PromICE

User Manual

Version 4.0

GRAMMAR ENGINE INC.

921 Eastwind Drive, Suite 122, Westerville Ohio 43081
(614) 899-7878 Voice - (614) 899-7888 Fax
www.promice.com

PromICE User Manual

Version 4.0

PRINTED IN THE UNITED STATES OF AMERICA

What's New in This Edition

This Version 4.0 release of Grammar Engine Inc.'s LoadICE software and PromICE User Manual benefits customers new and experienced alike. Here is how:

- **Setup -The Configuration Tool**
  Provides a cursor menu interface for choosing configuration options and creates the loadice.ini file to store your settings.
- **Simplified Command Descriptions**
  Command documentation now in an easy to read format.
- **Introduction to Dialog Mode**
  Get started with Dialog mode that provides convenient, interactive diagnostics and prototyping.
- **Analysis Interface (AI) Configuration Reference**
  In-depth procedure for AI configuration.
- **AI Interrupt Control Commands**
  Aircvint controls AI receiver interrupts. On AI2, **aixmtint** regulates AI transmitter interrupts. **Intlen** defines interrupt request duration.
- **New Commands for Flexible loadice.ini Programming**
  In addition to defining session settings, the **begin** and **clearfiles** commands enable limited scripting capability for fills and loads.
- **Ethernet Microplex Connectivity**
  The **ethernet** command supports Microplex print server connectivity.
- **Analyze Command**
  Reads your hex or image code files and reports the information you need to create the **file** command.
- **Fillall Simplifies Clearing Memory**
  Use the **fillall** command to erase all PromICE memory.
- **Refined Control of PromICE**
  The **modein**, **modeout**, and **modefixed** commands allow control of the entry and exit mode of PromICE (Emulation or Load mode).
- **Revised File Command**
  The file command syntax is simpler than before.
- **PromICE Unit IDs Chapter**
  For those who use unit IDs.

Table of Contents

WARRANTY

GEI provides a 30-day money back guarantee on its products. Within that period, if you are not fully satisfied with 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 return ship the product via three 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.

Technical Support

You can contact Grammar Engine Technical Support via our web site, email, phone and fax.

Web site	www.promice.com
By Phone	(614) 899-7878
By Fax	(614) 899-7888
By Email	support@gei.com

Please have the following information:

PromICE Model and Serial number (from the underside of the PromICE case, model number starts with a "P")

GEI adapter model number(s)

All switch and/or jumper settings on PromICE and adapter boards

Contents of the loadice.ini file or command line

The error message being encountered

Target information:

- CPU type
- Clock rate
- ROM part number
- ROM access time

Host type and type of connection to PromICE (serial, parallel or both)

1. Introduction

Contents

Welcome to PromICE

PromICE adds convenience and new capabilities to your embedded development environment. The AI option enhances target monitor-based debugging. The Trace option provides real time, non-intrusive trace information.

What is PromICE?

Grammar Engine PromICE began as a **memory emulator**. By replacing your target's ROM with its internal RAM memory, PromICE speeds firmware testing. Normally testing new ROM code involves burning an EPROM or programming a Flash memory, which are time-consuming tasks. Loading test code into **PromICE emulation memory** is fast and convenient.

More than just a memory emulator, PromICE accelerates the debugging process through a number of features that enhance the capabilities of software-based debuggers as well as stand-alone debugging.

Each PromICE system consists of the following:

- The PromICE unit.
- The external power supply supplied with PromICE.
- **Host cables** to connect PromICE to your workstation.
- **Target cables** to connect PromICE to your target system.
- **LoadICE** software, for your host or workstation, to manage your PromICE unit.

How PromICE works

During hardware and software installation, you will setup your PromICE and connect it to your host and target systems. You can setup the Host-to-PromICE communication settings using the LoadICE software via the **loadice.ini** file or the DOS console using **command line switches**.

When connecting PromICE to your target system, use GEI target cables, which may contain adapter boards and can connect to your target in the following ways:

- **Socket probe** that fits into the ROM socket in place of your target's EPROM or Flash memory chip.
- **Solder-down probe** which provides a sturdy connection to your target.

- GEI **Direct Connect interface** consisting of a ROM cable terminated with a female 34 or 60 pin connector and a male header built into your target board.

During setup for ROM emulation, you can use the LoadICE software to configure your PromICE via 1) building a loadice.ini file using the **setup** command, 2) using command line switches, or 3) issuing commands in the interactive **Dialog mode.** The **rom**, **word** and certain other LoadICE commands specify the characteristics of the PromICE emulation memory. This is called the current **PromICE ROM Configuration**.

Following hardware and software installation, you can load new code into PromICE emulation memory when PromICE is in **Load mode**. While PromICE is in Load mode, the target cannot access PromICE emulation memory. By connecting PromICE to a shareable reset input on your target, PromICE can hold your target in reset during Load mode.

Once the test code has been loaded, you can use LoadICE to command PromICE into **emulation mode**. While emulating, PromICE responds to target processor memory fetches like an EPROM. The target executes your new code, which it reads from PromICE emulation memory. While PromICE is in emulation mode, LoadICE commands cannot access the contents of PromICE emulation memory.

Internally, PromICE is composed of **PromICE 8 bit units**, each unit having an 8 bit word and containing the amount of emulation memory indicated by the PromICE model number. (See *Technical Specifications* for the model number format.) A **Dual PromICE** contains two 8 bit units and can emulate either two 8 bit ROMs or one 16 bit ROM with the use of a 16 bit adapter. Multiple PromICEs can be **daisy chained** using host cable adapters to emulate larger word sizes.

Main PromICE Features

Fast Downloading
PromICE downloads in seconds, rather than the minutes required by ROM programming, target serial ports, In-circuit emulators and BDM downloads.

Flexible Emulation
With up to 32 Mbit (2 MB by 16), a PromICE can emulate any ROM equal to or smaller than its full capacity. PromICE is available in access speeds as fast as 35ns.

Targets with 16 bit data buses can be emulated with a Dual PromICE. For 32 bit and larger data widths, PromICEs can be daisy chained together through serial or serial/parallel connections.

ROM footprints supported
PromICE supports many ROM package types including DIP, PLCC, TSOP, PSOP, SSOP, SIMM, and micro BGA footprints as well as thousands of ROM, EPROM, and FLASH devices. Custom cables can be provided for other socket configurations.

Full ROM space available for debugging
When you use a monitor burned into ROM, you must use RAM to test your code. This is inconvenient and sometimes impossible. PromICE makes the full ROM space function like RAM during debugging. Code can be loaded into the same ROM addresses used in the final design. This eliminates the need to relocate code after debugging.

Write to ROM
Connecting the target write line to PromICE allows debuggers to set breakpoints and load code. PromICE supports emulation of EPROMs, EEPROMs, RAM, etc. PromICE also supports reset and interrupt line outputs to the target, providing debuggers with more control.

PromICE management software included
The LoadICE software supports download file formats including Binary, all Intel hex, and all Motorola S-records. Host support includes DOS, Windows 95/98/NT, UNIX, HP-UNIX and VAX/VMS systems. Full ANSI C source code is included for the UNIX and DOS versions.

Non-Volatile memory
PromICE retains emulation memory when the target is reset and reinitialized for code testing. Data is retained for a time even when PromICE is shut off.

PromICE/AI

What is PromICE/AI?

The PromICE Analysis Interface option, referred to as "AI," implements a **virtual serial port** for use by a monitor-based debugger.

A monitor-based debugger consists of a host-side **front-end** and a **target-based monitor**. Monitors that support PromICE/AI communicate with their host side software through the **AI Interface** located within the ROM address space. The host connects to the PromICE serial port instead of a target serial port.

The serial port on your target is no longer needed for debugging. This is helpful when your application or a RTOS requires the target serial port.

Note: The PromICE AI option is available on units with "AI" or "AI2" as part of the model number.

How PromICE/AI works

The AI Interface, which emulates an UART, is composed of four **AI control registers**, which are mapped into a region within the PromICE ROM emulation memory. The target can communicate bi-directionally with the host by reading this interface. AI does not require the target to write into ROM addresses in order to transmit data.

While the AI Interface is active, PromICE is in **an AI communications session** and functions as a pass-through data-forwarding device. LoadICE cannot be used to access PromICE without disrupting the current AI session.

Internally, the AI option is a PromICE daughter board. On a dual PromICE, the AI control registers are only accessible through the **master PromICE 8 bit unit**, ID=0. The master unit connects to your target through the lower connector on the back of a dual PromICE.

With minimal code, your target system can have bi-directional communications using the AI option without costing target hardware resources. The AI option can provide interrupt driven communications. Also, if you have AI2 and your target permits write access to ROM addresses, your debugger monitor can transmit data by writing to one of the AI control registers.

Main AI Features

Virtual Serial Port
Your debugger monitor can communicate with the host through PromICE's virtual serial port, located in the PromICE emulation memory address range.

Frees target serial port for other uses
The serial port on your target is no longer needed for debugging. This is helpful when your application or a RTOS requires the target serial port.

Interrupt driven communications
By connecting the target interrupt line to PromICE, the debugger monitor can use interrupts to receive data from the host, and on AI2 use interrupts to transmit data to the host.

Dialog Mode

What is Dialog Mode?

Dialog mode provides an interactive session within the LoadICE software. You can use Dialog mode to prototype PromICE configurations and diagnose problems. At the Dialog mode input prompt, you can enter LoadICE commands and see the results immediately.

To access Dialog mode, use the **dialog** command as a switch on the LoadICE command line or in the loadice.ini file (see the **dialog** command for details). It will force LoadICE to enter Dialog mode upon finishing loadice.ini file processing.

How Dialog Mode works

Upon entering Dialog mode, PromICE will be in Emulation mode by default. Emulation mode means that your target can access PromICE emulation memory and can run your code. LoadICE commands that access emulation memory cannot be used in Emulation mode and are only available in Load mode.

You can use the **stop** command to place PromICE into Load mode. Load mode means that LoadICE is in control of PromICE emulation memory and that your target cannot access it. The LoadICE input prompt changes between Emulation mode and Load mode (see below). When in Load mode, use the **go** command to return to Emulation mode.

Note that if you specify files to be loaded in your loadice.ini file along with the **dialog** command, LoadICE will enter Dialog mode **without** loading these files. To load the files before entering Dialog mode, modify loadice.ini and place the following commands before the **dialog** command: the **begin** command followed by the **load** command (see below).

