3.40.1. Command Forms

restart Dialog mode

3.40.2. Syntax

{**restart**}

3.40.3. Use

restart To restart the link.

3.40.4. Default

None.

3.40.5. Description

This command lets you reestablish the link with PromICE without restarting the LoadICE application.

3.40.6. Notes

If you have reset or cycled power to PromICE then you can reestablish the link by using this command. If LoadICE has timed out waiting for PromICE then this command can be used as well. However, LoadICE will try to recover the link automatically unless the 'noautorecovery' command was specified.

3.40.7. Examples

```
restart
```

Regardless of what state PromICE is in, restart the link with PromICE.

3.41. rom

Specify ROM emulation memory size.

3.41.1. Command Forms

rom	LoadICE.ini file
-r	Command line
R	Dialog mode

3.41.2. Syntax

{**rom** | **-r** | **R**} [*id* {:}] {*size* [k] | *part_number*}

3.41.3. Use

id (integer) A valid PromICE unit ID number (0-255).

size (integer) The size of emulated memory, measured in bytes. The specified number must be an integral power of 2 and cannot be larger than the total memory emulated by the total number of PromICE units.

k Indicates the *size* in kilobytes.

part_number (string) JEDEC standard ROM part number. The number must begin with a 27.

3.41.4. Default

The emulation size is same as the amount of memory in a given PromICE module.

3.41.5. Description

This command allows you to specify the size of ROM you wish to emulate. It must be less than or equal to the amount of memory in PromICE module.

3.41.6. Notes

If your target is wired for a socket larger than the size of ROM you are emulating (see the description of the socket command). This will usually be the case only for 1, 2 and 4 Mb ROMs. A socket wired for 4Mb can be used with a 1, 2 or 4 Mb ROMs without any jumper changes.

All ROM sizes should be 8 bit. If you are using a 16 bit ROM, the ROM size should be half of what the 16 bit size. Since LoadICE treats all ROMs as 8-bit wide. The word size takes care of rest.

3.41.7. Examples

```
rom=27010 | rom=131072 | rom=128k
```

They all specify a 1Mbit ROM to be emulated.

3.42. save

Save PromICE memory contents to a file on the host.

3.42.1. Command Forms

save Dialog mode

3.42.2. Syntax

{**save**} {*file_name*} [[*id* {:}] [*start*] [*end*]]

3.42.3. Use

file_name (string) The name of the host file in which PromICE memory contents are to be saved. Path names may be included.

id (integer) A valid PromICE unit ID number (0-255).

start (hex) Specifies the starting address of that portion. The address must be an offset value within PromICE emulated ROM space. Data bus width multiples (8,16,32 bit) do not need to be taken into account.

end (hex) Specifies the ending address of that portion. The address must be an offset value within PromICE emulated ROM space. Data bus width multiples (8,16,32 bit) do not need to be taken into account.

3.42.4. Default

Save the whole ROM or the current ROM configuration.

3.42.5. Description

The contents of PromICE are uploaded and saved to a binary file.

3.42.6. Examples

```
save newfile 0:100 3fff
```

Save the data from unt-0 from 0x100 to 0x3FFF to a file caled “newfile” on the host.

3.43. search

Search PromICE memory for an ASCII data pattern.

3.43.1. Command Forms

search Dialog mode (also **S**)

3.43.2. Syntax

{**search** | **S**} [[*start*] [*end*]] {*pattern*}

3.43.3. Use

start (hex) Address must be an offset value within PromICE emulated ROM space. The start of PromICE emulated ROM space is assumed to be address 0.

end (hex) Specifies the ending address of the portion. The address must be an offset value within PromICE emulated ROM space. The start of PromICE emulated ROM space is assumed to be address 0.

pattern (string) The data pattern to be searched. Please enclose the string in double quotation marks (" ").

3.43.4. Default

Search all confugured memory.

3.43.5. Description

This command allows you to search PromICE for an ASCII string.

3.43.6. Notes

Start and End addresses are optional. The entire specified range is searched even if there are multiple occurrences of the string. Data bus width multiples (8,16,32 bit) do not need to be taken into account when specifying the address range.

3.43.7. Examples

```
S 0 1000 "Enter new value:"
```

Search from 0x0 to 0x1000 for the string given.

3.44. serial

Display PromICE micro-code serial number.

3.44.1. Command Forms

serial Dialog mode.

3.44.2. Syntax

{**serial**}

3.44.3. Use

serial Display the serial number in the micro-code.

3.44.4. Default

None.

3.44.5. Description

The micro-code has a place for a 32 bit serial number that can uniquely identify a particular unit. This command shows you what that number is.

3.44.6. Notes

This allows PromICE to be bundled with debuggers that will only work on given unit. If you would like specific serial numbers in the units, you must contact Grammar Engine.

3.44.7. Examples

```
serial
```

Report serial number.

3.45. socket

Specify the capacity of the ROM socket on the target.

3.45.1. Command Forms

socket LoadICE.ini file

3.45.2. Syntax

{**socket**} [id{:}] {romsize}

3.45.3. Use

romsize (part#) Specified as 27040 etc.
(K bytes) Specified as 512K.
(decimal number) Specified as 524288.

3.45.4. Default

Same as ROM size.

3.45.5. Description

If your target is wired to accept a range of ROM devices then this statement allows you to specify the largest size the target can address as it is currently configured. If you are emulating a size smaller than the socket size, LoadICE will make adjustments to make sure that both the host (LoadICE) and the target 'see' the same ROM space.

3.45.6. Notes

The 27010, 27020 and the 27040 parts (128k; 256k and 512k bytes) can be plugged into a socket wired for the 27040 part without changing any jumpers. This will cause a problem especially when AI circuit is in use. This command allows LoadICE to use the appropriate mask for unused address lines for such a case.

All socket sizes should be by 8. If you are using a 16 bit ROM, the socket size should be half of what the largest 16 bit size is.

3.45.7. Examples

```
socket=27040
rom=27010
```

Lets you emulate a 1MBit ROM in a socket wired for a 4MBit ROM.

3.46. status

Reports the status of the target system as seen by PromICE.

3.46.1. Command Forms

status	LoadICE.ini file
-st	Command line
status | st	Dialog mode

3.46.2. Syntax

{*status* | *-st*}

3.46.3. Use

status Displays the current status.

3.46.4. Default

N/A. Status is only displayed when requested.

3.46.5. Description

PromICE can report the target power status and whether the target is accessing the ROM being emulated by PromICE. This is a quick way to find out if the target has power and if it is running (at least accessing PromICE).

3.46.6. Notes

Sometimes you may have to execute the command multiple times if the target changed state recently (i.e. after resetting the target). Check the status a few times to see if it is running.

3.46.7. Examples

`status`	Displays the target status from LoadICE dialog mode or in the LoadICE.ini file.
`-st`	Displays the target status from the LoadICE command line.

3.47. stop

Take PromICE out of emulation mode.

3.47.1. Command Forms

stop Dialog mode (also **<ESC>**)

3.47.2. Syntax

{**stop** | **<ESC>**}

3.47.3. Description

Force PromICE units to stop emulating and go into load mode. If auto-reset is active then reset signal is asserted.

3.47.4. Notes

The target will crash if it is executing out of the emulated space, unless the reset line is attached to the target. When the 'stop' command is issued, the buffers on PromICE unit(s) are turned off. If the target is trying to access code in PromICE memory it will see incorrect data. If the reset line is connected, the reset will be held until a 'go' command is issued.

3.47.5. Examples

```
stop
```

The load light should come on so you can execute commands that modify PromICE data.

3.48. test

Test PromICE emulation memory.

3.48.1. Command Forms

test Dialog mode

3.48.2. Syntax

{**test** | **t**} [[*id* {:}] [*pass_count*]]

3.48.3. Use

id (integer) A valid PromICE unit ID number (0-255).

pass_count (integer) The number of times a full memory test is to be performed.

3.48.4. Default

Test unit with ID-0 once.

3.48.5. Description

It runs a simple RAM test within PromICE memory. If the test fails then the offending address is reported.

3.48.6. Notes

This command destroys all data. A failed test may indicate several problems. It can be a damaged buffer interfering with the test, it may be a weak battery causing write failures to the RAM, or it may be a bad RAM chip. If the test doesn't come back (hangs) the problem is probably communication related, not necessarily a PromICE problem. If the test fails or 'hangs' the problem may also be related to host communication interference.

3.48.7. Examples

```
t 3
```

Test unit-3 once.

3.49. version

Report micro code version of the PromICE and the LoadICE version.

3.49.1. Command Forms

version Dialog mode

3.49.2. Syntax

{**version**} [*id* {:}]

3.49.3. Use

id (integer) A valid PromICE unit ID number (0-255).

3.49.4. Default

Display the LoadICE version number and micro-code version of unit ID-0.

3.49.5. Description

This is to check the version of software and micro-code you are running. This information is helpful to our Technical Support Department.

3.49.6. Notes

Please have this information readily available when contacting Technical Support.

3.49.7. Examples

```
version 3
```

Report version of the micro-code in unit-3.

3.50. word

Specify emulated data bus width.

3.50.1. Command Forms

word	LoadICE.ini file
-w	Command line
w	Dialog mode

3.50.2. Syntax

{**word** | **-w** | **w**} [[*data_width*] [[*byte_n*] ...]]

3.50.3. Use

data_width (integer) The data bus width of PromICE emulated memory. It must be an integral multiple of 8 and can not be larger than the data bus width of the total number of daisy chained PromICE units.

byte_n (integer) The nth byte in a data bus that has a width that is an integral multiple of 8 bits. Bytes are assigned to successive PromICE units (0 - 255) based on the selected order of *byte_n* specifications.

3.50.4. Default

Word size is 8 bits.

3.50.5. Description

This command is used for setting the word size configuration of PromICE modules.

3.50.6. Notes

The word size specification is used to dump, edit, fill, and save PromICE memory. On a duplex PromICE (model P2*xxx*), when word size is set to 8 bit, if the master (bottom) unit's memory is overrun the data will go into the slave (top) unit. This is the same as having two ROMs mapped consecutively.

3.50.7. Examples

```
w 16 1 0
```

This specifies that a word size of 16 bits is to be used and that the first byte of data (all the even numbered bytes) will be loaded in unit ID-1 (top or slave unit) and the second byte of data (odd numbered bytes) to be loaded in unit ID-0 (bottom or master unit).

4. TROUBLESHOOTING

4.1. BEFORE YOU BEGIN

WARNING: If proper Electro Static Discharge (ESD) precautions have not been taken you may have damaged buffers. If your PromICE appears to load fine but will not emulate or will emulate to a point in code and hang up reliably, there is a good possibility that the buffers have been damaged.

NOTE: All warranties are void if unit is opened!
CONTACT GRAMMAR ENGINE TECHNICAL SUPPORT FOR ALL TECHNICAL SUPPORT RELATED INFORMATION.

4.2. HOST TO PROMICE

4.2.2. Windows / 95 / NT

Do not run Windows for the duration of the diagnostic procedure. You can try using Windows later. If exiting Windows solves the problem, you may not have the LoadICE window opened exclusively. Add the commands "noverify" to your LoadICE.ini file. You may also add “nomap” and “nocursor” to further speed download. See your Windows documentation for more information about exclusive applications.

4.2.3. LoadICE Version

Check your LoadICE version. Free upgrades can be downloaded from the GEI WEB site at www.gei.com.

4.2.4. Switch box / extension cable / software key

Disconnect any switch boxes, extension cables and/or software keys (hardware security keys, dongles). Connect PromICE to the host via the supplied cables only. Extension cables and switch boxes are not recommended for use with PromICE. Software keys may intercept code intended for PromICE and stop communication. Don't reconnect any of these cables or boxes until the problem is solved. They can always be reconnected once PromICE is communicating.

4.2.5. Port specification

Make sure that you are connected to the right port. If the port has a non-standard address specify it in the command line or LoadICE.ini file as follows:

```
output=com1:3f8
OR
pponly=lpt1:378
```

4.2.6. Baud rate

Make sure you have selected a valid baud rate. PromICE supports 1200, 2400, 4800, 9600, 19200 and 57600 baud. Try using a slower baud rate. If this solves the problem you probably have a noise related problem. Rerouting the cabling away from power supplies, monitors and power cables may help correct this problem.

4.2.7. Serial/parallel communication

If you are using the serial and parallel port together, slow the serial port to 19200 baud. PromICE cannot handle data at the full baud rate in this configuration.

4.2.8. Serial/parallel daisy chain

Make sure the daisy chain modules are connected properly. If the serial connection is the reverse of the parallel connection, PromICE won't communicate (see also "Serial/parallel communication" above). Refer to the "*Hardware Installation*" Section of this manual for more information.

4.2.9. Noise

Move the communications lines away from any power cables, monitors, power supplies, etc.

4.2.10. Display mode

If you are setting the display mode in LoadICE too high the data will overflow the host system during communications. The maximum display mode is "fe".

4.2.11. Conflicting device drivers/TSR's

Check your config.sys and autoexec.bat files for any drivers or TSR's that may conflict with LoadICE. Anything that effects I/O ports can affect LoadICE. If in doubt you can rename your config.sys and/or autoexec.bat files. Reset the system and try LoadICE without any other drivers loaded. If this solves the problem then one of the drivers is

affecting LoadICE communications. Disable one driver at a time until the problem is found. Some memory driver configurations have been known to cause problems.

4.2.12. Poor connection (damaged cable)

Disconnect any extension cables, switch boxes or software keys (dongles) that may be attached to PromICE. Use only the cables supplied with the unit. If the cable is damaged, contact Grammar Engine Sales for a replacement.

4.2.13. Damaged serial/parallel port

If communication is impossible through the serial port, try using the TTY interface as described earlier in this section. If you can connect with the TTY but not through LoadICE, you probably have a bad serial port. Try using another port, preferably on another host system.

4.3. PromICE TO TARGET

NOTE: Emulation problems show up in one of two ways. The target either never emulates, or stops emulating. This section of troubleshooting is split into two separate sections. Many of the problems experienced with emulation may show up in a number of ways. For example: If the target never emulates the problem could be noise related (in the "Never emulates or stops emulating section") or byte order related (in the "Never emulates" section).

If your target doesn't "emulate and never did" see the "Never emulates" section first. If the problem persists, see the "Never emulates or stops emulating" section. Remember to log off the network and exit Windows. If this solves the problem, see the "Windows" and/or the "Network" sections below.

If, after completing the section(s) below, the target still won't emulate, contact Grammar Engine Inc. Technical Support.