Without a **dialog** command in loadice.ini, LoadICE will load user data files according to the configuration defined in loadice.ini and then exit. However, if no data files are specified, LoadICE will simply exit.

You can use most ROM configuration commands in Dialog mode to prototype your configuration and test its validity. Note that commands to configure communication between the host and PromICE can only be used within loadice.ini or as switches on the loadice.exe command line. These commands include **output**, **baud**, **pponly**, etc.

For more information on diagnostic LoadICE commands, see the introduction to the *Troubleshooting* chapter.

Using Dialog Mode

In this manual, user input is shown in bold after the input prompt. For example:

```
02:LoadICE: your input
```

By adding the **dialog** command to your loadice.ini file, LoadICE will enter Dialog mode. For example:

```
pponly=lpt1
word=8
rom=27040
file=loaddata.hex 8000=0
dialog
```

This loadice.ini file specifies a data file to be loaded over the parallel port and then enters Dialog mode. Note that the file has not been loaded.

Upon entering Dialog mode, PromICE will be in Emulation mode by default. The Dialog mode command prompt in Emulation mode will resemble:

```
02:LoadICE:
```

Use the **stop** command to place PromICE into Load mode. For example:

```
01:LoadICE: stop
Now in Load Mode!
Use 'go' later to emulate

00:LOADice:
```

The Dialog mode command prompt in Load mode will resemble:

```
02:LOADice:
```

Next, use the **load** command to actually load the file into PromICE.

```
00:LOADice: load
Opening file `test.hex` for processing/Done
Transferred 865 (0x361) data bytes

00:LOADice:
```

Next, use the **go** command to return to Emulation mode.

```
00:LOADice: go
Now Emulating!

00:LoadICE:
```

Now reset your target. Next you can test your target and use the PromICE **status** command to see if PromICE can detect target power and if your target is at least accessing PromICE memory.

```
00:LoadICE: st
TARGET STATUS:  Power is ON - Target is accessing ROM

01:LoadICE:
```

For more information of on diagnostic LoadICE commands, see the Introduction to the *Troubleshooting* chapter.

Pre-load Example

To cause LoadICE to load your files before entering Dialog mode, modify loadice.ini and place the following commands before the **dialog** command:

```
begin
load
dialog
```

The **begin** command initiates communication with PromICE. The **load** command then loads the file immediately.

```
pponly=lpt1
word=8
rom=27040
file=loaddata.hex 8000=0
begin
load
dialog
```

This loadice.ini file will load a file over the parallel port and then enter Dialog mode. Note that PromICE will be in Emulation mode upon entering Dialog mode.

Getting Updates and Giving Feedback

Be sure to visit **www.promice.com** for a free upgrade to the latest LoadICE software and documentation. If you don't have web access, contact your sales representative to obtain a copy.

Note: LoadICE version 4.0 or higher is required to use the commands documented in this version of the PromICE User Manual.

Comments and suggestions should be sent to support@gei.com.

2. Installation

This chapter contains installation instructions for PromICE. Please complete hardware and software installation and then proceed to the next chapter to configure the software.

Contents

Unpacking

When you take PromICE out of the static shielding packaging, you must be in a **static safe environment** or else you can easily damage your unit.

You can protect your PromICE from static damage by following these safety guidelines and handling precautions:

Keep PromICE in its anti-static bag until you reach your work site. If the box must be opened elsewhere for inspection, under no circumstance should you or anyone else remove the PromICE unit from its anti-static bag.

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 your PromICE, target system or any other static-sensitive device.

If you must move PromICE:

1. Wear a grounded wrist strap.
2. Remove power from both the PromICE and target system.
3. Disconnect all cables beginning with the cables from the PromICE end first.
4. Return PromICE to its anti-static bag.

Now PromICE can be transported. Never carry PromICE with the target cables still attached. You can easily damage the buffers and the ROM cable pins.

When handling PromICE or your target, do not touch any exposed pins or other exposed metal parts. Most of the pins are direct connections to buffer inputs and are very sensitive to static damage. Damage to a buffer can cause intermittent or permanent failures.

Storing and Shipping PromICE

Always store and transport PromICE in an anti-static bag.

PromICE ships in a custom cardboard container with cutouts for various parts. This container is suitable for storing PromICE when not in use and returning it for repair.

Be sure to keep the anti-static bag to store PromICE when you are not using it.

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 due to mishandling or misuse.

Connecting PromICE

1. Power down your target system. Failure to turn off power may damage PromICE and/or the target.
2. Wearing a grounding wrist strap, remove PromICE from its anti-static bag.
3. Connect the serial and/or parallel cables to PromICE.
4. Use the sections on the following pages to connect PromICE to your target.
5. Power on PromICE and then your target.

Disconnecting PromICE

1. Power down your target and then PromICE. Failure to turn off power may damage PromICE and/or the target.
2. Disconnect the ROM cable from PromICE side **first**.
3. Disconnect the cable from the target system. If you are using a ROM socket probe, **first** disconnect the adapter board from the probe. Then remove the probe from the ROM socket. When removing probes equipped with a finger loop, be sure to pull straight up.
4. Disconnect the serial and/or parallel cables from PromICE
5. Place PromICE in an anti-static bag.

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

Note: Some PromICE may retain the contents of emulation memory for a period after power is removed. The length of the memory retention time varies with the PromICE model and memory size. To clear the entire contents of memory, use the **fillall** command.

Connecting PromICE to the host PC

PromICE can be connected to the host in these ways:.

- Serial port only (see **output** and **baud** commands)
- Parallel port bi-directional on a single unit (see **pponly** command)
- Parallel port bi-directional; the serial port used for AI communication (see the *AI Configuration* chapter)
- Serial port with download-only parallel port (see **ppbus**, **output**, and **baud** commands)
- Daisy chained units using serial ports
- Daisy chained units using serial ports with download-only parallel ports
- Ethernet (see the **ethernet** command)

Note: Certain A/B switch boxes, when connected between the host parallel port and the PromICE, 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** commands to your loadice.ini file. Refer to the daisy chain installation section for more information about daisy chaining.

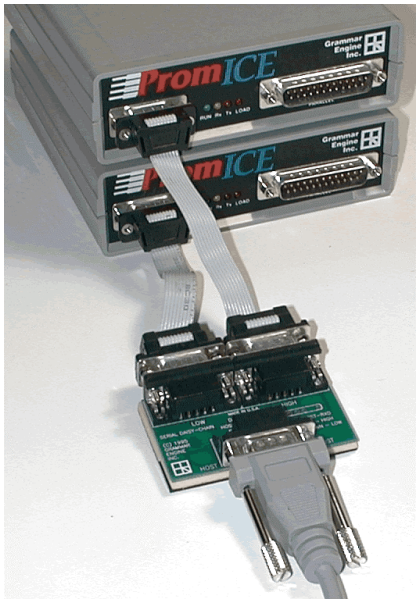
Serial / Parallel connection

Using the supplied serial and parallel cables, connect the PromICE to the host. Refer to the next chapter for software configuration.

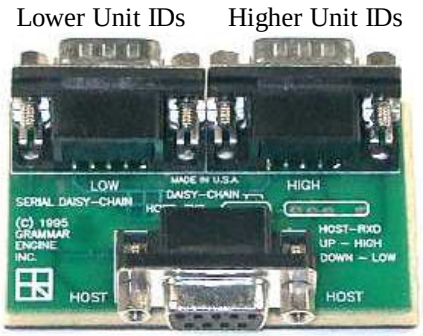
Daisy-chained PromICEs using serial ports

To connect multiple PromICEs to a single host serial port, use DB9 daisy chain adapters. You can connect multiple PromICEs using daisy chain adapters to emulate wider target word sizes and/or a large configuration of ROMs.

The DB9 daisy chain adapter comes with two DB9 cables.



DB9 Serial Daisy Chain Adapter



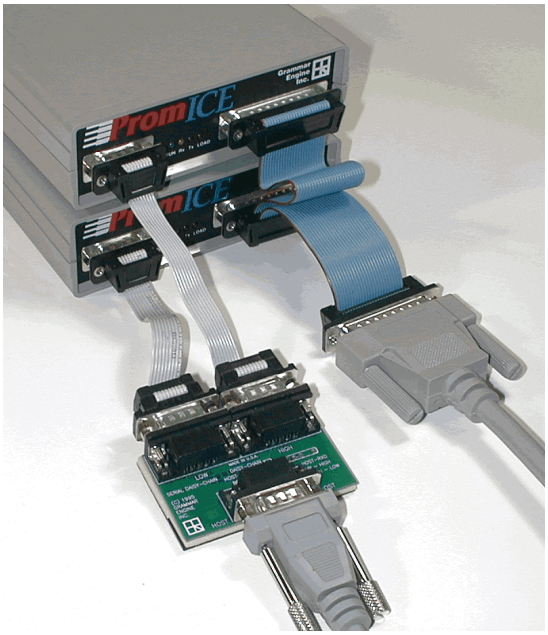
With the center jack facing you as shown, connect the first PromICE unit to the left connector and the other PromICE unit to the right connector.

The DB9 adapters can be connected to each other to create a tree arrangement. Up to 256 PromICE units can be daisy chained in this way. The PromICE 8 bit unit IDs are assigned at communication startup in ascending order beginning with zero on the left-most daisy chained PromICE.

When the target word width is greater than 8 bits, the default byte order is: all known PromICE 8 bit units arranged in ascending unit ID order. The **word** command can explicitly define the byte order within the target word width by means of a PromICE 8 bit unit ID list.

Daisy-chained PromICEs using serial and parallel ports

When using PromICE in this configuration, the serial port must be used for full communication with the PromICEs, whereas the parallel link mode is download only in order to achieve fast download times.

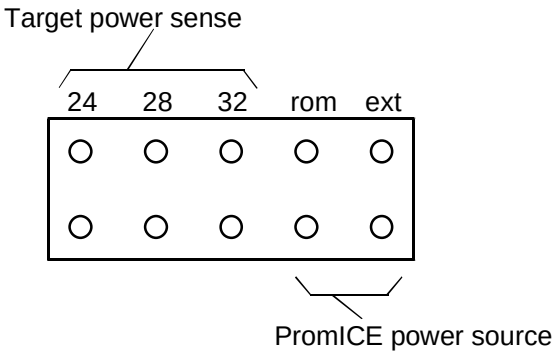


Connect the serial daisy chain adapter(s) as described on the previous page.