4.3.1. Never emulates or stops emulating

4.3.1.3. Mapping

If the file specification in the LoadICE.ini or command line is specified incorrectly, the program will not run or will run to a point and then stop. If you don't have any code that you are loading into RAM space on purpose, make sure that noaddrerr is NOT in your LoadICE.ini or command line. This way, if you are loading out of address range, LoadICE will show the error and location. If the file address and/or the ROM address are specified incorrectly the program won't run or will crash. Make sure you know where your file's starting address is and where in ROM you want it to load.

Example: Assume you have a file called mad.hex that starts at 0x400000 (the hex records in the file say to start loading data at this address). Assume that the file contains 16 bit data and the first byte is emulated by module ID1 (slave unit) and the second by module ID0 (master unit). The specification would be:

```
file=mad.hex 400000=0 16 1 0
```

The "16 1 0" is optional if "word=16 1 0" has been specified.

Example: Assume you have a file called mad.bin that has a 14 byte load header. Assume that the file contains 16 bit data and the first byte is emulated by module ID0 (master) and the second by module ID1 (slave). The specification would be:

```
image=mad.bin 14=0 16 0 1
```

Again, the "16 0 1" are optional if the "word=16 0 1" has already been specified. You can always be safe by specifying all the parameters.

4.3.1.4. Emulation size

Emulate the largest size that your target IS WIRED FOR. If the ROM you are emulating is a 27512, but your target's socket can handle a 27020, emulate the 27020. If you don't know for sure whether your target can emulate larger sizes or not, start by emulating the largest size ROM that your PromICE can emulate. Work your way down from there until you find the correct emulation size.

4.3.1.5. Noise

Check the cabling between PromICE and the target. Make sure that the cable isn't near any power supplies, monitors or cables. If your target is sensitive to noise or is noisy itself, it may not emulate (or at least not reliably). Make sure that none of your cables are anywhere near a source

of noise (i.e. monitors, power supplies, power cables, etc.). Keep the cables as short as is possible and route them as far away as possible from noise sources. If the target itself is noisy it may become necessary to obtain shielded cables. Contact Grammar Engine if this becomes necessary. See also "Parasitic power" below.

4.3.1.6. Parasitic power

PromICE unit is designed to run either parasitically (power comes from target ROM socket) or externally. PromICE draws about 100 mA per board inside. If you have a duplex (2 PromICE's in a box) with an AI interface, the unit will draw approximately 300 mA from your target if you are running PromICE parasitically. This may be enough power drain on the target system to make it fail. In addition, whether powered parasitically or externally, if the target voltage drops below a certain voltage threshold PromICE will go into protected mode. During this time, PromICE will stop emulation and turn off the buffers to protect its memory. For Revision 3 units the voltage is approximately 3.3 volts. Although the voltage may be high enough at startup, there may be a function that enables a peripheral that takes power away from the target. This drain may drop the target voltage below the threshold. To resolve this problem, the voltage sense can be forced to 5v:

Remove power from the target and PromICE. Place jumpers on the "EXT" and "ROM" positions. Place another jumper on the 28 or 32 pins depending on the adapter you are using. Only the 28 pin DIP cable will use the "28" pin jumper. All other PLCC adapters and 40 pin DIP adapters will use the "32" jumper. Connect the reset line on PromICE to the target reset line. If you have a reset button on the target, you can use it instead of the reset line on PromICE.

WARNING: Do not attempt to use this configuration if you are using the GEI 5V to 3V adapter board for 3V targets. You will not be able to power down your target to initiate a reset. In this configuration, powering down the target without first powering down PromICE can damage the unit.

Power up the target, THEN PromICE. Rerun LoadICE.

If you are using parasitic power (PromICE gets its power from the ROM socket) and the target stops emulating at certain points, PromICE is probably going into protected mode. PromICE stops emulation and turns off the buffers that go to the target when the target voltage drops below a certain voltage. If your target locks up while performing a certain function

the target voltage may be dropping below this threshold. Try powering PromICE externally. If this still doesn't help you may want to try to force the voltage sense.

4.3.1.6. Address line floating or pulled low

PromICE units have address lines A0-A19 connected to the 32 pin DIP cable, except for the following:

For units one megabit or less i.e. Px010 or less A17 and A18 are NOT connected to PromICE.

For two megabit units Px020, A17 is connected but A18 is NOT.

For units four megabit and larger i.e. Px040 or larger both A17 and A18 are connected.

The reason for this variation is that a 1, 2 or 4 megabit ROMs can be plugged into a socket wired for 4 megabit, without any jumpers or changes. This means that the target can drive A17 and A18 even though it may be addressing a 1 megabit ROM. To accommodate this situation the **socket** command is used to tell LoadICE to properly configure memory pointers so that both LoadICE and the target see the same memory from both sides.

4.3.2. Never emulates

4.3.2.1. Incorrect byte order

If you are emulating more than one ROM and/or are in a 16 bit or larger configuration, check the byte order in the LoadICE.ini file or your command line. If the byte order is reversed the code is going into the wrong ROMs. In duplex PromICE's, the bottom unit is ID 0 (usually the even bit) and the top is ID 1 (usually the odd bit).

4.3.2.2. Emulation size

Check your emulation size. Make sure the socket statement is set correctly and that the ROM size is not set larger than the socket size. If this is set incorrectly, PromICE will not emulate. If you are not sure what your target is wired for, set the socket size to emulate a 27080 and then work your way down.

4.4. If All Else Has Failed

If the problem cannot be solved from the information supplied in this manual then you should contact Grammar Engine Technical Support. Support is available via email, phone and fax.

If you choose to use the phone option be prepared with the following information:

PromICE Model and Serial number (Model number starts with a "P")
Contents of the LoadICE.ini file or command line
Exact error message being encountered
Target information:
  CPU type
  Clock rate
  ROM access time
  ROM part number
All switch and/or jumper settings on PromICE
Host type and type of connection to PromICE (serial, parallel or both)

Be at your computer with PromICE and target configured and ready to go. This will help us get you up and running faster.

Before calling Technical Support, check the Grammar Engine WEB site to see if the answer to your question is there. The fastest way to obtain technial support is through email. Email requests are received real-time and answered immediately.

Our numbers are:

Technical Support	(614) 899-7878
Fax	(614) 899-7888
INTERNET	support@gei.com

4.5. EMERGENCY REPAIRS

WARNING: **OPENING PromICE CASE WILL VOID ALL WARRANTIES. ONCE THE CASE HAS BEEN OPENED THE UNIT IS NO LONGER UNDER WARRANTY.**

You may be in a situation where you must repair PromICE and continue with your work. In such cases the instructions provided here may help. Warning: This will definitely void any warranty and you assume the full risk of applying any fixes here. Any mistakes may further damage the unit or your target system. If you are very careful and take full precautions against static damage, you may proceed. Grammar Engine will not be liable for any damages incurred when you follow these instructions.

Replace parts only with identical replacements. For Example: An HCT244 and ALS244 are not the same. If you replace an ALS with an HCT PromICE may not emulate properly.

4.5.1. Replacing ROM Interface Buffers

If the PromICE is not emulating or LoadICE is reporting Memory Size Zero errors when you connect with PromICE, then you may have damaged buffers on the ROM interface. One or more damaged buffers can keep one or more address lines from switching and cause the above mentioned problems.

To replace the buffers do the following:

1. Open PromICE unit by removing the flat head screw on the back panel that is attached to a heat-sink. Use #1 Phillips screw driver and remove the two screws from the bottom portion of PromICE box.

2. Slide out the circuit boards and the panel. Keep them all together as you are taking them out or they may bind in the case. Remove the panels and gently pry apart the circuit boards. If you have duplex unit, or AI option installed.

3. The address input buffers are marked U11, U12 and U13 and are located behind the ROM cable header. These are 20 pin socketed devices. These are generally 74ALS244 chips. They can be replaced by equivalent known good chips. If you do not have enough chips then replace them one at a time, until the defective chip is replaced.

4. The data buffer is marked U14 and is a 74ALS245. Replace it with a similar chip.

5. The interface chips are marked the same on both the master board (the one with main PromICE circuit on it) and the slave board (mounted on the master board).

6. After the chips are replaced, reassemble the boards carefully lining up the 44 pin headers on both sides of the board. You may refrain putting the unit in the box until the problem is completely fixed.

7. For complete reassembly, hold the panels in position with the boards and gently slide the whole assembly into the bottom portion of the box. Attach the heat sink back to the back panel with the flat head screw. Slide the top half of the box and close it with the Phillips screws through the bottom half of the box.

4.5.2. Replacing the Parallel Port Buffers

If you are having problems with the parallel port, and it can not be fixed by any of the things mentioned in regular trouble shooting section, then the parallel port buffers may be damaged.

Once again, when the unit is open, check the two buffers right behind the parallel port connector. These are also 20-pin socketed devices. They are 74LS244 and 74ALS244 chips. Replace them with equivalent chips.

4.5.3. Replacing Other Parts

Theoretically you can replace all the parts but the PAL and microcontroller. If the problem can not be solved by replacing the buffers, then the only other parts that are easy to replace are the ones in buffers, or some of the discretes. Here is a list of parts that you may replace:

1. The 8-pin Op Amp part (the only 8-pin IC on board, and in socket). The unit will not go into emulation if the target does not have enough power. This Op Amp will force emulation to shut down if it thinks the target power is going down.

2. The 7805 regulator. If PromICE will not maintain 5VDC power, the regulator may have gone bad. Look for discoloration from excessive heat. Desolder the regulator, solder a new one, reattach the heat sink.

3. The 74HCT373. This chip is soldered on to the board. If it has failed, it will not let PromICE correctly address the memory. It will have to be desoldered and replaced. Do not replace it with an HC part, it must be HCT or LS or ALS etc.

4. The 74LS125. This is a 14 pin socketed device. This chip drives the target RESET and INTERRUPT lines on the back pin rst(- +) and int(- +). If these signals are not functioning properly, then this chip can be replaced with an equivalent part. An HCT or ALS part will suffice and even an HC may be used.

5. The last chip is the LT1081, a 5VDC only RS232 interface device. If the problem seems to be RS232 then this chip can be desoldered and replaced with an equivalent chip.

   Note: Both the Dallas DS1221 and Linear Technology LT1081 (or LT1281) have pin compatible parts made by BenchMark and Maxim, respectively. Carefully check the part pinouts before substituting.

4.6. TECHNICAL SUPPORT

PRIORITY FAX

To: **GRAMMAR ENGINE TECHNICAL SUPPORT**
Fax: (614)899-7888
From: _____
Company: _____
Fax: ( ) _____
Voice: ( ) _____
Model # **P**_____ (Please include complete number)
Serial # _____ (Below model number)
(*model and serial numbers are located on the front or bottom of the unit*)

Jumper settings on PromICE: **EXT ROM 32 28 24**
LoadICE version number: _____ (Run LoadICE or type "v" in dialog mode to get full version #)

Contents of the LoadICE.ini file:

_____
_____
_____
_____
_____
_____
_____
_____

Target Information:

Target CPU: _____
Clock Rate: _____
ROM access time: _____
ROM part # _____
ROM wait states: _____
Bus width: **8 16 24 32**

Command line arguments: _____
Error messages: _____

Host type and speed: _____ Switchbox: **YES / NO**
Network: **YES NO** Windows: **YES NO**
Connection to PromICE**: Serial / Parallel / Both** Extension cables: **YES / NO**
Debugger used: _____

Additional Information:

_____
_____
_____
_____
_____

4.7. RMA INFORMATION

SHIP TO: Grammar Engine Inc.
921 Eastwind Dr.
Suite 122
Westerville, OH 43081

ATTN. RMA# _____ (Contact Grammar Engine Technical Support for RMA)

PromICE **MODEL #** (On front panel or bottom of unit)**:**
P_____

PromICE **SERIAL NUMBER** (Under model number)**:**
_____

Purchase Order number
(If out of warranty and/or for special shipping)**:** _____

Return Shipping instructions:
_____

Return Address: _____
_____
_____
_____
_____
_____

Contact Name: _____
Contact Phone # _____

Reason for Return: _____ **Upgrade** _____ **Repair**

Problem Description:
_____
_____
_____
_____

5. ERROR MESSAGES

Normal messages from LoadICE inform you of PromICE's status and show you information you have requested. The following messages, however, are error messages that are displayed when errors are encountered. Unless you are in dialog mode, errors will terminate LoadICE. In dialog mode the control is returned to you for further command input.

Error messages are always identified as such, and auxiliary data is also displayed to give you a little more information about errors. If the global variable 'errno' is set, then a system error message is also displayed. It may not always be meaningful, but it usually means that some system call that LoadICE has issued has failed, producing the error message. LoadICE displays the last user input it was processing when the error occurred. In some cases it may not apply to the error condition. In some cases the error may have occurred in processing the input after part of it had been successfully processed. The negative error numbers are actual failure codes returned from the emulation units. The positive messages are generated by LoadICE. Following, is the list of errors you may get from LoadICE:

5.1 (-6) Interface is not available or not active

Causes:
Relates to programming the AI. Either the AI option is not installed or it is not activated by the software, or the PI option is not activated before use.
Solution:
Check your configuration.
Contact Grammar Engine Sales for an upgrade.

5.2 (-5) Interface is busy

Cause:
AI interface can not accept more data. The interface is busy with previous command or is busy processing target data.
Solution:
Check your software.

5.3. (-4) Timer expired while waiting

Cause:
Command timed out on the AI or PI interface either AI or PI operation has timed out.
Solution:
Check your software.

5.4(-3) DataOverflow - lost host data

Cause:
The target is not servicing the interface, it may be hung. A write to the AI or the PI has failed due to previous write not completing.
Solution:
Rerun LoadICE.

5.5(-2) No data available from the target

Cause:
A read operation from the AI failed
Solution:
Retry the read attempt.

Cause:
Interface is hung or target has crashed. A read command was sent and no data was available from the target.
Solution:
Run LoadICE to reset the link.

5.6(-1) Feature not implemented

Cause:
You are trying to use an unimplemented feature.
Solution:
Contact Grammar Engine Technical Support for assistance.

5.7. (1) LoadICE parser internal error #1

Cause:
This error should only occure if LadICE has been modified.
Solution:
Correct the input script and retry tho command.

5.8. (2) Illegal command

Cause:
Spelling error
Solution:
Check spelling and retry

Cause:
Capitalization
Solution:

All commands are lower case unless otherwise noted.

5.9. (3) LoadICE parser internal error #2

Cause:
Internal error.
Solution:
Contact Grammar Engine Technical Support for assistance.

5.10. (4) Too many arguments supplied

Cause:
The parser found too many arguments to the command.
Solution:
Check command syntax and retry the command.

5.11 (5) Expected argument not supplied

Cause:
Required input was left out.
Solution:
Check the command syntax.

5.12 (6) Filename error

Cause:
Filename doesn't exist in directory or specified path.
Solution:
Check the path and filename for errors.

5.13 (7) Invalid baud-rate

Cause:
Invalid baud rate specified
Solution:
Specify a supported baud rate (i.e.,1200,2400, 4800, 9600, 19200, or 57600)

5.14 (8) Invalid ROM size

Cause:
ROM size specified is not valid
Solution:
Specify a generic part number (i.e. 27512), the size in Kbytes (i.e. 64k) or the actual decimal number (i.e. 65536).

5.15 (9) Invalid word size

Cause:
Word size is not a multiple of 8.
Solution:
Specify a valid word size (i.e. 8, 16, 32, 64...)

5.16 (10) Invalid ID list

Cause:
Attempting to load into more units than exist.
Solution:
Check your configuration for an ID that is larger than the number of physical PromICE units. Check file specification(s) and/or id list for incorrect id listing. Refer to PromICE configuration section for more information.

Cause:
Word size too large.
Solution:
Divide the word size by 8. If this number is larger than the number of PromICE units, you either need more units or he wordsize is incorrect. See the Quick Start section *Software Configuration"* for more information.

5.17 (11) Open failed

Cause:
File name not specified correctly.
Solution:
Make sure the file and it's extension are correctly spelled and in lower case.

Cause:
File not in path.
Solution:
Either the file should be in the same directory as LoadICE or the file path should be specified.

5.18 (12) Unable to skip file data

Cause:
Skipping more data than is in file.
Solution:
Check your skip count. It should not be larger than your file size.

5.19 (13) Device I/O error

Cause:
This can happen with either the file I/O or the I/O on the link to the PromICE.
Solution:
Check the system message for a more detailed description. If you are entering data from the command line or in dialog mode, try creating a LoadICE.ini file. You may have stated the commands in an incorrect order.

5.20 (14) Bad port name

Cause:
Port doesn't exist.
Solution:
Check the port name.

Cause:
Non-standard port address.
Solution:
Check the address of the host port. See the Quick Start section "*Software Configuration"* for more information.

5.21 (15) End-o-File

Cause:
Corrupt record in hex file.
Solution:
Remake the file.

5.22 (16) Bad parameters to picmd()

Cause:
Count field larger than allowed.
Solution:
Should only occur if LoadICE has been modified. Recopy LoadICE from the distribution disks.

5.23 (17) Communication error

Cause:
LoadICE could not establish a link with PromICE or the link failed.
Solution:
Check your connection and try to reconnect. If serial connection isn't working try using the parallel connection and vica versa. Check the cable connections at both the host and PromICE sides.

5.24 (18) Name too long

Cause:
File or device name too long.
Solution:
Shorten the name to no more than 128 characters long.

5.25 (19) Timed out waiting for response

Cause:
Noisy connection.
Solution:
Move cables away from monitors and/or power supplies.

Cause:
Using DOS version of LoadICE under Winodws 95 or NT.
Solution:
Download the correct version of LoadICE for your operating system.

5.26 (20) FEATURE NOT IMPLEMENTED YET!

Cause:
Internal error.
Solution:
Contact Grammar Engine Technical Support for assistance.

5.27 (21) Verify failed

Cause:
Noisy connection with the PromICE.
Solution:
Rerun LoadICE. Run a test on the PromICE memory (see “Test” in the command reference). If the test fails or if the problem persists contact Grammar Engine Technical Support for assistance.

5.28 (22) Invalid unit ID

Cause:
ID specified is larger than the total number of PromICE units.
Solution:
Check file and word specifications for an invalid ID. Refer to the installation section for more information.

5.29 (23) Address out of range

Cause:
Incorrect file offset.
Solution:
Correct the file offset. See “file” in the COMMAND REFERECNE for more information on file offsets.

Cause:
ROM size is too small.
Solution:
Set the ROM size to a larger part.

Cause:
Some code being loaded is intended for RAM.
Solution:
Add “*noaddrerr*” to ignore addresses outside of ROM space.

5.30 (24) LoadICE internal error #3

Cause:
Internal error.
Solution:
Recopy LoadICE from the distribution disks.

5.31 (25) Bad arguments

Cause:
A command has invalid input.
Solution:
This usually occurs when ailoc and pponly are included in the LoadICE.ini file. Add “*output=<com>*” (where <com> is the serial port the PromICE is connected) as the first line in the LoadICE.ini file. See the COMMAND REFERENCE section of this manual to verify syntax for a particular command.

5.32 (26) Bad checksum in record

Cause:
File has been manually edited (patched).
Solution:
Use the nochecksum or *-x* option to process and load the edited records.

5.33 (27) Feature not supported on this unit

Cause:
Unit does not have feature installed.
Solution:
Contact Grammar Engine Sales for an upgrade.

5.34 (28) Bad argument for driver call

Cause:
Corrupt LoadICE application.
Solution:
Recopy LoadICE from the distribution disks.

5.35 (29) Bad data in the hex record

Cause:
The file processor found data that it doesn't know how to handle.
Solution:
Recompile the hex file and retry. The error string contains information about the error. Correct any problems and retry.

5.36 (30) Unit is LOCKED

Cause:
Internal error.
Solution:
Contact Grammar Engine Technical Support for assistance.

5.37 (31) Not enough units to emulate the word-size

Cause:
An attempt was made to to emulate a word size larger than the number of PromICE units in the configuration.
Solution:
Select a smaller word size or contact Grammar Engine Sales for an upgrade. Divide the word size by 8. There should be the same number of PromICE units as the number you come up with. PromICE models beginning with "P2" count as two PromICE units and models beginning with a "P1" count as one unit.

5.38 (32) Memory Size Zero

Cause:
The PromICE memory controller hasn't charged yet.
Solution:
The PromICE contains a super capacitor that will take a few seconds to charge. Once charged, it will hold for at least several hours and at most a week (depending on the PromICE memory size and speed). If the error still occurs after 30 seconds, check the PromICE power supply. If the Replace the power supply if necessary.

Cause:
The target's power supply isn't powerful enough to supply the PromICE (PromICE jumper in "ROM" position only).
Solution:
Power the PromICE Externally.

Cause:
Bad or incorrect power supply.
Solution:
If the error still occurs after 30 seconds, check the PromICE power supply. If the Replace the power supply if necessary.

5.39 (33) Operation terminated by user

Cause:
A command was terminated during processing
Solution:
Retry the command.

5.40 (34) Data over-run

Cause:
Interference from a network.
Solution:
Move all files to the local drive.

5.41 (35) Key NOT assigned

Cause:
You pressed a function key that has no assigned value.
Solution:
Assign or press a different key see 'afn' or 'fn' command to assign a key.

5.42 (36) Can't do while Emulating!

Cause:
A command was issued that requires the PromICE to be in load mode.
Solution:
Type “*stop*” or prress the escape key (The escape key in DOS/Windows toggles the PromICE between emulation and load modes). When emulation is stopped, the target will crash and needs to be reset after the "go" command is issued. If the reset signal is connected to the PromICE, the target will be held in reset until emulation is turned back on.

5.43 (37) Not emulating! - do a 'go' first

Cause:
A command was issued that requires the PromICE to be in emulation mode.
Solution:
Type “*go*” or prress the escape key (The escape key in DOS/Windows toggles the PromICE between emulation and load modes). If the reset line is not connected from PromICE to the target, the target will have to be reset after the “*go*" is issued.

5.44 (38) No operation!

Cause:
The operation was canceled by the user.
Solution:
Retry the operation

5.45 (39) No link with units...

Cause:

Internal error.
Solution:
Contact Grammar Engine Technical Support for assistance.

5.46 (40) Must have AI Rev3.1...

Cause:
Command requires a newer Analysis Interface option.
Solution:
Contact Grammar Engine Sales for an upgrade.

5.47 (41) Must include 'word=' statement...

Cause:
You are specifying a command that requires the “word” statemetn to be specified.
Solution:
Place the "word" statement into your LoadICE.ini file. Refer to the SOFTWARE CONFIGURATION for more information.

6. Analysis Interface Configuration

6.1. Introduction

You must have a PromICE with "AI" as part of the model number. If you do not have this option and want to debug through the ROM socket, contact your Grammar Engine Sales Representative for an upgrade.

6.2. Description

The Analysis Interface (Referred to as the "AI") option implements a virtual serial channel through the ROM socket referred to as the ROMART. The AI is a separate board installed in PromICE and it operates through the master module (lower of the two or the only) ROM emulation pod.

6.3. How The AI Works

The AI acts like a serial port on a target system. The AI's address is mapped into the ROM address space. Communication is done entirely by reading the interface. This means that you don't have to be able to write into ROM space to communicate from the target to the host. The AI communications area takes up only four consecutive address bytes (within the master module) for communication.

With minimal code on the target system you can communicate to and from the host system to your target with no target resources taken up. PromICE AI configuration is also able to do interrupt driven communications. This is done by connecting a line from the **int-** (low asserted) or **int+** (high asserted) pins to an interrupt on your target. A target monitor can then be configured for interrupt driven communications.

By connecting the **wrt** (write line) from the back of PromICE to the write line on a target system, you can do write cycles into ROM space (assuming your target allows writes into ROM). This is strictly to set break-points in ROM or otherwise load the code in ROM space via the debug monitor.

6.4. How Debuggers Work With the AI

A debugger has two components, the front-end runs on your host and manages the target and debugging with the help of a monitor. This monitor generally communicates with the front-end via some connection between the target and the host system. Usually this is a serial channel connected to a

COM port on the host. The AI removes the requirement of having a spare serial or other I/O channel on the target for debugging use.

Adding a PromICE and AI to your target is like adding a serial port to your target. This serial port is mapped into your ROM address range and works by reading the interface only. In other words, you don't have to write into your ROM space to communicate bidirectionally with your host system. Other than that, PromICE AI operates just like a UART on your target system.

The only modification to the target monitor is made to support the AI virtual UART instead of some other device. All third party debugging packages support PromICE/AI system as an option.

6.5. Getting Started With the AI

In order to get your debugging configuration quickly, you should use the following procedure:

Step 1: Get the target system emulating with PromICE

At this point the debugger/monitor should not be involved. Follow the instructions in the beginning of the manual and get PromICE emulating with some known working code. Once the target is running with PromICE connected, you can be certain that everything in your PromICE configuration (PromICE jumper settings, adapter configuration, etc) and LoadICE.ini file are correct.

At this point, the only changes you will make to your LoadICE.ini file will be to the "file" (or "image"). You will also add the "ailoc" statement and possibly one or more of the other AI commands. You should NOT change your "socket", "rom", or "word" statements. If your target is emulating, these are correct.

Step 2: Configure the debugger monitor to work with your target's serial port (if possible)

This may be impossible for you to accomplish if you don't have a serial port or if your serial port communications is part of what you are trying to debug.

The reason for suggesting this is so that you can verify your monitor configuration without the AI communication routines in the loop. At this point you may choose to connect the write pin on PromICE to the write line on your target system. This will allow you to download your code into PromICE ROM space from the debugger, set breakpoints in ROM and singlestep.

If you will be using interrupt driven communications you may also chose to connect the **int-** (low asserted) or **int+** (high asserted) interrupt pin on PromICE to an interrupt on your target. Most debuggers suggest you use polled mode communications before trying interrupt communications. Most of the debugger monitors require you to supply your own routines to handle interrupt driven communications. By using polled communications, you can verify that your monitor configuration (everything except the interrupt communications) is working. If the target stops communicating when you move to interrupt driven communications, it is obviously the interrupt driver that is the problem and not the rest of the configuration.

At this point, your LoadICE.ini and monitor configuration should be correct.

Step 3: Configure the debugger monitor to work with the AI

Finally, add the AI specific modifications to the configuration and add the "ailoc" statement to your LoadICE.ini file. Again, start with polled mode communication. If you want to use interrupt driven communications, you can configure for that once you have the monitor working in polled mode. At this point you will be debugging through the ROM socket.

The procedure for configuring a debugger monitor to work with PromICE varies somewhat from debugger to debugger. However, the modifications necessary to work with PromICE AI are a very small part of the overall monitor configuration. We have listed several sample configurations for some of the debuggers we have used in house. The next few sections will describe the basics of monitor based debugging; How it works and how PromICE AI fits in with the solution.

6.6. Download Options

There are two ways to get your application from your host system to the target system for debugging:

1. Download the debugger monitor to PromICE and load your application from the debugger either into target RAM or PromICE (ROM) space.

   The advantage to setting up your system this way is that it is the fastest way to set up.

The disadvantage is that although you can download the debug monitor over the parallel port (very fast) you still have to download your application via the serial port. The other disadvantage is that if you are loading your application into PromICE ROM space (write line is connected), you must be very careful not to load your application over the monitor. If this occurs your target system will crash during the application download. The only way to recover is to reload the monitor using LoadICE and restart the system.

2. Download the debugger monitor and application to PromICE using LoadICE and then do a symbol load from the debugger application.

   The advantage to this configuration is that you can download the debugger monitor and your application to PromICE at the same time. You can download over either the serial or parallel port. This is the fastest way to get the code to your target system. Once you have the target running with the monitor and application on the target, you can do a symbol download in the debugger and your ready to start debugging. In addition, by loading and comparing the monitor and application files, you can see if one is being loaded over the other.

The debugger setup is the same for both options. Procedurally, the configuration is the same.

7. AI Command Reference

7.1. OVERVIEW

7.1.5. Debug setup commands

The most common debug setup is done by the **ailoc** command. This command will let you program PromICE (equipped with the AI option) to act as a transparent link between the host based debugger and the target resident monitor. This allows you to debug your target code without requiring additional hookups for communicating with the target system.

The rest of the commands condition and modify how this link operates and allow the debugger to interrupt or reset the target via the DTR or INIT line or support the interface when the target system can only make burst accesses to the ROM.

aicontrol	Controls the AI's access timing
aifast	Selects higher speed transparent communications
ailoc	Specify the address of AiCOM virtual channel
ainorci	Disables per-character receive interrupts
aireset	Allows host debugger to reset the AI interface
aitint	Allows the host debugger to directly interrupt the target
aitreset	Allows debugger to directly reset the target
burst	Allows support for burst mode access to ROM interface

7.1. aicontrol

Selects AI timing characteristics

7.1.1. Command Forms

aicontrol LoadICE.ini file
-aic Command line
aicontrol Dialog mode

7.1.2. Syntax

{**aicontrol** | **-aic**} [0 | 1 | 2 | 3]

7.1.3. Use

`0` Standard timing derived from target system.

`1` Insert a 50ns delay in clocking target to host data.

`2` 1 above plus use delay on target read of host data.

`3` Both 1 & 2 plus use 50ns write for ROM patch operation.

7.1.4. Default

No timing modifications.