Then connect the parallel bus cable in whatever way is most convenient. Since the serial daisy chain adapter arrangement handles all control duties, the order of PromICE units on the parallel bus cable is not important.

Connecting PromICE to the target system

Power Source Jumpers on the PromICE back panel



EXT / ROM Jumpers

Always select external power using the **EXT jumper**. Use the power supply rated for use with the PromICE model (see the *Technical Specifications* chapter).

The **ROM jumper** should only be used when connecting the 3 Volt Adapter or to parasitically power a small, low power target. Your target should not parasitically power the PromICE unit.

Target Power Sense Selection

Use the **32 jumper** for all other cables except the 24 and 28 pin DIP. Use **the 24 jumper** when using a 24 pin DIP cable. Use the **28 jumper** only when using a 28 pin DIP cable.

Jumper Settings for the 3 Volt Adapter

When using the GEI 3 Volt Adapter, use the **EXT**, **ROM**, and **32 jumpers**. The 3 Volt Adapter has a spare, long handled jumper on one of the **Write** pins that you can use for the **ROM jumper**.

Connecting Target Cables

This section describes installation of the 28 and 32 pin DIP cables only. Refer to the cable documentation included with the cable assembly.

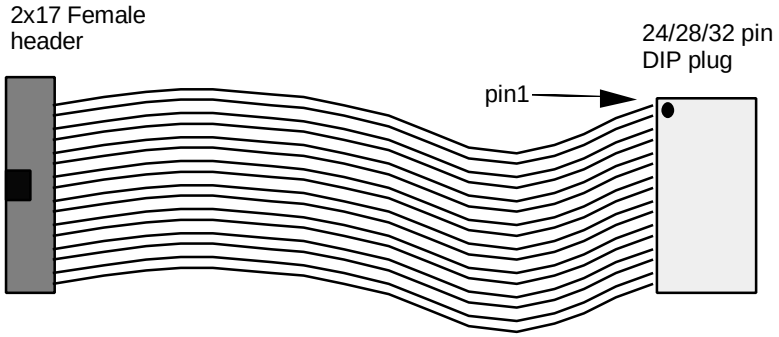
If you will be emulating a 16 bit or larger ROM, plan the byte order before connecting the target cables.

The **word** command's default byte order is: all known PromICE 8 bit units arranged in ascending unit ID order. On a dual PromICE, the lower ROM cable connector on the back of the PromICE will be assigned unit ID zero. The upper connector will be unit ID one.

If the ROM cable hookup does not match the default word order, then specify a unit IDs list in the LoadICE **word** command. The unit IDs list defines the order of bytes within the word width and depends on how the target cables are attached to the target.

Plug the cable(s) into the target system as instructed below:

Carefully locate pin 1 of the ROM socket. It is the top left pin when the notch on the socket is on the top. Connect the cable to the target ROM(s) with **pin 1** on the cable DIP aligned with **pin 1** on the ROM socket.



WARNING: If you cannot locate pin 1 on the target's ROM socket, then seek help before proceeding. DO NOT plug this cable in backwards. Doing so will damage PromICE and/or target.

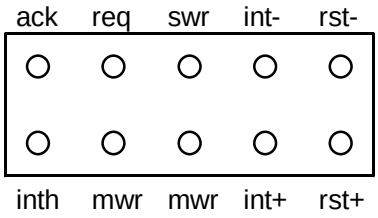
Carefully connect the other end of the cable(s) to PromICE via the keyed 34 pin female IDC connector.

Using the Reset Line

PromICE asserts the reset line automatically while downloading data (LOAD mode) and when the LoadICE **reset** command is issued.

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

The reset signals are among the auxiliary signals on the PromICE back panel.



rst- and rst+ : (tri-state outputs) These are reset signals that are driven by PromICE under LoadICE control. Both polarities of the signal are provided and are driven by a 74LS125 tri-state buffer.

Connecting the Reset Line

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.

The reset signal 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.

Note: Make sure that your target allows a shared reset. Connecting to a non-sharable reset on the target will damage the drivers in PromICE as well as on the target. Refer to your target hardware documentation.

Hardware Installation Completed

This completes the PromICE hardware installation section. Your target system should be properly connected to PromICE and PromICE should be connected to the proper host ports.

Next is the setup of LoadICE, the host software for downloading and emulating your code. It is best to check your setup by simply downloading known working code. This way you can verify that your target system and PromICE are functioning properly and that you can repeat the download/restart process.

Software Installation

The PromICE utility software is called LoadICE. You use LoadICE to load your target's files and manage the contents of emulation memory. 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.

UNIX Note: The distribution disk has LoadICE application executables for Sun Sparc-based workstations. If you have another UNIX system, then install the LoadICE sources on your machine and use the "make" command to create the LoadICE executable.

DOS / Windows 3.x / Windows 95 / Windows 98

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

Create a directory for LoadICE files, c:\loadice for example. Add this directory to your PATH= statement in c:\AUTOEXEC.BAT. Use the following to install LoadICE on your system:

```
copy a:\dos\*.*  c:\loadice
```

or

```
copy a:\windows\win9598\*.*  c:\loadice
```

Now you can execute LoadICE from anywhere in your system.

Windows NT 4.0

Create a directory for LoadICE files, c:\loadice for example. Add this directory to your PATH= statement. Open an MS-DOS Prompt window and use the following to install LoadICE on your system. For example:

```
copy a:\windows\winnt\*.*  c:\loadice
```

WIN NT Parallel Port Setup:
In order to use the parallel port for fast downloading, follow these steps to install the parallel port driver:

1. Open an MS-DOS Prompt window, and use the following to install LoadICE parallel driver installation files on your system. For example:

   ```
   cd c:\loadice
   md driver
   copy a:\windows\winnt\ntdriver\*.*  c:\loadice\driver
   ```

2. Reboot your computer and access your computer's BIOS setup screens.

3. While in BIOS setup, be sure that the parallel port is in either Standard or EPP mode. Do not use bi-directional, ECP or DMA modes.
4. While in BIOS setup, note the address of the parallel port. This is very important. Some new computers may not use the standard addresses.
5. Save any changes and exit BIOS. Reboot and login to Windows as Administrator.
6. Edit the PromICE.ini file in the directory where you installed the ntdriver files, c:\loadice\driver in the example. Change the IoPortAddress to match the address of your parallel port.
7. If necessary, edit the PromICE.bat file. This should only be necessary if you installed Windows NT system into a directory other than C:\winnt.
8. Run PromICE.bat.
9. Restart the computer.
10. From the Start Menu, select Settings->Control Panel.
11. In Devices Control Panel, locate the service called “PromICE”. It should be started and automatic.
12. In case of problems, recheck the BIOS and PromICE.ini settings then rerun PromICE.bat and reboot.

UNIX Installation

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):

On Solaris workstations:
Insert the floppy disk in the drive and do the following:

Run the file manager.

Check for the floppy.

Copy the sources to your hard disk.

Select the appropriate makefile:

.sun for old Sun OS4

.sol for Sun OS5

.unx for HP, BSD, etc.

Copy the selected file to "makefile".

Issue the make command.

On older Sun workstations:
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.
For 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 which is a generic make file.

3. Software Configuration

Contents

11. Boot your target

LoadICE Example: Loading a file

Introduction

This chapter builds upon the preceding chapter on hardware and software installation. Your configuration can be verified only if the PromICE hardware has been installed. This chapter is an introduction on how to configure PromICE to work in your environment.

You will use the LoadICE software to configure your PromICE to emulate a ROM within your target system. LoadICE configurations are usually stored in the loadice.ini file, which is processed at program startup time. The instructions for installing the LoadICE software can be found in the preceding chapter.

You can create your loadice.ini one of two ways: using the **setup** command or by editing manually.

LoadICE commands are case sensitive. All entries should be lower case unless otherwise specified. For more information on any of the following loadice.ini instructions, refer to the *Command Reference* chapter.

Comments in loadice.ini begin with asterisk "*" and hide the remainder of the line.

Note for FastPort users A serial connection is required for communication. The parallel port is optional, but allows for faster download speeds. Refer to the documentation supplied with your FastPort for installation instructions.

Using the setup command

The **setup** command allows you to create a rough draft of your loadice.ini file. Be sure to run LoadICE from the directory where you will normally use LoadICE.

Setup invokes a menu-driven interface for LoadICE/PromICE configuration. Please follow the on-screen instructions. Be sure to save your settings to the loadice.ini file before exiting the menu interface.

Once the **setup** command creates a loadice.ini file, you may need to edit it manually to make changes, or you may choose to build another file using the **setup** command. The following step by step instructions may be helpful in understanding the configuration process.

You can invoke the **setup** command on the LoadICE command line as follows:

```
loadice -setup
```

Step by Step Instructions

1. Edit or create the loadice.ini file

Create a loadice.ini file in the working directory where you will run LoadICE. LoadICE processes this file at program startup time. You may want to begin with the sample loadice.ini provided on the distribution disk.

2. Configure Host-to-PromICE communications

If you are using a serial port to communicate with PromICE, go to step 2.1.

Fastport and parallel port daisy chain users should go to step 2.3.

If PromICE is connected only via the parallel port, go to step 2.2.

If PromICE is connected via the parallel port as well as the serial, go to step 2.3.

2.1 Connected via the serial port

If you are using a serial port to communicate with PromICE, the loadice.ini file should read:

output=*comport*
baud=*bbbbb*

where

comport	is the serial port being used. On Windows, use com*n* where *n* is the port number. On UNIX, use /dev/tty*x* where *x* is a letter such as "a", "b", etc.
bbbbb	is the serial baud rate

2.2 Connected only via the parallel port

If PromICE is connected only via the parallel port, the specification should be:

pponly=*prtport*

where

prtport	is the parallel port number being used. On Windows, use lpt*n* where *n* is the port number. On UNIX, use /dev/bpp*n* where *n* is a number such as "0", "1", etc.

Note for FastPort and Daisy chain users	FastPort and daisy chain users cannot use the **pponly** command, go to step 2.3 below).

2.3 Connected via the parallel port in addition to the serial

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

output=com*n*
baud=*bbbbb*
ppbus=lpt*m*

where

n is the serial port number being used.

bbbbb is the serial baud rate

m is the parallel port number being used.

Daisy Chain Note:

When daisy chaining PromICE units, use the **output** and **baud** commands to configure the serial port. If you are also daisy chaining the parallel port, use the **ppbus** command, and optionally the **ppmode** and **number** commands.

output=com1
baud=9600
ppbus=lpt2

The **number** command defines the number of daisy chained PromICE units.

number=4

This example indicates that there are four PromICE units attached to the parallel port bus cable.

3. 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 0 for an 8 bit word using the "0" (master) PromICE unit.

OR

word=16 0 1 for a 16 bit word on a typical Intel-style target with the first byte going to the "0" (master) PromICE unit and the second byte going to the "1" (slave) unit.

OR

word=16 1 0 for a 16 bit word on a typical Motorola-style target with the first byte going to the "1" (lower) PromICE unit and the second byte going to the "0" (upper) unit.

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

4. Specify the ROM size to be emulated

The ROM size should not exceed the size of the PromICE emulation memory or the size specified in the optional **socket** command:

rom=27512 * Specify a generic ROM part number (i.e. 27512),

OR

rom=64k * Specify the ROM size in bytes.

When emulating word sizes greater than 8 bits, state the target memory size as *nnn* K by the *iii* word size. For example, an 8 megabit device in 16 bit mode is stated 512K by 16 where *nnn* is 512 and *iii* is 16. So the commands for this memory are: word=16 and rom=512k.

For a complete explanation, see the *Command Reference*.

5. Unused Address Lines

All unused pins on the ROM socket should be tied high. PromICE will not emulate properly if there are any floating address lines on the ROM socket within the PromICE address range. For more information, see the *Unused Address Lines* chapter.

6. Specify the file or files to load

You can load hex and binary files.

Hex files

LoadICE currently supports the following file formats: Intel 8 and 16 bit hex, Motorola S record format, Tektronix standard and extended hex, and the archaic Mostek and RCA formats. In HEX files, each record contains the address where the data must be loaded. The following command lets you map the hex file to a desired location within PromICE memory:

file *file_name file_address = ROM_offset*

where

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

file_address = ROM_offset
(optional, see below for individual arguments)
This expression will place data from the hex file *file_name* into PromICE memory using the difference between *file*_address and the ROM-relative address specified in *ROM_offset*. Follow *file_address* with an equal sign and *ROM_offset*.

file_address (hex) The absolute starting address of the data in 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.

For assistance in determining the *file_address* for your hex file, use the **analyze** or **setup** command.

ROM_offset (hex) A ROM-relative offset into the PromICE ROM emulation address space. The PromICE ROM emulation space begins at offset zero. This offset is used to determine the location within PromICE emulation memory where the data at *file_address* should be loaded.

You can use the **config** command to check the address range of the current PromICE emulation memory configuration.

To load multiple files, simply use multiple commands, one for each of the files to be loaded.

Binary files

To load a binary image, use the **image** command. A binary file does not contain address information but may contain a header at the beginning that will need to be omitted.

image *file_name skip_count = ROM_offset*

where

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

skip_count = ROM_offset
(optional, see below for individual arguments)
This expression will place data from the binary file *file_name* starting at file offset (skip_count + 1) into PromICE memory

at the ROM-relative address specified in *ROM_offset*. Follow *skip_count* with an equal sign and *ROM_offset*.

skip_count (hex) The count of data bytes to be skipped over from the beginning of the file. For example, binary files may contain a 14 byte header, with information for the loader, that will need to be discarded when the file data is loaded into PromICE.

ROM_offset (hex) A ROM-relative offset into the PromICE ROM emulation address space. The PromICE ROM emulation space begins at offset zero. This offset is used to determine the location within PromICE emulation memory where the beginning of the data from file '*file_name*' should be written.

Loading Multiple files

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

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

Make sure that you are not loading your files on top of one another. You can do a **compare** command, 'c', in LoadICE Dialog mode to see if any of the files are overlapping.

Sample loadice.ini files:

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

Intel-style 128K by 16 bit system, serial link to host

```
output=com1
baud=57600
rom=27010
word=16 0 1
file=mad.hex fe000=0          * Load  the  file data at  starting
                              * address  fe000 into ROM
                              * relative offset 0  in PromICE
                              * memory
```

Intel-style 512K by 8 bit system, parallel link to host

```
pponly=lpt1                   * Use  parallel port is as a
                              * bi-directional link
rom=27040
word=8
file=mad.hex fe000=0
```

Intel-style 128K by 8 bit system, dual link to host

```
output=com1                 * Use the output, baud and
baud=19200                  * ppbus  specifications if you
ppbus=lpt1                  * want to use parallel port for
                            * downloading and the serial
                            * port  for communications
rom=27256
word=8
file=mad.hex fe000=0
```

7. To initiate loading the files, use the load command

To initiate loading the files in the load set into PromICE for emulation, use the **load** command.

If you want files to load during loadice.ini processing, use the **begin** command followed by the **load** command.

8. Save your loadice.ini file in your working directory

9. Apply power to PromICE and the target system

The PromICE RUN light should lit.

10. Execute LoadICE

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

11. Boot your target

Boot your target by cycling power to the target or pressing the reset button. This is not necessary if the PromICE reset output has been connected to the target. The LOAD light on PromICE should be ON while target power is off. If you boot your target by turning on power then the LOAD light should go OFF.

If you now have a working configuration for ROM emulation, setup is complete. If you are using a debugger with PromICE AI, proceed to the AI *Configuration* chapter.

If your target does not run, the LOAD light remains on, 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 the problem persists, check your ROM configurations, file mapping and target boot process. Consult the *Troubleshooting* chapter for tips on solving common configuration problems.

LoadICE Example: Loading a file

Example of a loadice.ini file that loads a file and exits.

```
C:\LoadICE>type loadice.ini
pponly LPT1
rom 27512
word 8 0
file ai8051.hex 0=0

C:\LoadICE>loadice

LoadICE version 4.0 for Windows 95/98
(C) Copyright 1989-99 Grammar Engine Inc.

 Opening initialization file 'loadice.ini'

Connecting.. Please WAIT..
Opening Parallel Port LPT1 (@0x378)
Connecting to PromICE via the Parallel Port.
 Connection established

EMULATION UNITS PRESENT:
 PromICE ID-0 Memory=512KBytes Emulating=64KBytes FillChar=0xFF Master/AI2
 PromICE ID-1 Memory=512KBytes Emulating=64KBytes FillChar=0xFF Slave of 0
Opening file `ai8051.hex` for processing./Done
Transferred 1856 (0x740) data bytes

LoadICE Exiting with NO Errors
C:\LoadICE>
```

- For an example of a Dialog mode session, see the *Introduction* chapter.

4. Command Reference

Contents

OVERVIEW

LoadICE commands fall into four major categories, which are described below. You can locate commands by category and obtain detailed explanations in the following reference pages.

Be sure to visit **www.promice.com** for a free upgrade to the latest LoadICE software and documentation. If you don't have web access, contact your Sales Representative to obtain a copy.

Note: LoadICE version 4.0 or higher is required to use the commands documented in this version of the PromICE User Manual.

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 or a network link. These commands allow you to completely specify your communication configuration:

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

ROM specifications

These commands allow you to describe your ROM configuration. The number, size and arrangement of ROMs must be specified.

The target address mask may need adjustments to account for differences between PromICE memory size, target socket size, and emulation ROM sizes.

rom	Set size of ROM to be emulated
word	Set word size for ROMs to be emulated

socket	Modify address mask, if needed
xmask	Set arbitrary address mask, if needed
checksum	ROM checksum specifications
fill	ROM fill specification

ROM operations

These commands allow you to perform operations on the ROM emulation data.

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

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:

analyze	Report on address range of hex file
bank	Bank emulated memory
file	Add hex data file to set of files to load
image	Add image data file to set of files to load
load	Download 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	Save emulation memory contents to a file

Frequently Used Dialog commands

These commonly used commands provide interactive control of PromICE.

stop	Stop PromICE emulation, enter Load mode

go	Begin PromICE emulation, leave Load mode
config	Display PromICE configuration information
status	Display target status (power on; executing)
file	Add hex data file to set of files to load
image	Add image data file to set of files to load
load	Download the data files
exit	Exit LoadICE Dialog mode

Miscellaneous commands

begin	In loadice.ini, initiates communication with PromICE and processes each remaining "ini" command before continuing.
dialog	Enter Dialog mode on startup
display	Change output level detail
help	On-line help
log	Record all LoadICE command traffic to a log file
reset	Define reset duration and assert reset signal on back panel
ver	Report LoadICE and PromICE micro-code versions
hso	Define the operation and polarity of the interrupt signal
notimer	Disable PromICE internal timer
fkey & altfkey	Assign commands to function keys
! (system)	Escape commands to the host shell
delay	Change the time out period used by LoadICE
sleep	Use when waiting for something to complete in batch mode
test	Test emulation memory

General Listing of LoadICE Commands

The general-purpose LoadICE Commands are summarized in the following list and explained in detail in the rest of this chapter. Some PromICE options, such as AI, Trace, and Code Coverage, have their own command sets. These option commands are documented in their respective chapters.