7.1.5. Description

The AI is clocked by the accesses from the target system. Clocking is accomplished by combining *chip_enable*, *output_enable* and address lines, there is a possibility of false clocks from overlap between these signals. Under ideal conditions this would never happen. However, due to decoding and buffering, these signals can be skewed by the time the logic on the AI board decodes them for its own purposes. This command allows a one-shot timer to be inserted, allowing the signals to be "de-skewed". The one-shot sets up a 50ns pulse that delays the sampling of input signals. This can result in more reliable functioning of the AI circuit. The circuit affected is the one that receives target data for the host. The circuit that responds to the target read of host data can be programmed to use this delay for reliable clearing of data available latch.

7.1.6. Notes

On target's where the accesses cycle is about 50ns or less, this command may cause the AI to fail completely.

7.1.7. Examples

`aicontrol 1`	Enables one-shot delay to help "de-skew" the control signals.

7.2. aifast

Selects non-buffered transparent AI communications.

7.2.1. Command Forms

aifast LoadICE.ini file
-aif Command line
aifast Dialog mode

7.2.2. Syntax

{**aifast** | **-aif**} [0 | 1]

7.2.3. Use

0 Disable non-buffered communications

1 Enable non-buffered communications

7.2.4. Default

Use the receive buffered method. Up to 40 bytes can be buffered before data overflows and is lost.

7.2.5. Description

This command allows you to turn off I/O buffering. When buffering is turned off, communication speed increased at the cost of reliable communications.

7.2.6. Notes

If you expect the target to be busy servicing an interrupt or expecting bursty traffic from the host then the buffered method (default) is better. If you want overall faster throughput and buffered I/O works, then try this option.

7.2.7. Examples

```
aifast
```
or
`aifast 1` Enable non-buffered communication from LoadICE.ini file.

`-aif 0` Disable non-buffered communication from the LoadICE command line.

7.3. ailoc

Sets up a communication channel between the host and the target.

7.3.1. Command Forms

ailoc LoadICE.ini file
-a Command line

7.3.2. Syntax

{**ailoc** | **-a** } [*id* {:}] {*address*} {*baud*} [*break_char*] [*int_count*]

7.3.3. Use

id (integer) A valid PromICE unit ID number (0-255), when specified it must be followed by a ':'. This number is the ID of the master module of PromICE unit with the AI option. If you are using multiple units for 16 or 32 bit emulation, etc. the other units are automatically programmed to be in pass through mode.

address (hex) The address of the virtual UART as seen by the target system, specified as an offset from the beginning of ROM space. Since the AI interface occupies four consecutive bytes, this address must be on a quad boundary (i.e. lowest two bits are always zero) within the single ROM through which the interface is accessed. This means that for a 16 bit system the address of the AI interface must align with an octal boundary (lowest three bits are always zero) and for a 32 bit system on a 16 byte boundary (lowest nibble is always zero).

baud (integer) A valid baud rate (1200, 2400, 4800, 9600, 19200, 57600) at which the host based debugger will communicate with the target via PromICE. A baud rate of zero (0) will invoke transparent parallel port mode.

break_char (hex) A character used to break PromICE out of transparent mode. This is used with ASCII host-target communication protocols. Use -1 to specify no break character when the next argument must be supplied.

int_count (integer) A cumulative number of host interrupts to ignore before PromICE breaks out of transparent mode. Used

with binary host-target communication protocols. Some debuggers will toggle DTR line on startup and thus cause PromICE to restart and break the link, this option allows you to override that.

7.3.4. Default

The default *id* is 0. The *address* and *baud-rate* must be specified. The *address* is normalized to map to master ROM module according to word size specification (see word command). The default for the*break-character* is no break character or "-1" (i.e. binary transparency) and the default for *int_count* is to ignore all interrupts from the host. So for most cases you need only specify the address and the baud rate.

7.3.5. Description

ailoc is used for putting the AI link into transparent mode as the LoadICE application completes. It is used after down-loading a debugger monitor and before starting the debugger's front end (DFE). The address specified is where the target will have the interface mapped. The address is normalized automatically by LoadICE for mapping to the master ROM module. The baud-rate programs the serial link baud rate to the host. The parallel port may be used for communication with the host, PromICE bi-directional parallel port protocol must be used.

Once the AI link is in transparent mode, the DFE in the host can communicate with the ROM monitor in the target as if they were connected with a standard serial link.

7.3.6. Notes

In order to run LoadICE again, the transparent link must be broken. LoadICE does that by toggling the DTR or INIT signal (on parallel port) 4 times in 5 seconds. This allows PromICE to recognize the LoadICE attempt to break the link. However, UNIX systems the DTR can not be toggled fast enough (security precaution for letting the modems hang-up the connection), so the time period must be increased to almost 30 seconds. This can be avoided by removing the toggle code in piunix.c and power-cycling PromICE unit or pressing the reset switch on the back of PromICE.

7.3.7. Examples

```
ailoc 8 19200
```

This example assumes that the AI link is mapped in the target space at location (offset from beginning of ROM space) 0x8 in ROM space and the serial baud-rate for communication with the host is 19200 baud.

7.4. ainorci

Disables per-character receive interrupts in AI transparent mode.

7.4.1. Command Forms

ainorci LoadICE.ini file
-ainri Command line
ainorci Dialog mode

7.4.2. Syntax

{**ainorci** | **-ainri**}

7.4.3. Use

N/A

7.4.4. Default

Both the '**int-** and **int+**' lines on the back of PromICE are asserted whenever host data is available (HDA bit set in AI status register).

7.4.5. Description

Normally the AI receiver will act like an interrupt driven device, i.e. every time a character is available from the host (HDA set), the **int**- and **int**+ lines are asserted. The target clears the interrupt condition by reading the host data. However, this command can be used to turn off this interrupting. Then the target must poll the status for HDA . Debuggers can use this to disable receive interrupts from PromICE/AI on the target system and then use the DTR line to directly cause the interrupt when needed, such as to break a run away application.

7.4.6. Notes

Use this command when the per character interrupt overhead is not desired or not useful. The debugger then must control the reset directly via DTR/INIT and 'aitint' command.

7.4.7. Examples

`-ainri` Disable interrupts from the LoadICE command line.

7.5. aireset

Allows a host debugger to reset the AI interface.

7.5.1. Command Forms

aireset LoadICE.ini file
-aire Command line
aireset Dialog mode

7.5.2. Syntax

{**aireset** | **-aire**} [0 | 1]

7.5.3. Use

0 Disable the feature

1 Enable the feature

7.5.4. Default

aireset 0

7.5.5. Description

This command allows the host driven DTR or the INIT signal to reset the AI interface. It is applicable only while operating the AI in transparent mode and running a debugger. This allows the debugger to toggle the DTR line and reset the interface. This resetting amounts to clearing all status bits and throwing away any data in the pipes.

7.5.6. Notes

The command takes effect when LoadICE application exits and programs the AI to be a transparent link between the host and the target systems.

7.5.7. Examples

```
aireset
```
or
```
aireset 1
```
Enable the debugger reset feature

```
-aire 0
```
Disables the feature from the command line

7.6. aitint

Allow host debugger to directly interrupt the target system

7.6.1. Command Forms

aitint	LoadICE.ini file
-aiti	Command line
aitint	Dialog mode

7.6.2. Syntax

{**aitint** | **-aiti**}

7.6.3. Use

N/A

7.6.4. Default

No interrupt is generated to the target by the DTR/INIT toggle.

7.6.5. Description

When the host toggles the DTR line or the INIT line, PromICE can translate the signal and generate an interrupt to the target system via the 'int-; int+' pins on the back of PromICE. This allows the host based debugger to interrupt the target to return control to the monitor.

Generally you would have used the *ainorci* command to disable receive character interrupts and then this command lets the debugger drive the interrupt line whenever desired.

7.6.6. Notes

The interrupt is generated for a fixed period of time. The default is 100 micro seconds and can be changed by using the 'intlen' command.

7.6.7. Examples

`aitint`	Enable host debugger interrupt from the LoadICE.ini file.
`-aiti`	Enable host debugger interrupt from the LoadICE command line.

7.7. aitreset

Allows a host debugger to directly reset the target system.

7.7.1. Command Forms

aitreset	LoadICE.ini file
-aitr	Command line
aitreset	Dialog mode

7.7.2. Syntax

{**aitreset** | **-aitr**}

7.7.3. Use

N/A

7.7.4. Default

No reset is done.

7.7.5. Description

This command allows a host based debugger to reset / restart the target system by toggling the DTR or the INIT line.

7.7.6. Notes

This is a reliable way to recover when the target system is unable to recover from an error.

7.7.7. Examples

`aitreset`	Enable debugger reset from the LoadICE.ini file or in LoadICE dialog mode.
`-aitr`	Enable debugger reset from the LoadICE.ini command line.

7.8. burst

Program the AI to accept burst mode accesses.

7.8.1. Command Forms

burst	LoadICE.ini file
-B	Command line

7.8.2. Syntax

{**burst** | **-B**} [#]

7.8.3. Use

#	Specifies the number of reads executed during the cycle. The available bursts are: 0, 4, 8, and 16.

7.8.4. Default

Burst mode is disabled.

7.8.5. Description

Some 680x0 based target systems do burst mode reads from the ROM. The processor holds the /RD line and reads either 4, 8 or 16 bytes of data. During this time the /RD line is held and the address is changed.

In such cases this command allows the interface to be shifted up the address range and the interface would appear as if in a longer word configuration. On a normal access 8-bit target the interface occupies four successive bytes with internal offset of 0,1,2 & 3. However on a 16-bit system the consecutive bytes have target offset 0,2,4 & 6 and for a 32-bit system 0,4,8 and C (12). When burst mode is active the interface is effective spread over by as many bytes as the burst access allows. So a 4 bytes burst effectively maps the interface same as 32-bit target.

7.8.6. Notes

This command is only needed when using the AILOC command. Target burst reads do not affect normal emulation. The AI needs to be set up to handle the burst accesses.

7.8.7. Examples

`burst 4` Specifies a 4 byte burst in the LoadICE.ini file.

8. Sample Debugger Configurations

This section will outline sample configurations for AI communications with some of the debuggers that support PromICE AI option. These examples are taken from actual configurations.

Whenever possible, debugger versions have been listed. Refer the documentation for the particular debugger you have purchased for information specific to your configuration.

This section gives sample configurations for the following debuggers:

8051 family
ChipTools Chipview
68000 family
SDSI SingleStep
MRI XRAY

We have not included sample configurations for the following debuggers either because their configurations were not available at the time of printing or PromICE specific configuration is described in their documentation:

80*x*88/*x*86 family
Paradigm
CSI
PharLap
SSI

The list of debuggers supported by PromICE is continually growing. Refer the Application Notes for any additional debuggers.

8.1. ChipView® -51

ChipView requires certain hardware modifications to take full advantage of all of its capabilities. Following is a list of target requirements and why you may need them:

Overlapping CODE/XDATA space: If you don't have the ability to overlap the space you will NOT be able to download your application through the debugger, single step or set breakpoints in your code. How you go about this overlap is up to you, but you must tell the monitor how to overlap the memory space and/or where the overlap is.

One target interrupt: This allows you to bring control back from a "runaway" application. For example: If you press the F9 function key the debugger jumps to the application and lets it run at full speed. When you press the F9 key again, the interrupt is asserted and control comes back to the monitor.

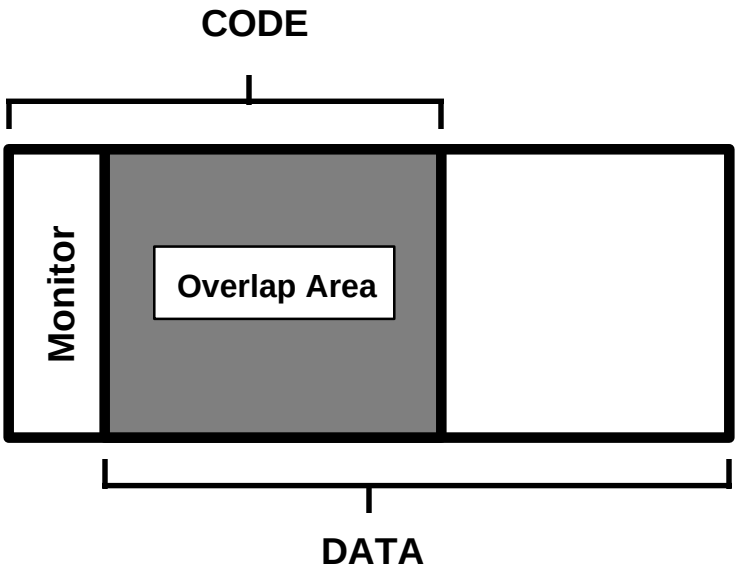
If you want to overlap the CODE/XDATA space you will need to modify your target hardware. How you do this is up to you, but you will need to configure the monitor to take advantage of the modifications.

8.1.1. 8x51 Debugging Concept

All references to the 8x51 and from this point on will be referred to as the 8051 unless otherwise noted. The 8051 has separate read lines for RAM and ROM and a write line for the RAM. This makes it impossible to do a write cycle to the ROM space without modifying the target system. In order for ChipView-51 to be able to download the application to your target system, set breakpoints and single step, you must modify the hardware so that some of your CODE and DATA address space overlap. Ordinarily, the physical ROM space where the CODE is to be downloaded to would be populated with RAM chips. By inserting PromICE into the ROM socket and connecting the WRITE line to PromICE, the monitor can be downloaded into PromICE and the rest of the CODE space can be written to. This still requires overlapping CODE and DATA space, but it isn't necessary to add an extra socket to your target system (one for the monitor and one for overlapped CODE) when using PromICE.

This requires a hardware modification to your target system. One way of implementing this configuration is to dedicate a port pin to be used to set

the overlay only when you want to write to ROM. By redirecting the write signal to the ROM address space you can then use the RAM as ROM (so when you do a write to the ROM space, it doesn't do a write to the same address in RAM and when you do a read, it will only be from RAM), and then clear the bit to put things back to normal.



If you overlap all of RAM and ROM space you must be careful that you don't write over your monitor code. If this happens your target and/or host will lock up. If this happens you will have to reload the monitor and start over.

8.1.2. Building the Monitor for Use with the AI

In order to take full advantage of PromICE/ChipView®-51 package you should have the following:

- A small amount of RAM in DATA space to hold the monitor state.
- A small amount of stack to breakpoint into the monitor code.
- Overlapping CODE and XDATA space.

Overlapping CODE and XDATA space is necessary for downloading your application and inserting breakpoints. This means that your target's architecture must support writing into the CODE space. The amount of RAM and stack necessary for the monitor depend on your configuration.