. (switch)	Bypass the loadice.ini file from the command line.
! (shell)	Escape command to DOS or UNIX shell.
@filename	Use LoadICE configuration file *filename*, available on command line.
afn	Allows assigning hot keys to LoadICE or host commands.
analyze	Report addresses contained in a hex or image record file.
bank	Allows the emulation space to be broken into a number of banks.
baud	Specify serial baud rate between PromICE and the host.
begin	In loadice.ini, this causes LoadICE to shift into an interactive mode.
checksum	Checksum on ROM and store result in PromICE emulation memory.
clearfiles	Clears the current load set - the set of files to be loaded.
compare	Compares data loaded in PromICE against files on the host.
config	Display the current PromICE configuration.
cursor	Controls display of the LoadICE Busy/Progress indicator.
delay	Set the timeout period for PromICE to respond.
dialog	Enter LoadICE Dialog mode.
display	Change the output display level of LoadICE.
dump	Display contents of PromICE emulation memory on the screen.
edit	Modify PromICE ROM emulation memory.
ethernet	Defines the type of Ethernet print server to use.
exit	Exit LoadICE when in Dialog mode.
fast	Lengthens the strobe on the host parallel port.

fastport	Configure PromICE to connect via a FastPort.
file	Setup a hex record file for ROM emulation or a comparison operation.
fill	Fill the current PromICE ROM configuration with a data pattern.
fillall	Fill the entire PromICE memory capacity with a data pattern.
find	Find a binary data pattern in PromICE emulation memory.
fn	Assign hot keys to LoadICE or host commands.
go	Instruct PromICE to begin emulating.
help	Obtain help about a LoadICE command.
hso	Program the interrupt signal to the target (on PromICE back panel).
image	Setup a binary file for ROM emulation or a comparison operation.
load	Initiates loading files into PromICE memory for emulation.
log	Record all LoadICE command traffic to a log file in real-time.
map	Control the address range display during the loading process.
modein	Specifies the initial mode for PromICE upon LoadICE invocation.
modeout	Specifies the final mode for PromICE upon LoadICE termination.
modefixed	Maintains the current PromICE mode at LoadICE termination.
move	Copy bytes within PromICE emulation memory.
noaddrerr	Ignore address-out-of-range errors during file loading.
number	Specify the number of Daisy Chained PromICE units.
output	Use the serial output port for connection with PromICE.
ppbus	Use the parallel and serial ports for PromICE communication.
ppmode	Sets the parallel port's communication mode.
pponly	Use only the parallel port in bi-directional mode for communication.
promiceid	Display the PromICE Identification number.
reset	Initiate a target reset and the duration of target reset signal.

resetfp	Control whether LoadICE resets the FastPort before connecting to PromICE.
restart	Restart the LoadICE to PromICE communication link.
rom	Specify ROM emulation memory size.
save	Save PromICE emulation memory contents to a binary file on the host.
search	Search PromICE emulation memory for an ASCII data pattern.
setup	Access a menu-driven tool for creating loadice.ini.
socket	May be needed when unused address lines cannot be pulled high.
status	Displays the status of the target system as detected by PromICE.
stop	Cause PromICE to stop emulating and enter Load mode.
test	Test PromICE emulation memory on a particular PromICE 8 bit unit.
version	Report micro code version of the PromICE and the LoadICE version.
word	Set data word width for the PromICE ROM emulation configuration.
xmask	May be needed when unused address lines cannot be pulled high.

. (period, command line)

Bypass the loadice.ini file.

Command Forms

. Command line

Syntax

. [***args***]

where

. On the command line, bypass the loadice.ini file.

args (optional) Additional command line switches (see note).

Default

When this directive is used by itself, LoadICE will fail to connect PromICE and will prompt you to use a **setup** or **restart** command.

Description

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

Notes

When bypassing the loadice.ini file, define the PromICE configuration using 1) the ***@filename*** switch to specify another INI file or 2) the **-d** switch to invoke the interactive Dialog mode or 3) specify all configuration settings using switches. Otherwise, LoadICE will terminate without configuring PromICE.

Examples

loadice . -q lpt1 -d

Bypass the loadice.ini file. Connect to PromICE using the parallel port *LPT1* only (refer to the **pponly** command in this chapter for more information on the parallel port option). Enter Dialog mode.

! (shell)

Escape command to DOS or UNIX shell.

Command Forms

!	Dialog mode

Syntax

! *string*

where

string	The command string to be executed by the operating system shell. Place double quotes around commands containing a comma.

Default

None.

Description

This command allows you to execute arbitrary operating system commands without leaving LoadICE. After the command completes, control is returned to LoadICE. To invoke a new shell, use “command” on Microsoft systems, and “sh” or “csh” on UNIX systems.

Examples

```
!dir
```

Show the files in current directory.

```
!edit myfile
```

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

@filename

Use LoadICE configuration file *filename*, available on command line.

Command Forms

@filename Command line

Syntax

@file_name

where

file_name The name of the configuration file to use. Full directory path names can be specified.

Default

LoadICE will automatically look for the file *loadice.ini* in the current directory. When the @ directive is used, LoadICE will search for the file as specified. If *file_name* is not found and *file_name* lacks an extension then LoadICE will search for *file_name* with an .ini extension.

Description

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

Notes

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

Examples

loadice @c:\mycfg\myini.ini

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

afn

Allows assigning hot keys to LoadICE or host commands.

Command Forms

afn	loadice.ini file
-afn	Command line
afn	Dialog mode

Syntax

afn#=[***cmd***]

where

#	(decimal number) alt-function key number
cmd	A single LoadICE command with arguments.

Default

No keys are assigned.

Description

The **afn** command allows you to assign LoadICE commands or system commands to function keys. For example, you can edit and compile your source file without exiting LoadICE by assigning command strings to function keys.

Notes

Assign single word commands directly but enclose multiple word commands in double-quotes. Within multiple word commands, surround quoted strings with the quote escape, \". Operating system commands must be preceded by a '!'.

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.

analyze

Report on range of addresses contained in a hex or image record file.

Command Forms

analyze Dialog mode
an Dialog mode

Syntax

analyze ***file_name***

where

file_name (string) The name of the hex or image record file to be analyzed.

Description

The **analyze** command allows you to obtain address information from a hex record file. To display a further breakdown of the file address information, use the **config** command. This breakdown will be cleared during the processing of the next **load** or **analyze** command.

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

Examples

analyze myfile.hex

Report on addresses of the data in *myfile.hex*. For example, the resulting display would be:

```
file myfile.hex

Opening file `myfile.hex` for processing/Done

File Data Spans from 0x000A0000 to 0x000FFFF4
 ROM Space Spans from 0x00000000 to 0x0007FFFF
```

The file *myfile.hex* contains data from 0xA0000 to 0xFFFFF. The current PromICE ROM configuration supports 0x80000 bytes of emulation memory. As a result, you may want to specify the file command as:

```
file myfile.hex A0000=0
```

bank

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

Command Forms

bank loadice.ini file
bank Dialog mode

Syntax

bank ***num***

where

num First use defines the number of banks. Subsequent **bank** commands switch to bank number *num*.

Default

Banking is disabled.

Description

This command is used to divide ROM emulation memory into banks. The first occurrence of the **bank** command defines the number of banks. Additional **bank** commands select a bank as the current bank for subsequent commands.

Notes

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

Examples

This example loads files into multiple banks from loadice.ini file:

```
file=mon.hex    * file to load into multiple banks
bank  4         * divide ROM into 4 banks
bank  1         * select bank one
begin           * begin communication with PromICE
load            * load file here
bank  3         * select another bank
load            * load same file here
clearfiles      * reset file history
file = app.hex  * specify new file
bank  2         * select bank
load            * load new file at end of .INI processing
```

baud

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

Command Forms

baud	loadice.ini file
-b	Command line

Syntax

baud ***baud_rate***

where

baud_rate	(integer) A valid baud rate: 1200, 2400, 4800, 9600, 19200, or 57600.

Default

57600 on Windows/DOS and 19200 baud on Unix.

Description

The **baud** command specifies the baud rate for a particular serial port.

Notes

UNIX can use up to 19200 baud.

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

Examples

-b 57600
baud 57600

Sets the baud rate of the specified device to 57600.

begin

During loadice.ini command processing, this command causes LoadICE to shift command processing into a more interactive mode of operation.

Command Forms

begin loadice.ini file

Syntax

begin

Description

The **begin** command causes LoadICE to immediately initiate communication with PromICE and treat the remaining commands in loadice.ini one at a time, much like Dialog mode commands.

Without the **begin** command, LoadICE initiates communication with PromICE after loadice.ini processing has finished. The commands in loadice.ini serve as configuration settings, not as a general-purpose scripting language.

For more information on the use of the **begin** command, see the *LoadICE Environment* chapter.

Notes

All commands that configure Host-to-PromICE communications should precede the **begin** command, which initiates LoadICE communication with PromICE.

The **begin** command will cause command line switches to be processed, which allows them to override commands in loadice.ini prior to **begin**.

Commands following **begin** do not have INI Command level scope.

Examples

```
begin
load
```

In loadice.ini, this causes LoadICE to initiate communication with PromICE and immediately load the current set of load files.

checksum

Perform checksum on ROM and store result in PromICE emulation memory.

Command Forms

checksum	loadice.ini file, Dialog mode
-k	Command line
k	Dialog mode

Syntax

checksum ***start end store*** [***function***] [***sum_size***] [***order***]

where

start (hex) *start* is the beginning address of the range of PromICE memory for checksum computation. *start* is a PromICE ROM-relative address.

end (hex) *end* is the ending address of the range of PromICE memory for checksum computation. *end* is a PromICE ROM-relative address

function (optional 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 whereas small x stores the result as is. A capital "A" causes a 1's complement of the addition be stored and a lower case "a" causes a 2's complement to be stored.

sum_size (optional 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 PromICE units.

order (optional digit **0** or **1**) checksums over 8 bits wide are stored high byte first, unless this argument is supplied and is '1', then the low byte is stored first.

Default

You must specify the start, end and store arguments for the **checksum** command. The default for *function* is '*x*' and *sum_size* is 8.

Description

Compute and store an 8, 16 or 32 bit checksum in the ROM emulation memory. The checksum is computed inclusive of the *start* and *end* PromICE ROM-relative addresses within the current configuration. The results are stored in the given *store* address. The checksum is also displayed.

Notes

Larger ROM checksums will take longer to compute.

You can specify separate checksums for each ROM (See the *PromICE Unit IDs* chapter).

Examples

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

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

clearfiles

Clears the current load set - the set of files to be loaded.

Command Forms

clearfiles loadice.ini file, Dialog mode
cf Dialog mode

Syntax

clearfiles

Description

This command empties the current load set. The **file** and **image** commands build the load set one file at a time. When the **load** command is issued, whatever files are contained in the load set are sent to the PromICE for emulation.

The **clearfiles** command is useful after some files have been loaded, but it is necessary to load a different group of files. Also in Dialog mode, experimentation with the **file** and **image** commands may lead to invalid entries in the load set. These can be deleted with the **clearfiles** command.

Notes

In versions of LoadICE prior to version 4.0, the period command prefix, used on the "**.file**" and "**.image**" commands, was used to empty the current load set.

Examples

cf

Clear the current set of files to be loaded.

compare

Compares data loaded in PromICE against files on the host.

Command Forms

c Dialog mode

Syntax

c

Description

The **compare** command will compare contents of the data files in the files to load list with the contents of PromICE emulation memory. Use compare to obtain an explicit verification of a successful download. Differences are displayed on the screen and there is an option to continue displaying the differences or canceling the compare.

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 the target could also have 'written' to a location in the emulation memory and thus caused the compare failures.

If the unit is failing to compare after a load and none of the above problems are the cause, there may be problems with the PromICE emulation memory.

Examples

c

Compare now.

config

Display the current PromICE configuration.

Command Forms

config	loadice.ini file, Dialog mode
C	Dialog mode

Syntax

config [***link*** | ***rom*** | ***file*** | ***all***]

where

link Displays diagnostic information about the communication link (serial or parallel)

OR

rom Displays diagnostic information about the ROM configuration

OR

file Displays diagnostic information about the list of files loaded into PromICE

OR

all Displays all diagnostic information

Default

all

Description

The **config** command 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 modes.

Notes

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

Examples

C all

Display the entire configuration.

The following is sample output from the **config** command in Dialog mode following a **load** command:

```
00:LOADice: config

HOST TO EMULATOR CONNECTION
 Parallel link: LPT2(@0x278) Turbo-mode Bus-mode Down-load only
output=COM2     Serial link: COM2(@0x0) @57600 baud

OPTIONS IN EFFECT:
 Data verification via block-checksum is OFF

EMULATION UNITS PRESENT:
 PromICE ID-0 Memory=512KBytes Emulating=512KBytes FillChar=0xFF Master/AI2

FILE-1  name = "test.hex"       type = Intel hex
        Offset = 0xFFF80000     Skip = 0x0
        Use DEFAULT ROM CONFIGURATION for down-load

DEFAULT ROM CONFIGURATION
         word = 8
         Emulation Space: TOTAL Address Range = [0x0->0x7FFFF]
          IDs = 0 Address Range = [0x0->0x7FFFF]

Down-loaded Data Map
Data 0 - 219
Data 21c - 22b
Data 230 - 273
Data 280 - 28f
Data 7fc00 - 7fcdd
Data 7fff0 - 7fff4

00:LOADice:00:LOADice: exit

LoadICE Exiting with NO Errors

C:\LoadICE>type loadice.ini
output com2
baud 57600
ppbus lpt1
word 16
rom 27010
file test.hex 80000=0
dialog
```

cursor

Controls the display of the LoadICE Busy/Progress indicator.

Command Forms

cursor	loadice.ini file, Dialog mode
-cu	Command line

Syntax

cursor ***mode***

where

mode (integer) A number indicating the display mode for the LoadICE Busy/Progress indicator as follows:

0 By default, periods are gradually displayed to indicate progress.

1 Displays a spinning cursor (character-based).

2 Suppress all Busy/Progress indications.

Default

The default mode is zero, which periodically displays periods to indicate progress.

Description

The **cursor** command controls the display of Busy/Progress for lengthy LoadICE commands.

Examples

cursor 2

Suppress all Busy/Progress indicators.

delay

Change the timeout period used by LoadICE when it is waiting for a response from PromICE.

Command Forms

delay loadice.ini file, Command line

Syntax

delay [***num***]

where

num (decimal) Optionally, a multiplier that results in *num* times 4 seconds of delay. A *num* of zero results in zero delay (wait indefinitely). No argument restores the default.

Default

Delay 4 seconds.

Description

The **delay** 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.

Notes

While executing certain commands where the response from PromICE may take a variable amount of time, LoadICE internally disables the delay. For example, when executing the **test** command to test PromICE memory, timeout is disabled.

Examples

delay 8

Wait 32 seconds before timing out (8 times 4).

dialog

Enter LoadICE Dialog mode.

Command Forms

dialog	loadice.ini file
-d	Command line

Syntax

dialog

Description

This command will force LoadICE to enter interactive Dialog mode upon finishing loadice.ini file processing. In Dialog mode, you can enter LoadICE commands and see the results interactively.

For more information on using Dialog mode, see the introduction in the *LoadICE Environment* chapter.

Notes

Upon entering Dialog mode, PromICE will be in Emulation mode by default. Use the **stop** command to place PromICE into Load mode. LoadICE commands that access emulation memory are only available in Load mode.

If you specify files to be loaded in your loadice.ini file along with the **dialog** command, LoadICE will enter Dialog mode **without** loading the files. To load the files before entering Dialog mode, modify loadice.ini and place the following commands before the **dialog** command: the **begin** command followed by the **load** command.

Without a **dialog** command in loadice.ini, LoadICE will load user data files according to the configuration defined in loadice.ini and then exit. However, if no data files are specified, LoadICE will simply exit.

Examples

loadice -d

Invoke LoadICE application and enter the Dialog mode.

```
pponly=lpt1
word=8
rom=27040
file=loaddata.hex 8000=0
begin
load
dialog
```

This loadice.ini file specifies a file to be loaded over the parallel port and then enter Dialog mode. The **begin** command initiates communication with PromICE. Then the **load** command will process immediately.

display

Change the output display level of LoadICE.

Command Forms

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

Syntax

display [***level***]

where

level (hex digits) Optionally, A valid number (0 - FF) indicating the level of diagnostic display desired.

The bits defined within *level* 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 as they go over the link

Default

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

Description

The **display** command displays information about LoadICE processing at the level of detail that is specified by the bits in *level*.

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.

Examples

-D fe

Display everything but the actual data going over the link.

dump

Display contents of PromICE emulation memory on the screen.

Command Forms

dump	Dialog mode
d	Dialog mode

Syntax

d [***start*** [***end***]]

where

start	(hex) Optionally, the first PromICE address to display
end	(optional hex if *start* is defined) If *end* is greater than *start*, then *end* is the last PromICE address to display. If *end* is less than or equal to *start*, then *end* is number of bytes to display.

Default

Use in the current configuration. The default for start address is 0 and the end will equal start + 64 (or the last valid address in ROM).

Description

The **dump** command will display PromICE emulation memory data, 16 bytes per line, in hex and ASCII. Data is displayed according to the current PromICE ROM configuration word size. A <CR> will simply repeat the command with the next range of arguments. Optionally, display data either 1) from an address range or 2) from a base address for a byte count.

Notes

If the second argument, *end*, is less than or equal to *start*, then *end* is the number of bytes to display.

Examples

```
d 10 ff
```

Dump data from the 0x10 to 0xFF ROM-relative address range.

```
d 1000 ff
```

Dump data from the 0x1000 to 0x10FF ROM-relative address range.

edit

Modify PromICE ROM emulation memory.

Command Forms

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

Syntax

edit [***address*** { ***value*** }]

where

address	(optional hex) Address in PromICE emulation memory to modify.
value	(if *address* is defined, zero or more hex values) A value or list of data values to be written to emulation memory starting at *address*, according to the current word width (defined by the **word** command, default size is 8 bits).

Default

Start interactive editing at address zero of the current configuration.

Description

The **edit** command can change one or more data bytes within the emulation memory defined in the current ROM configuration.

Used in the loadice.ini file or the command line, the patch data is written after the first load of the files has completed. Then the patches expire. **Note:** Subsequent loads will not be patched.

In Dialog mode, edit will enter an interactive mode if no address or no data values are specified. To stop editing in Dialog mode, type a period followed by <enter>. This will return you to the LoadICE prompt.

Examples

edit 500 ab cd

Set locations 0x500 and 0x501 to values 0xAB and 0xCD.

ethernet

Defines the type of Ethernet print server to use. Also specifies the host name to use on the network to communicate with a print server that has a PromICE attached.

Command Forms

ethernet	loadice.ini file, Dialog mode
-et	Command line

Syntax

ethernet = fastport *hostname*

OR

ethernet = microplex *hostname*

where

fastport	(optional literal) Use a FastPort print server. Follow with an equal sign and a *hostname*.
microplex	(optional literal) Use a Microplex print server. Follow with an equal sign and a *hostname*.
hostname	The name of the host for the print server on the network. In UNIX it is generally kept in the /etc/hosts file.

Description

The **ethernet** command configures access to a PromICE over a network by means of a print server. These servers support serial and parallel printers on Ethernet networks. During installation, the PromICE unit is attached to the print server's serial port and optionally the printer port. Once configured, LoadICE can access the PromICEs over the network. Full support for the PromICE AI option is included.

Notes

FastPort Users:

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

Microplex Users:

The Microplex is a product made by Microplex Systems Ltd. Follow the installation instructions that come standard with the Microplex**. Do not install the printer server software.** Follow the instructions supplied by Grammar Engine for configuring the Microplex for use with PromICE. Currently the Microplex can use only the PromICE serial port.

Examples

ethernet = fastport mypromice

Communicate to FastPort "mypromice" in loadice.ini file.

-et = microplex mypromice

Specify Microplex "mypromice" print server on the LoadICE command Line.

exit

Exit LoadICE when in Dialog mode.

Command Forms

exit	Dialog mode
x	Dialog mode
quit	Dialog mode

Syntax

exit [***num***]

where

num	(optional integer) Set the exit code for LoadICE.

Default

Exit code is “0” if all goes well. If an error is encountered then the exit code is “1”.

Description

The **exit** command will terminate the LoadICE application.

Notes

When running LoadICE in batch mode, the exit code can be used to determine if an error occurred.

Examples

exit

Exit LoadICE application.

fast

If parallel port transfers fail, use this command to lengthen the strobe on the host parallel port. Use on high performance hosts.

Command Forms

fast loadice.ini file

Syntax

fast [***num***]

where

num (optional integer) Lengthen the parallel port strobe by a factor of *num.*

Default

The default value for the delay loop count is 10 (see below).

Description

The parallel port strobe is generated by turning it ON and then OFF. If the **fast** command is specified then a delay loop is inserted while strobe is asserted. Optionally, this command allows you to control the length of the strobe by specifying a count for the delay loop.

This delay loop is only activated by using this command. Otherwise the length of the strobe depends on how long it takes for the host computer to execute the instructions that drive the strobe signal.

Examples

fast 20

Pace a 100MHZ RISC host computer to work with a PromICE parallel link.

fastport

Defines the host name to be used to communicate over a network with a PromICE connected via a FastPort.

Command Forms

fastport	loadice.ini file
-fp	Command line

Syntax

fastport ***hostname***

where

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

Description

In order to access a PromICE over a network, use the **fastport** command to configure access to a FastPort print server. For complete details, see the *FastPort Configuration for PromICE* instructions (not included in this manual).

The FastPort print server supports serial and parallel printers on Ethernet networks. During installation, the PromICE unit is attached to the FastPort's serial port and optionally the printer port. Once configured, LoadICE can access the PromICEs over the network. Full support for the PromICE AI option is included.