If you are not able to overlap the CODE and XDATA space you will need to load your application into PromICE using LoadICE at the same time you load the monitor. In the monitor program, you will need to eliminate any code that does writes to the "ROM" space. The monitor will try to write to the overlapping area. Instead, it will be writing to RAM space and may write over your reset vectors, interrupt vectors etc. Without this overlap, you won't be able to single step or set breakpoints.

In addition, you should have INT0 or INT1 available to break into the code in the event it runs away. If you cannot use an interrupt you will not be able to stop your application from running.

8.1.3. Process

- The first step to successful debugging with PromICE is to get the target emulating known working code. By doing this, you can verify that the LoadICE.ini file (discussed later) is configured correctly.

- The second step is to get the debugger working through a serial port on the target, with PromICE being used as a ROM emulator only. By doing this, you can verify that the monitor program has been configured correctly for your target system. If you don't have a serial port skip this step.

- The third step is to get the debugger running using PromICE Analysis Interface (AI). At this point, you will be adding PromICE AI specific code to allow you to debug through PromICE Virtual UART (AI).

8.1.4. PromICE Specific Modifications

PromICE disk you received with your ChipView®-51 / PromICE order contains prebuilt monitors and sources for the GET451 compiled under Archimedes/IAR, BSO/Tasking and Franklin/Keil formats. Also supplied on the disk are generic, prebuilt monitors and monitor sources for each of the compilers.

If you are using the GET451 target board you may use one of the prebuilt monitors and LoadICE.ini files and start debugging right away. You may also choose to modify the monitor to your particular application.

If you are using another target you may choose to use one of the prebuilt monitors as well. However, you will probably want to modify the monitor to take into consideration your target's configuration. For the purpose of

this document, we will assume that you are using your own target system.

1. The LoadICE disk contains two versions of the monitor program. They are described as follows:

CVMON51.ASM	BSO/Tasking format
CVMON51.A51	Franklin/Keil format
CVMON51.S03	Archimedes/IAR format

Select the format for the compiler you will be using.

2. Edit the monitor file you selected. Near the top of the file will be a selection for DEVELOPMENT or PRODUCTION.

For the DEVELOPMENT phase the monitor requires the following:

570 Bytes ROM at 0000
3+ Bytes RAM in XDATA/CODE space
INT0 or INT1 input pin (optional)

For the PRODUCTION phase the monitor requires the following:

700 Bytes ROM
INT0 or INT1 input pin

Select the option you want by setting the DEVELOPMENT or PRODUCTION bit to 1 and the other to 0.

Once you have decided to use DEVELOPMENT or PRODUCTION it is necessary to tell the monitor something about your target configuration and how you want to control it. You will only need to modify the code under either the DEVELOPMENT or PRODUCTION, which ever you selected.

3. If you want the target to start up in the monitor set the MONBOOT bit to 1.

4. MONSERIAL should be set to 0.

5. Set MONINT0 or MONINT1 to 1. This will determine which interrupt is used to vector to the monitor. You will need to set up

your interrupt vectors (ROMVECS) and make sure the interrupt is enabled in your application code. If you use this configuration you will also need to set MONSINT to 1. This will specify an interrupt driven UART.

6. If the chip you are using has a watchdog timer set MONDOG. If you set this bit you will also need to fill in the WATCHDOG routine.

7. Set the CHIP to the part number of the Controller you are using. Refer to the ChipView®-51 User's Manual for supported chips.

8. Set the MONCODE statement to the address where you want the monitor to reside.

9. Select the CSEG address where monitor will address PromICE's AI. This address will be used in the AILOC statement in the LoadICE.ini file (described later).

10. Save the file.

11. Next, edit the CSTART.ASM file.

12. Set up this file so that it matches your target's configuration.

13. Save the file.

8.1.5. Building the Monitor

The following is a sample list of commands to build the monitor for the BSO and Franklin compilers:

BSO/TASKING:

```
mpp51 cstart.asm
asm51 cstart.src

mpp51 cvmon51.asm
asm51 cvmon51.src
link51 cstart.obj, cvmon51.obj, c51s.lib to cvmon51.out
ihex51 cvmon51.out cvmon51.hex
```

Franklin/Keil:

```
a51 startup.a51
```

```
a51 cvmon51.a51
l51 startup.obj, cvmon51.obj TO cvmon51.abs
oh51 cvmon51.abs HEX(cvmon51.hex)
```

Refer to your compiler reference manuals if you want to customize any of these settings. For now, stay with the default settings until you get the debugger working. Once the configuration has been verified, you can customize these settings for your application.

Now we are ready to create a LoadICE.ini file and download the monitor into PromICE.

8.1.6. Downloading the Monitor to PromICE

1. Create a LoadICE.ini file and add the following lines:

```
output=com1
baud=57600
socket=27xxx
rom=27xxx
word=8
file=cvmon51.hex
ailoc xxxx,9600
```

where:

`output/baud`	The output serial port and baud rate you will be communicating at. If you want to use the parallel port to download the monitor faster, you can replace these lines with "*pponly=lpt1*". The debugger will still connect via the serial port, but the monitor program will be downloaded via the parallel port.
`socket`	Is set to the part number of the largest size ROM your target can use. This should be set to what your ROM socket is wired for, NOT the size you want to emulate.
`rom`	Can be set to any size, up to the socket size but not larger. This tells LoadICE what size ROM you want to emulate.
`word`	This is the word size of the target you are emulating. In this case it should be set to 8.

`file`	This tells LoadICE to download the monitor. You may need to relocate this file in PromICE memory if it starts at some address other than 0. Refer to the LoadICE Command Reference chapter for more information if this is necessary.
`ailoc`	This tells PromICE where the virtual UART (AI) is mapped. The *xxxx* is the address and "9600" is the baud rate. This address is equal to the address you specified in step 9 of the "*PromICE Specific Modifications*" section.

8.1.7. Starting the Debug Session

Before downloading the monitor and starting the ChipView®-51 debugger, a few additional lines must be connected from PromICE to the target. We will assume that you have already connected PromICE ROM cable from the target to PromICE. Refer to the installation section of this manual if you haven't connected the cable.

MAKE SURE THAT PromICE AND TARGET ARE TURNED OFF/UNPLUGGED BEFORE ATTEMPTING TO ATTACH OR DISCONNECT ANY CABLES. FAILURE TO DO SO WILL DAMAGE PromICE AND MAY ALSO DAMAGE THE TARGET!

Connect the write line (WRT) from PromICE to the target system. Where this connection is made to depends on how you have decided to overlap the code space. If you are not overlapping the CODE/DATA space this connection will not be necessary (refer to section 8.1.2, *Building the Monitor for use with PromICE/AI system* for more information).

Next, connect the interrupt (INT-) to the interrupt pin on the target you chose to use in step 5 of the "*PromICE Specific Modifications*". This connection is not necessary if you chose not to use an interrupt to bring control back to the monitor from the application.

You may connect the reset line (RST+) to the reset line on your target. By connecting this line, you won't have to manually reset or cycle power to your target system each time you download the monitor program. This is not required for the debugger operation.

Run the LoadICE application to load the monitor program. Once the LoadICE application has completed, reset the target either by pressing the reset button on the target or by cycling power to the target (This is not necessary if you have connected the reset line from PromICE to the target). DO NOT PRESS THE RESET BUTTON ON THE BACK OF PromICE UNIT. THAT WILL RESET PromICE ONLY AND BREAK THE AI LINK. If you press the button on PromICE you will need to rerun LoadICE to reinitialize the AI.

Now run CVM51. Once the debugger comes up you are ready to load your application and start debugging!

8.1.8. It Did Not Work

The debugger fails to connect with the monitor:

1. Verify that the target is emulating. Take a known working ROM and, using an EPROM programmer, backup the ROM up to a binary image and download it into PromICE using the same LoadICE.ini file with the following differences:

image=image.bin

The image statement tells LoadICE that you will be loading a binary image. Comment out the "file=" statement by placing a '*' in front of the line. Run LoadICE and reset the target if necessary. If the target doesn't run you may have a problem with your "socket" statement. Verify that the "socket" is set for the largest ROM that the target can use. Refer to the *LoadICE Command Reference* Chapter for more information on the "socket" statement.

2. Verify that the "ailoc" statement has been correctly specified in the LoadICE.ini file. If this has been set incorrectly the debugger will not be able to connect.

If none of these appear to be the problem, refer to PromICE *User's Guide* for more detailed diagnostics.

Debugger runs but monitor crashes when trying to load application:
If the debugger comes up but the monitor crashes when you try to load your application you are loading over all or part of the monitor program. Verify that the code you are trying to debug and the monitor are not located at the same address.

Cannot load application / set breakpoints in code:
The write line either isn't connected or the target isn't allowing the ROM/RAM overlap. Verify your target's configuration.

Cannot break execution of application (F9 key in ChipView®-51):
Verify that you have connected the interrupt line from PromICE to the correct interrupt on the target. The interrupt you are using should be defined in *PromICE Specific Modifications* step 5.

8.2. SDS SingleStep

This example generates a monitor for a simple 68000 board called TUTOR. Throughout this document we have used the notation used by SDSI for specifying a hex number by appending 'H' to the number, otherwise the numbers are decimal.

8.2.1. 68000 System

The TUTOR has 32K of RAM that starts at 000000H and 32K of ROM that starts at 8000H. Both RAM and ROM are 16-bit wide.

8.2.2. PromICE Specific Modifications

Since this target system doesn't have a prebuilt board support package, we will need to use the base model found in the \sds60\boards\proto directory.

- Edit the "board.h" file.

- Comment out the defines for FLOW_IN, FLOW_OUT, COMBINED, INTERVAL, VECT_READ AND READ_LEVEL by placing a ';' at the beginning of the line:

```
;INTERVAL:   equ 2
```

- Set the DEV_IN and DEV_OUT to PROMICE:

```
#define DEV_IN  PROMICE
#define DEV_OUT PROMICE
```

- Set BASE_IN and BASE_OUT to valid addresses.

- Set the PNAME_IN and PNAME_OUT to some valid value so that the correct I/O routines will be called. For this example, leave them at the default 0x0010040.

- Add the following lines to the USER_DEFS macro:

```
xdef  AI_ZERO,AI_ONE,AI_DATA,AI_STAT
SECTION usr_promice

AI_ZERO: dcb.b  2,$00
AI_ONE:  dcb.b  2,$01
AI_DATA: dcb.b  2,$00
```

```
AI_STAT: dcb.b  2,$CC
```

- Save the board.h file.
- Edit "board.def".
- Comment the defines for MMU and FPU
- Set the RAMADDR to 0:

```
#define RAMADDR 0x0
```

- Set ROMADDR to 8000:

```
#define ROMADDR 0x8000
```

- Comment the define for RAMSIZE.
- Save board.def.

8.2.3. Building the Monitor

- Run the MKMON utility:

```
mkmon board
```

- Use the DOWN utility to make a file that can be downloaded to PromICE:

```
down -d mot board.out -v -w 0x8000
```

where:

`-d mot`	Specifies a Motorola S-record format `board.out`
	Specifies the file to make a downloadable file from
`-v`	Displays verbose information to the screen
`-w 0x8000`	Sets the starting address of the created file to 8000H

8.2.4. Downloading the Monitor

- Use the sym utility to find the AILOC:

```
sym -m board.out |find "promice"
```

where:

`-m board.out` The .out file to search `find "promice"`Find the address where "promice" is located.

- When executed, the program should display the following:

```
usr_promice            8010H       8H
```

8010H is the physical address where the AI is located. The LoadICE "ailoc" requires that the address be an offset from the beginning of ROM (8000H). Subtract the starting address of ROM from the address from the "sym" utility. The result will be used in the ailoc statement when we create the LoadICE.ini file in the next step.

- Create a LoadICE.ini file. For this target, the file should appear as:

```
output=com1
baud=57600
socket=27256
rom=27256
word=16
file=board.dwn
```

- The file must be mapped properly to load in the right location. In general, most target's ROM starts at some address other than 0. For this particular case the address is linked to address 0, so the file statement should appear as:

```
file=board.dwn
```

This implies that [data in file and] ROM starts at 0H and should be loaded at address 0 in PromICE memory.

8.2.5. Specifying the AILOC

- If you have an 8-bit data bus to the ROM, then this number must end in 0H, 4H, 8H or CH. If your data bus is 16-bit wide, then this number must end in 0H, 8H and for a 32-bit wide ROM data bus, the number must end in 0H.

  In this case it happens to be at 0x10. Edit LOADICE.INI file to make sure that the ailoc statement is correct. For the MACH1 it should be

```
ailoc 10 9600
```

The second argument is baud rate that SingleStep will use (this is the debugger front end that you will execute).

- Now you are ready to load the monitor into PromICE unit(s). Make sure that your PromICE system is set up such that the MASTER module (lower emulation module, the one with host interfaces) is connected to the EVEN byte. The AI option is addressed via the EVEN byte. Edit the LOADICE.INI for the proper communication port. If you want faster downloading, use the parallel port specification.

- Now run LoadICE:

```
loadice
```

  LoadICE will download the monitor and program the AI to be a transparent link. If you have the `'rst-'` line connected from PromICE back panel to the reset on the target then the target should be running the monitor when LoadICE exits. Otherwise, you will need to reset the target. NOTE: If you must boot your target by power cycling then make sure that PromICE is externally powered. PromICE will loose the transparent link if it gets power cycled.

- Now invoke the debugger front end by double clicking on the SingleStep Monitor Debugger icon. You will see PromICE Rx and Tx light flicker and the debugger screen will come up.

8.2.6. It Did Not Work

At this point you need to make sure:

1. Your target system is running. You may need a scope to see that it is doing bus cycles.

2. Your ROM cables are connected to appropriate socket and PromICE modules.

3. You are emulating the proper size of ROMs for your target (check your socket statement in the LoadICE.ini file).

4. If you are plugging into a socket that is configured (wired) for ROMs larger than you are emulating then specify the socket statement to specify the socket size. (This is generally true when a socket is wired for a 4Mbit ROM and a 1 or 2 or 4 Mbit device can be inserted without any jumper requirements. These particular devices have the pins that are *don't_care* for higher address. PromICE has no *don't_care* lines, so the socket statement takes the additional lines into consideration).

5. If you have a 'write' line connected to PromICE, then the target could be writing over the monitor program by doing stray cycles while booting. For now, connect the line after the target has booted.