Notes

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

Examples

fastport=mypromice

Specify FastPort "mypromice" in the loadice.ini file.

-fp mypromice

Specify FastPort "mypromice" using LoadICE command line.

file

Setup a hex record file for ROM emulation or a comparison operation. Multiple **file** commands will accumulate into a load set.

To initiate loading the files belonging to the load set into PromICE for emulation, use the **load** command.

Command Forms

file	loadice.ini file, Dialog mode
file_name	Command line, preceding all other command line switches. The command options shown below are fully supported.

Syntax

file ***file_name***

OR

file ***file_name file_address = ROM_offset***

OR

file ***file_name file_address = ROM_offset*** [**(** ***range_start, range_end*** **)**]

OR

file ***file_name file_address = ROM_offset*** [***word_width unit_ID*** { ***unit_ID_n*** }] [**(** ***range_start, range_end*** **)**]

where

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

file_address = ROM_offset
(optional, see below for individual arguments)
This expression will place data from the hex file *file_name* into PromICE memory using the difference between *file*_address and the ROM-relative address specified in *ROM_offset*. Follow *file_address* with an equal sign and *ROM_offset*.

file_address (hex) The absolute starting address of the data in 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.

For assistance in determining the *file_address* for your hex file, use the **analyze** or **setup** command.

ROM_offset (hex) A ROM-relative offset into the PromICE ROM emulation address space. The PromICE ROM emulation space begins at offset zero. This offset is used to determine the location within PromICE emulation memory where the data at *file_address* should be loaded.

word_width (optional integer with variable length integer list) defines the word width for the file. It must be an integral multiple of 8 and cannot be larger than the maximum data bus width determined by the total number of PromICE 8 bit units. *word_width* is followed by a list of one or more unit IDs, which specifies the byte order of the PromICE 8 bit units within the word width.

unit_ID (integer, required if *word_width is present*) The first PromICE 8 bit unit number corresponding to the byte at the lowest memory address within the word.

unit_ID_n (optional integers following *unit_ID*) The nth PromICE 8 bit unit number specifying the byte order within the word size specified by *word_width* for this file configuration.

(range_start , range_end)
(optional hex address range) transfer data only from address *range_start* through address *range_end* inclusive. This specification allows partial loading of data files. The parentheses and comma are required.

Note: *range_start* and *range_end* are hex file addresses. For assistance in determining the address range for your hex file, use the **analyze** command.

Default

Load the file *file_name* using the current system configuration starting at PromICE relative address zero. This is equivalent to **file** command "file *file_name* 0=0". In order to work, the data in the file must be located at zero.

Description

The **file** command allows you to specify a hex record filename to be loaded and a configuration for how it will be downloaded to PromICE.

Optionally, this command can specify how the data in the file will be mapped or relocated into the ROM emulation memory, especially for Intel-style targets with high memory ROMs. The word width of the file data, as well as the assignment of PromICE 8 bit units to the bytes within the word can be specified. Partial loading from an address range in the data file can also be specified.

To load multiple files in the loadice.ini file or in Dialog mode, add a separate **file** command for each file to be loaded. Multiple **file** and **image** commands accumulate into a load set. Issuing the **clearfiles** command will clear the existing load set.

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

Notes

Only one partial loading address range can be specified per file command.

Examples

file myfile.hex
file myfile.hex 0=0

Load data from '*myfile.hex*' with addresses starting at 0x0 in the file to be mapped to 0x0 in PromICE emulation memory, according to the current LoadICE ROM configuration.

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. The file contains 16 bit data with the first byte (even byte address) to be written to PromICE unit ID-1 and the second byte (odd byte address) to be written to PromICE unit ID-0.

file=myfile.hex 400000=0 (401000,402000)

Load data from '*myfile.hex*' from the address range starting at 0x401000 through 0x402000 in the file to be mapped into PromICE emulation memory starting at to 0x1000.

fill

Fill the current PromICE ROM configuration memory with a repeating data pattern.

Command Forms

fill	loadice.ini file, Dialog mode
-f	Command line
f	Dialog mode

Syntax

fill [***data***]

OR

fill ***start end***

OR

fill ***start end data***

OR

fill ***start end data fill_size***

OR

fill ***start end data*** [***data2***] ***fill_size***

where

start	(optional hex) The first PromICE ROM-relative address to fill.
end	(optional hex) The last PromICE ROM-relative address to fill.
data	(optional hex) The data pattern with which to fill the address range. The width of *data* can be up to four bytes as determined by the current word size or *fill_size*, if specified (see below).
data2	(optional hex) If fill pattern size is larger than 4 bytes, *data2* can specify up to 4 additional bytes of the fill pattern.
fill_size	(optional decimal) The fill pattern size, with a range of 1 to 8 bytes, overrides the current word size.

Default

Fill all memory in the current PromICE ROM configuration with the default fill character (0x0FF). All fill pattern data supplied is interpreted according to the current word size, unless *fill_size* is specified.

Description

Used to write a fill pattern into all or part of emulation memory in the current PromICE ROM configuration. The fill pattern data will conform to the current word configuration (8, 16, 32 or 64 bits where 64 bit data is specified as two 32 bit items). Optionally, you can load patterns 1 to 8 bytes in length regardless of the current configuration.

Notes

When specified in the loadice.ini file, the fill is done prior to every file load throughout that LoadICE session.

Examples

```
f 200 300 ab
```

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

```
word 16
f 200 300 abcd
```

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

fillall

Fill the entire PromICE memory capacity with a repeating data pattern.

Command Forms

fillall	loadice.ini file, Dialog mode
-fall	Command line
fall	Dialog mode

Syntax

fillall [***data***]

OR

fillall ***data fill_size***

OR

fillall ***data*** [***data2***] ***fill_size***

where

data	(optional hex) The data pattern with which to fill the address range. The width of *data* can be up to four bytes as determined by the current word size or *fill_size*, if specified (see below).
data2	(optional hex) If fill pattern size is larger than 4 bytes, *data2* can specify up to 4 additional bytes of the fill pattern.
fill_size	(optional decimal) The fill pattern size, with a range of 1 to 8 bytes, overrides the current word size.

Default

Fill all memory in the entire PromICE memory capacity with the default fill character (0x0FF). All fill pattern data supplied is interpreted according to the current word size, unless *fill_size* is specified.

Description

Use the **fillall** command to write a fill pattern into the entire PromICE memory capacity, regardless of the ROM size. Individual PromICE 8 bit units or the entire memory capacity can be filled. The fill pattern data will conform to the current word configuration (8, 16, 32 or 64 bits where 64 bit data is specified as two 32 bit items). Optionally, you can load patterns 1 to 8 bytes in length regardless of the current configuration.

Notes

When specified in the loadice.ini file, the **fillall** is done prior to every file load throughout that LoadICE session.

Examples

fillall ab

Fill all PromICE memory with data value 0xAB

word 16
fillall abcd

Fill all PromICE memory with the word data value 0xABCD

find

Find binary data pattern in PromICE emulation memory.

Command Forms

F Dialog mode
find Dialog mode

Syntax

find ***start end find_count data_byte*** { ***data_bytes*** }

where

start (hex) The first PromICE ROM-relative address to search.

end (hex) The last PromICE ROM-relative address to search.

find_count (integer) The number of bytes to find, from 1 to 32.

data_byte { *data_bytes* }
(variable length list of hex numbers) This list of hex bytes defines the byte pattern for the search. The number of bytes must match the *find_count* (see above). The list consists of a *data_byte* followed by zero or more additional *data_bytes*. The data must be specified as hex bytes separated by spaces.

Default

All parameters must be specified. If both *start* and *end* are equal to zero, then the **find** command scans the entire address range of the current PromICE ROM configuration.

Description

The **find** command searches the current PromICE ROM configuration for a binary data pattern.

Examples

F 0 1ffeo 4 de ad fe ed

Looks for the byte pattern 0xDE 0xAD 0xFE 0xED in a 128kB ROM space, in the ROM-relative address range 0x0 to 0x1FFE0. For example:

0000021C: DEADFEED
00000340: DEADFEED

fn

Assign hot keys to LoadICE or host commands.

Command Forms

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

Syntax

fn#= [***cmd*** | ***"string"***]

where

#	(decimal number) function key number (1-12) No space precedes *#*. Follow with an equal sign.
cmd | *"string"*	LoadICE or system shell command. Precede the system commands with a '!' so that LoadICE will pass them to the DOS or UNIX shell. Use double quotes if the command text contains a comma.

Description

The **fn** command allows you to assign LoadICE commands or operating system commands to function keys.

Notes

You can assign most commands directly, but enclose commands that contain a comma in double-quotes. Operating system commands must be preceded by a '!'. On the command line, the quote mark must be specified as \".

Examples

fn12=restart

This will issue the **restart** command to LoadICE when the 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 execute the operating system command '*edit*' to edit the file '*test.c*'. When you exit the editor, you will return to the LoadICE prompt.

go

Instruct PromICE to go into emulation mode.

Command Forms

go	Dialog mode
go	loadice.ini file
<ESC>	Dialog mode (toggles Emulation/Load modes)

Syntax

go

Description

Allows you to start emulation from the Dialog mode. The load light on the front panel of PromICE indicates the emulation mode. When the light is on, then PromICE is in Load mode.

When the **go** command is executed, the load light should go out, indicating that PromICE is now in emulation mode. If it does not, the PromICE is not sensing power on the target ROM socket.

Notes

You can use the [ESC] key to toggle between Load and Emulation modes.

Examples

<ESC>

PromICE will toggle between LOAD and EMULATE modes.

help

Obtain help about a LoadICE command.

Command Forms

?	Command line
help	Dialog mode

Syntax

help [***command***]

where

command	(optional string) Any valid LoadICE command name.

Default

Display the list of commands for which help is available.

Description

The **help** command provides on-line help for most commands. When used alone on the command line, '?' will give you help on all the commands that can be used 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 individual commands.

Notes

Full documentation is available in this manual and on our web site at www.promice.com.

Examples

help find

In Dialog mode, this will give information on how to use the **find** command.

hso

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

Command Forms

hso	loadice.ini file
-I	Command line

Syntax

hso [***bits***]

where

bits (optional hex) 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. At startup lower-raise-lower it.
- 5 - Interrupt is low asserted. At startup raise-lower-raise it.
- 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 start up lower it for 1 second, then raise the interrupt line.