6. You can use a terminal emulator (i.e. PROCOMM) to test the interface. Set the communications to 19200,N,8,1 (you may need to change the baud rate). Download the monitor program to PromICE. Enter the terminal emulator and reset the target. The following should appear on the screen:

   ```
   {#@
   ```

   This character set will display repeatedly, pausing between each set. If this prompt appears, type:

   ```
   {#}
   ```

   If the monitor's user-provided initialization and output routines are working, the following should appear:

   ```
   {#+
   ```

   If the monitor doesn't pass this test either the LoadICE.ini file has been configured incorrectly, PromICE isn't connected correctly or the user-provided routines aren't working. There are three steps to diagnosing and correcting these problems:

   Take a known working file and download it into PromICE. If it doesn't work your LoadICE.ini file isn't correctly configured. Check your "socket" statement, byte order and/or file mapping.

   SOCKET: If you are plugging into a socket that is configured (wired) for ROMs larger than you are emulating then specify the socket statement in the LoadICE.ini file.

   This is generally true when a socket is wired for a 4Mbit ROM and a 1 or 2 or 4 Mbit device can be inserted without any jumper requirements. These particular devices have the pins that are *don't_care* for higher address. PromICE has no *don't_care* lines, so the socket statement takes the additional lines into consideration.

   MAPPING: If the debugger comes up but crashes when you try to load your application, you are probably writing over the monitor program.

This will only happen if you have the write line connected from the target to PromICE.

BYTEORDER: If you are using a 16 or 32 bit configuration verify that the byte order is correct. If the byteorder isn't correct the target won't emulate.

7. If none of these appear to be the problem refer to the Troubleshooting chapter for more extensive diagnostics.

8.2.7. CPU32

The primary difference for the CPU32 is the proper initialization of the processor for correct operation for your board. You may have specific initialization code for your board. This should be inserted after COLD_PC: in the MONBASE.SRC file. CPU32 chips are highly programmable in just about every aspect and require specific startup code.

Due to the detail and application specifics of this configuration refer to the SingleStep Debugger manual for more information on this subject.

8.3. Microtec XRAY

We have chosen to show this process by giving two specific examples. The first example generates a monitor for a simple 68000 board called MACH1. Through out this document we have used the notation used by Microtec Research for specifying a hex number by appending 'H' to the number, otherwise the numbers are decimal.

8.3.1. 68000 System

The MACH1 has 128K of DRAM that starts at 000000H and 64K of ROM that starts at 400000H. Both RAM and ROM are 16-bit wide. The MACH1 has a hardware BOOT bit that is set by a reset and must be cleared soon after reset. This bit forces RAM references to go to ROM until cleared, so the reset-vector is the first thing in ROM. The BOOT bit is kept in a CSR at address 800001H and is cleared by writing a '1' to the CSR. This will require that the BOOT bit be cleared before the monitor can use any RAM. The DRAM is refreshed by hardware and requires no set up. Also, for the MACH1, We will show you how to generate a monitor that does POLLED I/O and also a version of the monitor that uses INTERRUPT I/O. For interrupt I/O a wire (mini-hook) must be attached from the `'int-'` pin on PromICE back-panel to an interrupt source on the target. We used auto-vectored interrupt level 1, which is interrupt vector #25 (19H).

8.3.2. CPU32 System

The second example uses a board with a 68331 processor on it with 128K of ROM that is at address 000000H. The ROM is 16-bit wide (2 chips). However it has 32K of SRAM and it is only 8-bits wide, the RAM starts at 100000H. The internal registers of the 68331 must be programmed to generate proper chip_select for the SRAM before the monitor can use any RAM.

8.3.3. General Process

Use the MCT68K utility supplied by Microtec Research to build a monitor that will support your board. The MCT68K utility will ask you quite a few questions about your board. It will sequentially go through the main items and depending on the answers, it will ask you for different questions. Once it has acquired all the information, you will have a chance to review it. Once you tell it to go ahead, it will copy several files and then actually try to assemble the monitor. This process will fail since at AI specific files

have not been copied. Don't panic and follow the instructions given in this document and then the build procedure can be invoked successfully.

It is best to create a sub directory in the MON68K directory and run MCT68K from there. In this example we created a sub-directory MACH1 within MON68K and invoked the MCT68K as follows:

```
MCT68K -s
```

If your base directory is not /MON68K then you must specify the path to source files and invoke the command as such:

```
MCT68K -p path_to_source_files  -s
```

You can set the MON68K variable in the environment like this:

```
SET MON68K=C:/monitor-directory
```

Answer the questions for your target board as honestly as you can. The following is set to show you how to generate a version of the monitor for a custom board. In this particular case I am using the MACH1.

- For the first item select board type 15, a custom board.
- For the second item select 1, 68000 processor.
- For the third item select 2, Serial RS232 for debugger communication.

For POLLED I/O:

We will select a device that does not generate interrupt, so that the MCT68K will ask the proper questions. We will of course replace the driver with our own.

- Select item 2, 6850 ACIA. This device does not generate interrupts.
- For base address select item 3 "address hard coded in software".
- For the next item select #3. This has nothing to do the with baud rate we will use. It simply makes the monitor call the INIT routine, which we need.
- The next item is parity type, pick 1, it does not matter.
- Now we are back at main item #4. We will come back to it after showing you what you must pick for INTERRUPT mode I/O.

For INTERRUPT I/O:

Here we will select a device that generates interrupts so that MCT68K will ask the proper questions for configuring the monitor for interrupt driven I/O. Once again we will replace the driver with our own (for AI).

- Select item 4, 68562 DUSCC. This device generates interrupts.

- Select item 2 or 3 depending on if you want interrupt only or both. PromICE/AI will support either automatically.

- Now specify the interrupt vector. You must know where you are going to connect the interrupt output from the back panel of PromICE to your target. This interrupt is generated on the pins `'int-'` and `'int+'`. You will generally use `'int-'` since generally 68K interrupts are low asserted. In addition, the interrupt on these pins is asserted whenever there is a character available from the host. It is cleared when the target reads the character. The signal is tri-stated (turned off) when not asserted. Make sure that the place you are connecting this signal to is shareable. You may need to attach a pull-up resister to make sure that the signal goes high when not asserted. In other words, PromICE will only drive this signal low when asserted and turns it off when not asserted. A value like 1 or 2K Ohm to VCC will pull-up the tri-stated signal. Refer to the hardware reference for your target board to determine where the interrupt should be attached on the board. On the MACH1, we attached to a spare input pin (#11) on a 74LS148 priority encoder chip. You must then determine the vector number. For example, We hooked the signal to interrupt level 1 for auto-vectored interrupt. This is vector #25 or 0x19.

  Select item #2. This allows you to specify the vector number. Then specify the vector #. Remember that the numbers are decimal unless followed by an 'h' for hex.

- Next, for the base address of device pick # 3.

- Pick #3 for the baud rate selection. This allows monitor to call the INIT routine.

- Pick #1 for parity type, it does not really matter.

. . . Continuing the main process:

- Now we are at main menu item #4. This one asks if you have another monitor. Answer is 49, no monitor. You could have another monitor

on-board or a real-time OS that you want to hook up with the debugger monitor. If this is the case you will want to make the appropriate selection here. Refer to the XRAY68K reference manual for more information if necessary.

- Item 5 deal with where code starts. Pick #2 for this monitor to boot at reset. You may pick #1 if you have a different monitor. The debug monitor is then called from your other monitor.

- Now specify the address where your ROM starts. In this case (for MACH1), the answer is 400000H (hex value).

- For the data start address you should specify the address in RAM where the monitor can use the space for its own use. Pick #2. (You could pick #1 if you plan to hook the '`write`' line to PromICE and use it as RAM for the monitor to use. However, be aware that unless your target hardware allows, you may not be able to do byte writes to PromICE. This is true only if your target generates a single chip_select for all the ROMs for multi-byte ROM configurations, quite common).

- If you picked #2, then specify the RAM address which monitor can use. Don't specify 0 since low RAM is used by interrupt vectors (may not be true for some systems). We specified 1000H for MACH1.

- If you have an abort switch then pick something appropriate here, otherwise pick #1. The abort switch will let you return control to the monitor if your application hangs.

- Now you must select the interrupt level you want the monitor to run at. We picked #2 for interrupt mode I/O and #3 for polled I/O. However, you would probably not want #3 even for polled I/O.

- Pick #1 unless you have a 68881 coprocessor.

- This one is a bit tricky. You should pick #1 unless you have some reason to patch the config table, like if you plan to hook some multitasking OS etc.

- You can now see the configuration. If there is something you don't like then you can select that entry number and make any changes.

- Now pick #7. This will let you review your choices again. Answer Y if everything is correct.

- MCT68K will now copy several files to the current directory and try to build the monitor. This will probably fail with errors. Ignore any errors. The appropriate files needed to work with PromICE haven't been configured yet. We will now make the changes to support PromICE's AI virtual channel.

8.3.4. PromICE Specific Modifications

As a result of picking a UART there are two files in the current directory. They are HWEQU.INC and HWDRV.INC. These are of no use to us.

- From your distribution disk copy the AIEQU.INC and AIDRV.INC files to the current directory and rename them HWEQU.INC and HWDRV.INC respectively.

  • Edit HWEQU.INC to define the bus-width for your target. You can define a BYTE_WIDE, WORD_WIDE or LONG_WIDE configuration. You may also have to edit this file later to move the AILOC to appropriate address.

  This next step is critical for making sure that the monitor will execute on your target board. In the case of the MACH1, the boot bit must be cleared before the monitor can use any RAM. You must determine what code, if any, you need for your hardware.

- Use a text editor and edit file MONBASE.SRC. Find the label COLD_PC: Right after this label insert the code required to properly initialize the hardware. For MACH1 the following line is inserted here:

```
MOVE.B   #1,$800001 ; clear the BOOT bit in
                    ; CSR
```

  Save the file.

- There is also a label BOARD_START: in the file BOARD.SRC. Insert your board initialization code there. This is where you will insert any code that must initialize any peripherals etc. that you will use. The code to initialize vital resources such as turning on chip_selects, or DRAM refresh etc. must be done in the MONBASE.SRC as shown above.

8.3.5. Building the Monitor

Now you can build the monitor by typing the following:

```
BUILDMON
```

This process should be error free and produces an XDM68K.ABS file. This is the downloadable S-record file. This is absolutely linked. In case of MACH1 it will load at 400000H (where the ROM starts in target's address space).

If you get any assembly errors then check the syntax of the code you inserted. There should be no errors in the stock software.

8.3.6. Downloading the Monitor to PromICE

Now you are ready to load the monitor into PromICE unit(s). Make sure that your PromICE system is set up such that the MASTER module (lower emulation module, the one with host interfaces) is connected to the EVEN byte. The AI option is addressed via the EVEN byte. Copy the LOADICE.INI file from the distribution directory to your current directory. Edit the LOADICE.INI for the proper communication port. If you want faster downloading, use the parallel port specification.

You must make sure that your file is mapped properly to load in the right location. In general, most target's ROM starts at some address other than 0:

```
file=xdm68k.abs 400000=0
```

This implies that [data in file and] ROM starts at 400000H and should be loaded at address 0 in PromICE memory.

8.3.7. Specifying the AILOC

When you run BUILDMON it creates a MAP file. This file will give you location of many of the monitor globals. View this file with a text editor and look for AILOC. This line will show the AILOC address. In this case (MACH1), it is at 400068H. Since 400000H is the ROM start address, the AILOC is at offset 68H within the ROM space. Note this number for your particular case.

If you have an 8-bit data bus to the ROM, then this number must end in 0H, 4H, 8H or CH. If your data bus is 16-bit wide, then this number must end in 0H, 8H and for a 32-bit wide ROM data bus, the number must end in 0H.

If the offset for the AILOC is not correct for your bus size, then edit the HWEQU.INC (was AIEQU.INC) file and insert DC.L 0 statements to align the AILOC to the proper value (i.e. the last digit of the address in the MAP is outlined in the previous paragraph).

In this case it happens to be at 0x68. Edit LOADICE.INI file to make sure that the ailoc statement is correct. For the MACH1 it should be

```
ailoc 68 9600
```

The second argument is baud rate that XHM68K will use (this is the debugger front end that you will execute).

Now run LoadICE as:

```
loadice
```

LoadICE will download the monitor and program the AI to be a transparent link. If you have the `'rst-'` line connected from PromICE back panel to the reset on the target then the target should be running the monitor when LoadICE exits. Otherwise, you will need to reset the target. NOTE: If you must boot your target by power cycling then make sure that PromICE is externally powered. PromICE will loose the transparent link if it gets power cycled.

Now invoke the debugger front end:

```
XHM68K
```

You will see PromICE Rx and Tx light flicker and the debugger screen will come up.

8.3.8. It Did Not Work

At this point you need to make sure:

1. Your target system is running, you may need a scope to see that it is doing bus cycles.

2. Your ROM cables are connected to appropriate socket and PromICE modules.

3. You are emulating the proper size of ROMs for your target (check your socket statement in the LoadICE.ini file).

4. If you are plugging into a socket that is configured (wired) for ROMs larger than you are emulating then specify the socket statement to specify the socket size. (This is generally true when a socket is wired for a 4Mbit ROM and a 1 or 2 or 4 Mbit device can be inserted without any jumper requirements. These particular devices have the pins that

are *don't_care* for higher address. PromICE has no *don't_care* lines, so the socket statement takes the additional lines into consideration).

5. If you have a 'write' line connected to PromICE, then the target could be writing over the monitor program by doing stray cycles while booting. For now, connect the line after the target has booted. (Soon you will be able to tell LoadICE to disable the 'write' line until the target has booted, this will still leave your ROMs susceptible to application bugs, however, that may be fixed in some future version of the XHM68K front-end).

6. Edit the HWDRV.INC file and enable the LOOPBACK code. Run the BUILDMON program. This will include code that will read input from the host and increment the data by one and echo it back. You can use a terminal emulator (i.e. PROCOMM). Whatever you type should be echoed back as the next higher character (i.e. a will echo back as b etc.).

   If this test works, the AI circuit is working correctly. If the interface is randomly sending characters or is echoing wrong characters the AI suffering from some target timing glitches. Call Grammar Engine Technical Support for assistance.

7. If none of these appear to be the problem refer to the Troubleshooting chapter for more extensive diagnostics.

8.3.9. CPU32

The primary difference for the CPU32 is the proper initialization of the processor for correct operation for your board. You may have specific initialization code for your board. This should be inserted after COLD_PC: in the MONBASE.SRC file. CPU32 chips are highly programmable in just about every aspect and require specific startup code. Due to the detail and application specifics of this configuration refer to the XRAY68K Debugger manual for more information on this subject.

9. AI Porting

9.1. Overview

The AI is a memory mapped peripheral that can be located anywhere in the target's ROM address space (being emulated by the PromICE). The address at which the AI is located is configured by the host software.

The AI consists of four 8 bit registers. The AI registers are mapped in ROM address space to the master (bottom) PromICE. Once the AI is enabled, (reading) any of the four locations has the side effect of operating the interface.

The PromICE programs and enables the AI as commanded by the host software. Once enabled, it allows the host and the target systems to exchange data.

9.2. AI Register Description

This section contains the bit descriptions for each of the four registers.

ZERO
(read only) **Offset Address: 0**

7 **0**

-	-	-	-	-	-	-	-

Bit #	Mnemonic	Function
7-0	-	Reading this register sends a 0 bit to the AI.

ONE
(read only) **Offset Address: 1**

7 **0**

-	-	-	-	-	-	-	-

Bit #	Mnemonic	Function
7-0	-	Reading this register sends a 1 bit to the AI.

HOST_DATA
(read only) **Offset Address: 2**

7							0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Bit #	Mnemonic	Function
7-0	RB7:0	Receive Buffer Bits: These bits make up the last word received.

STATUS
(read only) **Offset Address: 3**

7							0
-	-	-	-	0	OVR	HAD	TDA

Bit #	Mnemonic	Function
7-4	-	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
3	0	Reserved. Do not modify this bit.
2	OVR	Host data overflow bit
1	HAD	Valid host data in buffer
0	TDA	Valid target data in buffer

9.3. AI Operation

9.3.1. Target Initialization

Initializing the AI is a process that involves both the target software and host LoadICE application. The target software sets up a four byte address in ROM address space as the AI address. These four locations must be initialized with 0xCC with a fill command in LoadICE. When the target is first started (power-up or reset), the target must wait for the AI to

be enabled. When the AI is ready (status changes from 0xCC), the status will change and the data register is read to clear the status.

9.3.2. Host Initialization

Since the target has no way of directly telling the PromICE the address of the AI, the PromICE must be configured by the host software (LoadICE). Using the AILOC command in LoadICE, the AI address, baud rate and port are set:

```
ailoc 10, 57600
```

The address “10”, is an offset address from the beginning of ROM. For example: The target accesses the AI at address 80010H. ROM starts at address 80000. By subtracting the base ROM address of 80000 from the AI address of 80010, we get the ailoc address (10). The AI address must be aligned on a proper boundary depending on the bus width:

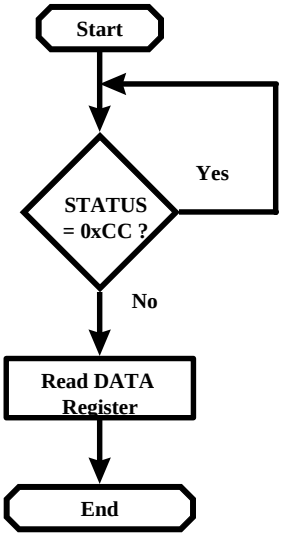
8 bit - lower two bits of the address must be zero
16 bit - lower three bits of the address must be zero
32 bit - lower four bits of the address must be zero

If, for example, you are using a 32 bit processor with one 8 bit ROM, your bus width would be considered to be 8 bit. The bus width is determined by the ROM bus width, not the processor’s bus.

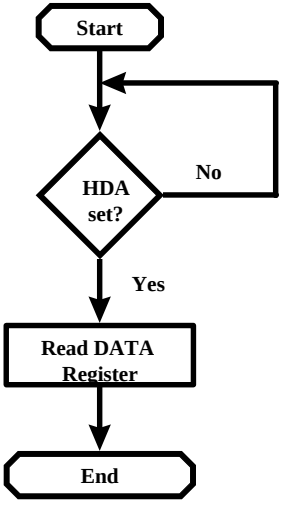
The baud rate (57600) is specified next. The parallel port can be used for communications by setting the baud rate to zero.

9.4. AI Algorithms

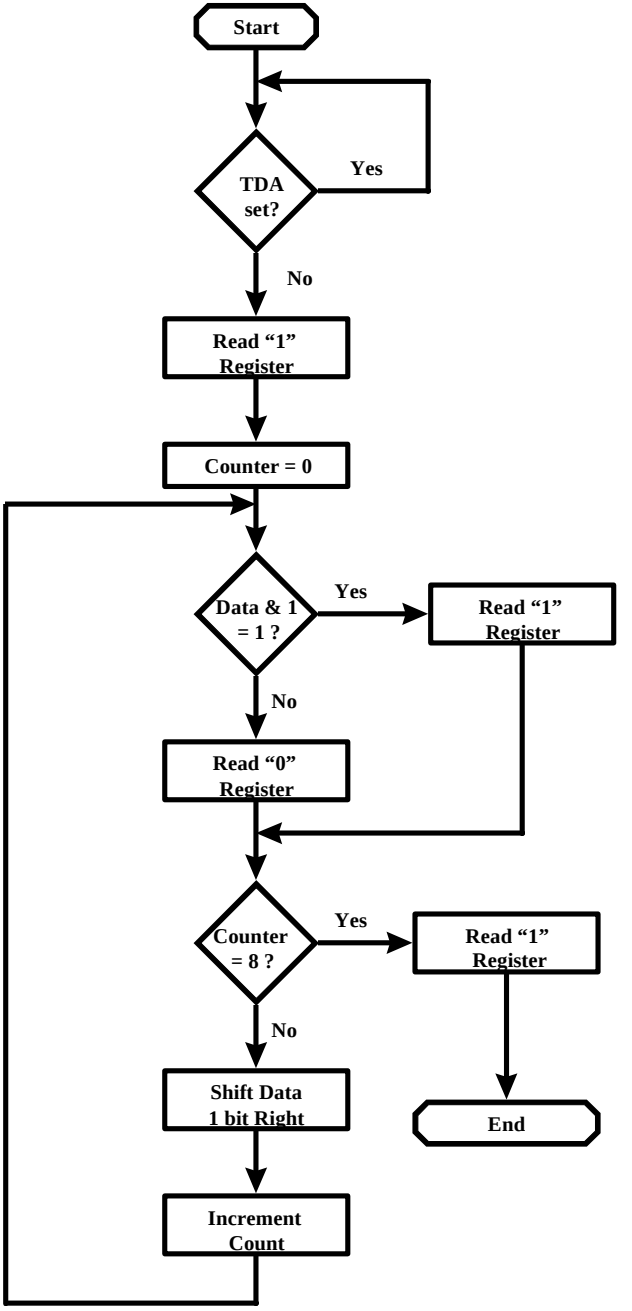
9.4.1. Initialization Algorithm



9.4.2. Read Algorithm



9.4.3. Write Algorithm



9.5. AI Code Example

The following example was written using Borland C/C++ 4.5 and Paradigm Locate 5.0. The target system is a 386EX development board (available from Grammar Engine) with a 512K x 8 ROM in address range from 80000H - fffffH.

```
#define BUS_SIZE   8      // Target ROM bus size (8,
                          // 16, 32)
#define AILOC  0x82000UL  // AI base address (Base
                          // ROM address = 80000H)

 // AI status register masks
#define TDA 0x01   // Target data available
#define HDA 0x02   // Host data available
#define OVR 0x04   // Host data overflow

#if BUS_SIZE == 8
typedef struct{
 unsigned char ZERO;
 unsigned char ONE;
 unsigned char DATA;
 unsigned char STATUS;
}FPORT, far * AISTRUCT;
#endif      // BUS_SIZE = 8

#if BUS_SIZE == 16
typedef struct{
 unsigned int ZERO;
 unsigned int ONE;
 unsigned int DATA;
 unsigned int STATUS;
}FPORT, far * AISTRUCT;
#endif      // BUS_SIZE = 16

#if BUS_SIZE == 32
typedef struct{
 unsigned long ZERO;
 unsigned long ONE;
 unsigned long DATA;
 unsigned long STATUS;
}FPORT, far * AISTRUCT;
#endif      // BUS_SIZE = 32

 // Create a pointer to the AI memory location
 // using above structure
volatile AISTRUCT AI = (AISTRUCT) MK_FP((unsigned int)
      (AILOC >> 4), (unsigned char)(AILOC & 0x0f));
```

```
/*********************************************
 * FUNCTION:    AIinit                       *
 *                                           *
 *   INPUT: NA                               *
 *   RETURNS: NA                             *
 * DESCRIPTION:  Waits for the AI become     *
 * available and data to be received from the*
 * host.  The first byte read from the AI on *
 * startup is not valid.                     *
 *********************************************/
void AIinit(void)
{
 unsigned char dummy;
 while(AI->STATUS == 0xcc){}  // Wait for host data
                              // available
 dummy = AI->DATA;            // clear interface
 return;
}

/*********************************************
 * FUNCTION:    AIputc                       *
 *                                           *
 *   INPUT: Data to be output                *
 *   RETURNS: NA                             *
 * DESCRIPTION:  Sends data one bit at a time*
 *           to the host.  Start and stop    *
 *           bits are also sent.             *
 *********************************************/
void AIputc(char data)
{
 int count;
 unsigned char dummy;

 while(AI->STATUS & TDA){}    // Wait until target
                              // data gone
 dummy=AI->ONE;               // Send start bit

 for (count = 0; count < 8; count++)
 {
     if(data & 1)
         dummy=AI->ONE;       // Send a 1 bit
     else
         dummy=AI->ZERO;      // Send a 0 bit
     data >>= 1;              // rotate right one bit
 }
 dummy=AI->ONE;        // Send stop bit
}
```

```
/*********************************************
 * FUNCTION:    AIgetc                       *
 *                                           *
 *   INPUT: NA                               *
 *   RETURNS: Data received from host        *
 *   DESCRIPTION:  Receives 8 bit data from the *
 *            host.                          *
 *********************************************/
char AIgetc(void)
{
 while(!(AI->STATUS & HDA)){} // Wait for host data
 return AI->DATA;         // Return data received
}
```

LoadICE.ini file:

```
output=com1
pponly=lpt1
socket=27040
rom=27040
word=8
ffill cccc
noverify
file=aitest.hex 80000=0
ailoc 2000, 57600
```

9.6. Using The AI with Interrupts

Interrupt driven communications is primarily used as a way for a debugger to regain control from a runaway application.

Implementing interrupt driven communications is simply a matter of connecting the interrupt pin on the PromICE (INT+ high asserted, INT- low asserted) to one of the target's interrupt lines. Usually, the INT pin is connected to NMI on the target system to allow for debugging.

Interrupts do not occur until the AIinit routine is complete (see code example). Once the port is initialized, an interrupt is generated for every character received from the host (HDA bit set).

9.7. Writing Into The PromICE Memory

The ability to write into ROM gives a debugger the ability to set breakpoints and single step in ROM address space.

The MWR and SWR pins on the PromICE are the write pins for the master (bottom connector) and the slave (top connector) of the PromICE. On newer duplex (P2xxx) PromICE units there will be two MWR pins and one SWR pin. There should be a jumper between one of the MWR pins and the SWR. The system write line should be connected to the available MWR pin.

On some targets, separate write signals are generated for odd and even writes. If this is the case, the jumper should be removed from the MWR/SWR and one of the connected to one of the MWR pins (it doesn't matter which one; they are connected internally) and the other to the SWR pin. It is very important that the correct write line get connected to the correct write pin.

9.8. Byte Swapping (16/32 bit only)

The AI is an 8 bit device mapped to the master PromICE unit. In most cases, the target will access the AI at an even address (the master PromICE is Byte 0). When the "word" statement is set to 16 or 32, the byte order is set to "0 1" or " 0 1 2 3" by default.. "0" refers to the master or bottom unit where the AI is physically mapped. When the byte order isn't set to the "0 1 …" default the target can no longer access is at an even boundary.

To move the AI to an odd address, the monitor and AILOC statements must be offset to odd addresses. Depending on the compiler being used, this may require some special padding so that the software can access a peripheral mapped to an odd address.

In a 32 bit system, the complexity is compounded by having more than one possible "non-zero" offset for the AI.

9.9. Breaking Transparency

The AI can be put into serial transparent mode using the parallel port. For example: Suppose we have the following LoadICE configuration file:

```
pponly=lpt1
..
..
..
ailoc 200, 19200
```

This configuration file tells LoadICE to use the parallel port only for LoadICE to PromICE communications. The ailoc statement puts the PromICE into serial transparency. This means that LoadICE talks to the parallel port and

the debugger or terminal program talks to the AI through the PromICE serial port.

When the PromICE is put into serial transparent mode, the parallel port is turned off. When LoadICE is run again (trying to connect to the parallel port), the connection hangs until the PromICE is reset. The solution is to add the serial port specification to the LoadICE.ini file:

```
output=com2
pponly=lpt1
..
..
ailoc 200, 19200
```

When LoadICE parses the LoadICE.ini file, it will see the combination of output, pponly and ailoc. LoadICE will then use the serial port to break the AI transparency. From there, LoadICE will communicate with the parallel port normally.

9.10. Burst Mode ROM Access

Some processors (i.e. 68EC030, 486) do burst reads from the ROM. This occurs when the processor holds the /RD line and changes the address. Either 4, 8 or 16 bytes of data are read at a time.

Adding ‘burst’ to the LoadICE.ini file with the number of reads per cycle as the argument, adjusts the addressing to support this processor function. Refer to the PromICE User’s Manual for more information.

9.11. Cache

If the memory space the AI occupies is cached, the AI will not be able to communicate. You must either disable the cache for debugging or find some other way of forcing a cache miss for every AI access.

9.12. Adjusting AI Timing

The AI is clocked by the accesses from the target system. The clocking is accomplished by combining /CE, /OE and the address lines. On some target systems, there is a possibility of false clocks from overlap between these signals. The ‘aicontrol’ LoadICE.ini switch is used to “de-skew” the signals.

This command is used most often by 683*xx* and DSP users (you may need to use 1 or 2 as the argument). If the target works reliably don't use this command. Refer to the PromICE User's Manual for more information.

9.13. Product Support

The following companies have ported their debugger environments to support the PromICE AI:

ChipTools
Concurrent Sciences
Paradigm
Microtec Research
Phar Lap
SDSI
SSI

If you have any comments or suggestions concerning this document please email or fax them to us:

email: support@gei.com
phone: 614-899-7878
fax: 614-899-7888

10. AI TROUBLESHOOTING

This section explains some of the possible causes why communications problems may occur when using PromICE with a debugger.

10.1. QUICK FIXES

Here are a few solutions to communication issues. If these suggestions don't apply or solve the problem, continue to one of the following sections.

10.1.1. Target Is Doing Burst Mode Access to ROM

If you have a target (i.e. 68EC030) that does burst mode accesses to ROM, add the "burst" statement to your LoadICE.ini file. Refer to the "burst" statement in the *LoadICE AI Command Reference* section of this manual for a description and configuration options.

10.1.2. Receiving Bad Data Back From Target

Add the statement "aicontrol 1" to your LoadICE.ini file. Depending on your target system, you may want to try "aicontrol 2" or "3". Refer to the "aicontrol" statement in the *LoadICE AI Command Reference* section of this manual for a description and configuration options.

10.1.3. Target Crashes During Download

If you are downloading your application into ROM space from the debugger and your target crashes at some point during the download, your application is overwriting you monitor.

The fastest way to diagnose this problem is to load your application and the monitor into PromICE together using LoadICE. Once the files are loaded, enter LoadICE dialog mode. This can be done automatically by adding the statements "load" and "dialog" to the LoadICE.ini file. Next, "compare" the files that have been loaded against the files on your disk. In dialog mode this is done using the command "c". If one of the files overlaps the other, the address that is overlapping will be listed. Relocate your application if this is the problem.

10.1.4. Watchdog Timer

If you have a watchdog on your target, disable it in the monitor startup code. The debug monitor will not come up if a watchdog is enabled or not configured in the monitor.

10.1.5. AI Not Getting Into Transparent mode

This can occur if you are trying to connect through the AI transparent mode in Windows with LoadICE running in another. The AI is only put in transparent mode when LoadICE exits.

10.1.6. Byte Order Swapped

The AI must be mapped to the master PromICE unit. If your LoadICE.ini file word statement looks something like:

```
word = 16 1 0
```

you will need to map the AI to the ODD address location. Since this word statement specifies that the ODD byte go to unit 0 (the master or bottom PromICE unit) you need to map the AI to the ODD address boundary. This usually means adding a "1" to the address location.

10.1.7. Can't Interrupt Target

In order to bring control back to the monitor from a runaway application or to enable interrupt driven communications you must have either the "int+" (high asserted) or "int-" (low asserted) pin on back of PromICE connected to the target.

The target monitor must be configured to handle the interrupt. Refer to your debug monitor documentation for information on interrupt driven communications.

10.1.8. Can't Write, SingleStep or Set Breakpoint in ROM Space

In order to write into the ROM space, the "wrt" pin on the back of PromICE must be connected to the target's write line.

Some target systems will not allow a write into the ROM space. Usually this is controlled by a PAL that is not allowing the write. Verify that the target supports write cycles into the ROM space and make appropriate modifications if necessary.

If the debugger needs to do byte writes (using multiple write lines) in a 16 or 32 bit configuration you need to have a PromICE customized to support multiple write lines. Contact your Grammar Engine Sales Engineer if you need this option.

10.2. DEBUGGER DIAGNOSIS

10.2.1. Never Worked

Step 1: Verify your LoadICE.ini configuration. Get PromICE emulating some known working code. At this point, you should not be trying to use the debugger. If possible, back up a working ROM to a binary image and load it into PromICE.

If this works, the only thing that could be wrong with your LoadICE.ini file could be the "socket", "file", or AI control statements. The rest of the configuration (i.e. "rom", "word"...) is correct.

If this doesn't work, go back to the installation section of this manual to reconfigure your LoadICE.ini file.

Step 2: Once you have verified your LoadICE.ini file, verify your monitor configuration. If possible, try to configure the monitor to work through a serial port on your target. If the monitor doesn't work there, it won't work with the AI either. If this is the case, check your monitor configuration.

Step 3: Next, uncomment or insert (depending on the monitor) the test code in the monitor. Refer to your debugger manuals if you are unsure how to do this.

If you are not receiving any characters from the target, your AILOC statement is probably incorrect. Verify that the ailoc is on the correct boundary and is at the beginning of the AI mapped area.

Verify that the baud rate specified in the ailoc statement matches the baud rate specified to the debugger. If you are still getting incorrect characters back from the target add the statement "aicontrol 1" to your LoadICE.ini file. If the problem persists, change the statement from "1" to "2", then "3".

If the monitor appears to be running, but you aren't getting any characters back at this point, contact Grammar Engine Technical Support for assistance.

10.2.2. Worked previously

If you have made a change to your configuration or to your source file and something fails (i.e. The debugger can't connect and/or the monitor crashes) the change is most likely where the problem is.

Go back into your configuration, step by step. Look at even the most benign changes. Most of the time it is the most minor change that causes the most problems.

10.3. WHO TO CALL

If you have a question and don't know who to call for support please call Grammar Engine. If we can't answer your question directly, we will direct you to someone who can. Here are some ways to determine who to call:

Call The Debugger Company...

Compiler and/or locator questions.
Target specific monitor configuration questions.
Debugger switches and options.

Call Grammar Engine Technical Support...

PromICE / LoadICE questions.
AI specific configuration questions.
If you don't know who to call.

15. Technical Specifications

15.1. Identification

PromICE models are identified as P1xxx or P2xxx as follows:

P1*nnn*- Simplex: Single (master) module for emulating 1 ROM.

P2*nnn*- Duplex: Two modules (master and slave in one box) for emulating 2 ROMs, where *nnn* is one of the following indicating the maximum capacity of the ROM:

512	Emulates 2716 - 27512	(64KBytes)
010	Emulates 2716 - 27010	(128KBytes)
020	Emulates 2716 - 27020	(256KBytes)
040	Emulates 2716 - 27040	(512KBytes)
080	Emulates 2716 - 27080	(1MBytes)
160	Emulates 2716 - 27160	(2MBytes)

Further postfixes that may be added to indicate the following options:

nnn: If faster than standard speed, then speed in nano-seconds is *nnn*.
AI: Analysis Interface for special firmware development features.

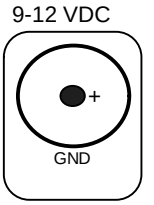
15.2. Power Consumption

Power consumption will vary depending on the buffers in the ROM emulation interface. Also faster models consume more power due to faster SRAMs and faster buffers. The figures below are based on a 4 Meg unit built with 4 1Mbit SRAMs and the ALS buffers.

PromICE Master: +5VDC < 200mA
PromICE Slave : +5VDC < 150mA
PromICE Analysis Interface: +5VDC < 70mA

Power Jack:

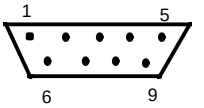
Pin and sleeve plug, with pin as +V and sleeve as ground. External supply provides 9VDC ~1A (unregulated)



15.3. Interfaces

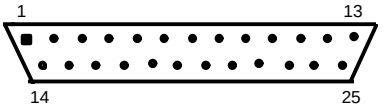
15.3.1. Serial Interface / DB9 female

The RS232-C connects to the host or a terminal via 9 conductor serial cable. Pinout are GND-5, RXD-3, TXD-2, CTS-8, DTR-4. GND is signal ground. TXD is data out of PromICE and RXD is data into PromICE. DTR is used as interrupt from the host. Among other things the interrupt signal is used for restarting PromICE from a known state.



15.3.2. Parallel Interface / DB25 male header

It is a Centronics compatible parallel printer port configured for direct connection to DB25 connector on the back of PC or Compatibles. It can operate bidirectionally by using 'error status lines' for sending data back 4 bits at a time.



PIN#	SIGNAL	SIGNAL	PIN#
1	STROBE	AUTOFEED	2
3	D0	ERROR	4
5	D1	INIT	6
7	D2	SELECTIN	8
9	D3	GND	10
11	D4	GND	12
13	D5	GND	14
15	D6	GND	16
17	D7	GND	18
19	ACK	GND	20
21	BUSY	GND	22
23	PE	GND	24
25	SELECT	GND	26

The data transfer protocol is modified so that the parallel port can be used bidirectionally. The STROBE line is used to send data from the

host to PromICE. The BUSY signal is asserted by PromICE to indicate that it either has not read the previous data from the host or it has data to send to the host. For this reason, the SELECTIN signal is used to acknowledge the unasserted state of BUSY line to PromICE. This ensures that the sense of the BUSY line is never confused by the host.

For sending data to the host, PromICE places data 4-bits at a time on the ACK, PE, SELECT, and ERROR lines and asserts BUSY signal.

When two PromICE units are daisy-chained on parallel port, the AUTOFEED signal is used as a STROBE line for the second unit, and the PE signal is used as the BUSY line. Because of this, the port is used for download only and the serial daisy-chain must also be used.

15.4. Indicators

RUN - a programmable run light, blinks during connect sequence; Rx & Tx - two LEDs for received and transmitted data signals (serial data only); LOAD - indicates when the unit is in load mode, i.e. not emulating.

15.5. Enclosure

5.08" Wide, 1.5" High w/o rubber feet, 5.25" Deep, Impact-resistant, ABS-molded, Grade DFA/R Plastic.

15.6. Environmental Restrictions

Operating Temperature: 5 to 32 degree C (41 to 90 degrees F)
Storage Temperature: -40 to 70 degrees C (-40 to 158 degrees F)
Humidity: 90% maximum without condensation.

15.7. Accessories

15.7.1. External Power Supply

A wall mountable power-supply is able to provide 912VDC unregulated and up to 1A of current. The plug on the power supply has the sleeve as ground and the pin as positive.

15.7.2. Standard ROM Cables

A standard (.6") DIP plug on 12" 24,28 and 32 pin Shielded (ground plane) Ribbon cable with a 34-position Female Header for mating with connector on the back of the unit.

15.7.3. ROM Socket

JEDEC 24/28/32 pin DIP socket w/100ns access. Non-JEDEC and non-DIP footprints are handled via custom cables.

15.7.4. Host cables

Shielded DB9M to DB9F and DB25M to DB25F cables are supplied. The DB9 cable is for connection from COM port to serial port of PromICE and the DB25 cable is for connection from the LPT port to the parallel port on PromICE. For connecting to COM ports that are DB25M a DB25M to DB9M adapter is provided.

15.7.5. Mini-Clip

The pin sleeve for attaching to PromICE pins and a micro-hook for attaching to the target system. Clip provided for reset line.

15.7.6. Custom ROM Cables

We currently support a wide range of footprints including DIP, PLCC, TSOP, PSOP and SSOP. Contact Grammar Engine Sales for the latest information.

15.8. Standard ROM Cable Pinouts

32-PIN DIP

PIN#	SIGNAL	SIGNAL	PIN#
1	A19	VCC	32
2	A16	A18	31
3	A15	A17	30
4	A12	A14	29
5	A7	A13	28
6	A6	A8	27
7	A5	A9	26
8	A4	A11	25
9	A3	OE_	24
10	A2	A10	23
11	A1	CE_	22
12	A0	D7	21
13	D0	D6	20
14	D1	D5	19
15	D2	D4	18
16	GND	D3	17

28-PIN DIP

PIN#	SIGNAL	SIGNAL	PIN#
1	A15	VCC	28
2	A12	A14	27
3	A7	A13	26
4	A6	A8	25
5	A5	A9	24
6	A4	A11	23
7	A3	OE_	22
8	A2	A10	21
9	A1	CE_	20
10	A0	D7	19
11	D0	D6	18
12	D1	D5	17
13	D2	D4	16
14	GND	D3	15

24-PIN DIP

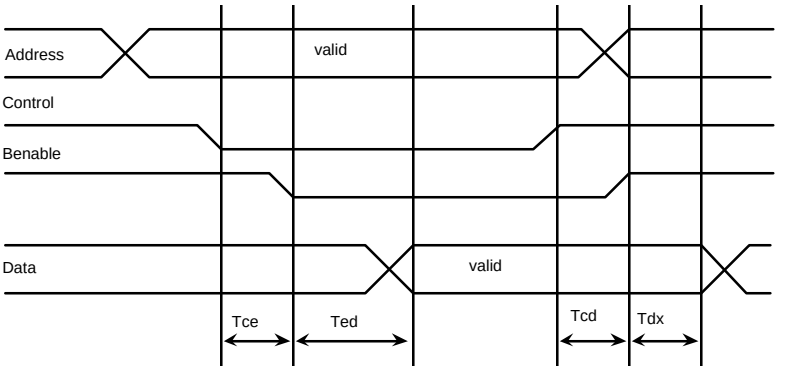
PIN#	SIGNAL	SIGNAL	PIN#
1	A7	VCC	24
2	A6	A8	23
3	A5	A9	22
4	A4	A11	21
5	A3	OE_	20
6	A2	A10	19
7	A1	CE_	18
8	A0	D7	17
9	D0	D6	16
10	D1	D5	15
11	D2	D4	14
12	GND	D3	13

15.9. Timing Diagrams

Here is how the target signals are received by PromICE. The *address bus* and the *chip_select* and *output_enable* control signals are received by uni-directional buffers (74ALS244) and the *data bus* is connected to a bi-directional buffer (74ALS245) . When PromICE is in 'load' mode, these buffers are turned off and their outputs are tri-stated.

When PromICE is emulating these buffers are turned on and the address buffers supply the address to the emulation memory within PromICE. The data buffer is enabled by a combination of the *chip_select* and*output_enable* signals. The direction of the data buffer is controlled by a target supplied *write* signal.

Emulation memory is made up of SRAM chips that are selected directly by the address supplied by the target system (in other words without using *chip_select* or the *output_enable* signals). This allows faster access to data. The data is placed on the target's data bus by the data buffer. This achieves a faster response from PromICE to a target driven ROM read cycle.



Tce - time from chip_select & Output_enable to internal buffer enable.

Ted - time from buffer enable to data valid (delay through 74xxx245)

Tcd - time from chip_select & output_enable to buffer disable

Tdx - time from buffer disable to data bus tri-state

These times will vary according to the buffers used. The typical values are: Tce - 35ns / Ted - 20ns / Tcd - 35ns / Tdx - 20ns

16. Internal Memory Addressing

PromICE can address up to 1 megabyte of memory per module. Master and slave modules are addressed by switching the internal circuit to select either module. There a total of 20 address lines required to access the 1 megabyte of memory. When less than 1 meg of memory is present, the higher address lines are pulled high. The micro controller in PromICE can set its I/O lines to an "off" state and the internal pull up resisters will pull the signals up. The unused address lines will remain pulled up. Therefore, internally PromICE addresses memory by using this map:

address 0	last address	Emulated ROM size
1F F8 00	1F FF FF	2K
1F F0 00	1F FF FF	4K
1F E0 00	1F FF FF	8K
1F C0 00	1F FF FF	16K
1F 80 00	1F FF FF	32K
1F 00 00	1F FF FF	64K
1E 00 00	1F FF FF	128K
1C 00 00	1F FF FF	256K
18 00 00	1F FF FF	512K
10 00 00	1F FF FF	1M

When using a terminal or a terminal emulator to talk to PromICE the actual address used by PromICE is displayed. An example would be a unit equipped with 128KBytes of memory when emulating a 32kbyte ROM will display address 0F8000 for address zero. This will compute to be the highest one addressed of the four chunks of 32kbytes that comprise the 128kbyte unit.

This is why the LoadICE software must be told exactly what size ROM you are emulating using the "socket" and "rom" statements. It ensures that the data is loaded at the proper place in the internal space.

17. Index