Default

The signal is low asserted.

Description

The **hso** command allows the host system to alert the target that LoadICE is talking to PromICE.

Notes

This command allows you to program the polarity of the **int** (handshake out) signal on the back of (older) PromICE units.

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

Examples

hso 1

Defines the interrupt to the target to be high asserted.

image

Setup a binary file for ROM emulation or a comparison operation. Multiple **image** commands will accumulate into a load set.

To initiate loading the files belonging to the load set into PromICE for emulation, use the **load** command.

Command Forms

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

Syntax

image ***file_name***

OR

image ***file_name skip_count = ROM_offset***

OR

image ***file_name skip_count = ROM_offset*** [***word_width unit_ID*** { ***unit_ID_n*** }] [**(** ***start_offset, end_offset*** **)**]

where

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

skip_count = ROM_offset
(optional, see below for individual arguments) This expression will place data from the binary file *file_name* starting at file offset (skip_count + 1) into PromICE memory at the ROM-relative address specified in *ROM_offset*. Follow *skip_count* with an equal sign and *ROM_offset*.

skip_count (hex) The count of data bytes to be skipped over from the beginning of the file. Binary files may contain a 14 byte header that may need to be skipped, such as with the UNIX a.out format.

ROM_offset (hex) A ROM-relative offset into the PromICE ROM emulation address space. The PromICE ROM emulation space begins at offset zero. This offset is used to determine the location within PromICE emulation memory where the beginning of the data from file 'file_name' should be written.

word_width (optional integer with variable length integer list) defines the word width for the file. It must be an integral multiple of 8 and can not be larger than the maximum data bus width determined by the total number of PromICE 8 bit units. word_width is followed by a list of one or more unit IDs, which specifies the byte order of the PromICE 8 bit units within the word width.

unit_ID (integer, required if word_width is present) The first PromICE 8 bit unit number corresponding to the byte at the lowest memory address within the word.

unit_ID_n (optional integers following *unit_ID*) The nth PromICE 8 bit unit number specifying the byte order within the word size specified by *word_width* for this file configuration.

(*start_offset*, *end_offset*) (optional hex address range) transfer data only to the PromICE ROM-relative address range *start_offset* through address *end_offset* inclusive. This specification allows partial loading of data files. The parentheses and comma are required.

Note: *start_offset* and *end_offset* are PromICE ROM-relative addresses. For assistance in determining the address range for your image file, use the **analyze** command.

Default

Load the file in its entirety starting at zero in PromICE configuration. This is equivalent to “image *file_name* 0=0 “

Description

The **image** command allows you to specify a binary data filename to be loaded and a configuration for how it will be downloaded to PromICE.

This command can specify if the whole file should be loaded or if any bytes need to be skipped at the start of the file. The word size and byte order in the file can also be specified. Partial loading from an address range in the data file can be specified.

To load multiple files in the loadice.ini file or in Dialog mode, add a separate **image** command for each file to be loaded. Multiple **image** and **file** commands accumulate into a load set. Issuing the **clearfiles** command will clear the existing load set.

Notes

When loading multiple binary files you must specify multiple ***ROM_offsets*** or else the files will overlap each other in emulation memory.

Only one partial loading address range can be specified per **image** command.

Examples

image=myfile.bin 0=10000 16 0 1

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

image=myfile.bin 20=0

Load data from *myfile.bin* with addresses starting at 0x21 in the file to be mapped to 0x0 into emulation memory.

image=myfile.bin E=0 (1000,2000)

Load binary data from *myfile.bin* starting at 0x100F from the beginning of the file to be mapped into emulation memory starting at PromICE ROM-relative address 0x1000 through 0x2000.

load

Initiates loading files into PromICE memory for emulation.

Command Forms

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

Syntax

load [***filename***]

where

filename (optional string) If you wish to download a specific file immediately (in Dialog mode only).

Default

Load the current load set.

Description

The **load** command will download to PromICE the entire contents of the current load set, which is the set of files to load. The load set is built using the **file** and **image** commands. All specified files are processed and downloaded. The downloading process generates a file data map, which can be displayed with the **config** command.

When the **load** command appears in the loadice.ini file or as '-l' on the command line, it signals that LoadICE should download the files in the load set at the end of loadice.ini processing and before either exiting or entering the Dialog mode

Notes

Dialog mode is invoked only if loadice.ini contains the **dialog** command or '-d' is used on the command line.

Examples

load

Load the files now.

log

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

Command Forms

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

Syntax

log ***filename***

OR

log ***num***

OR

log

where

filename	(optional string) specify the log filename and begin logging.
num	(optional integer) logging control: 0 - turn logging off. 1 - turn logging on (will open a file if necessary).
no args	Toggle logging on/off (will open a file if necessary).

Default

No log is kept. The default log filename is loadice.log.

Description

The **log** command captures the traffic between LoadICE and PromICE in a file that can be sent to Grammar Engine for analysis. This is an alternative to changing the display level in LoadICE (refer to the **display** command). The **log** command writes the log data into a disk file. This saves the overhead of the **display** command and will not loose information or error conditions. The log can also be viewed using the DOS program vlog.exe, which is on the LoadICE distribution disk or can be downloaded from www.promice.com.

Notes

If the **log** command is used in Dialog mode without having previously specified a file name, then the file loadice.log. is created to store the log data. This is also the default filename for the vlog.exe viewer program.

Examples

```
log logfile.log
.
.
.
log 0
```

Log PromICE communications traffic to the file logfile.log. Then disable logging.

map

Control the display of the address range information during the loading process.

Command Forms

map loadice.ini file
map Dialog mode

Syntax

map ***num***

where

num (integer) map display control
0 - turn off display of map information
1 - turn on display of map information

Default

Off.

Description

LoadICE can display the range of addresses where data is being loaded. The **map** command controls this display.

Notes

When the data file has multiple data regions, there may be a large number of address ranges displayed. You can use this command to turn off such displays.

Examples

```
map 0
```

Don't display the map information.

modein

Specifies the initial mode for PromICE upon LoadICE invocation, either Load or Emulation mode.

Command Forms

modein	loadice.ini file
-mi	Command line

Syntax

modein [**stop** | **go**]

where

[**stop** | **go**] (keyword) PromICE mode upon LoadICE startup:

stop	places PromICE in Load mode at startup.
go	places PromICE in Emulation mode at startup.

Default

The default mode depends on the command line option and loadice.ini file processing. At connect time, LoadICE will place PromICE into Emulation mode, unless files are to be loaded. Loading files will leave PromICE in Load mode.

Description

The **modein** command defines the initial PromICE mode upon entering Dialog mode. This command takes effect when LoadICE initiates communication with PromICE, which usually occurs at the conclusion of loadice.ini and command line option processing.

Examples

```
modein stop
dialog
```

Place PromICE in Load mode at the beginning of the interactive Dialog session.

modeout

Specifies the final mode for PromICE upon LoadICE termination, either Load or Emulation mode.

Command Forms

modeout	loadice.ini file
-mo	Command line

Syntax

modeout [**stop** | **go**]

where

[**stop** | **go**] (keyword) PromICE mode upon LoadICE termination:

stop	places PromICE in Load mode at exit.
go	places PromICE in Emulation mode at exit.

Default

By default, LoadICE puts PromICE into Emulation mode upon exit.

Description

The **modeout** command defines the final PromICE mode upon program termination.

Examples

modeout stop

Place PromICE in Load mode when LoadICE exits.

modefixed

Maintains the current PromICE mode at LoadICE termination.

Command Forms

modefixed	loadice.ini file
-mf	Command line

Syntax

modefixed

Description

The **modefixed** command overrides the default LoadICE action of placing PromICE into Emulation mode on exit. The current PromICE mode in Dialog mode or command processing (command line option and loadice.ini) is not changed when LoadICE exits.

Notes

If files are loaded into PromICE during command processing (command line option and loadice.ini), PromICE will be in Load mode unless a **go** command follows the **load** command.

Examples

modefixed

Exiting LoadICE will not change the current PromICE mode.

move

Copy bytes within PromICE emulation memory.

Command Forms

move	loadice.ini file, Dialog mode
m	Dialog mode

Syntax

move ***start end destination***

where

start	(hex) The starting address of source data block where data is copied from. *start* is a PromICE ROM-relative address.
end	(hex) The ending address of source data block where data is copied to. *end* is a PromICE ROM-relative address.
destination	(hex) The starting address of destination location where source data block is to be copied to. *destination* is a PromICE ROM-relative address. **Note:** *destination* should not be within the *start* to *end* range.

Default

All three address parameters must be specified.

Description

The **move** command allows you to replicate data within PromICE memory. All addresses are PromICE ROM-relative addresses, which start at zero.

Notes

Overlapping memory regions, where *destination* is within the *start* to *end* range, are not recommended and are likely to fail.

Examples

m 100 120 300

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

noaddrerr

Ignore address-out-of-range errors during file loading.

Command Forms

noaddrerr	loadice.ini file, Dialog mode
-z	Command line
z	Dialog mode

Syntax

noaddrerr [***num***]

where

num (optional integer)
1 - View hex file records with addresses out of range.

Default

When this command is not used, the download process stops when an address error is encountered.

Description

During loading, if an address is outside the specified ROM emulation configuration (i.e. the address is larger than the ROM size), LoadICE will stop processing data and report an error message. The **noaddrerr** command will cause the offending data to be skipped and the load processing to continue.

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 ignore these records.

Examples

noaddrerr 1

Skip data that has addresses out of emulation memory and display them.

number

Specify the number of Daisy Chained PromICE units.

Command Forms

number	loadice.ini file
-n	Command line

Syntax

number ***total_units***

where

total_units	(integer) The total number of daisy chained PromICE units (0-255).

Default

The default module count is one.

Description

The **number** command is necessary whenever daisy chaining PromICE units.

This allows the LoadICE application to send out sufficient auto baud characters to ensure that all the PromICE units will complete the auto baud sequence. When the sequence concludes, the units will transmit some characters back to the host. LoadICE can then begin controlling the PromICEs.

On UNIX, if a host read were to be issued but no characters have been received, then the read will wait (indefinitely). Even though a serial line is opened with the option to not wait on the read (i.e. return error if there is 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.

Examples

number 3

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

output

Specify serial output port for connection with PromICE.

Command Forms

output	loadice.ini file
-o	Command line

Syntax

output *port_name* [***address***]

where

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

address (optional hex) On an IBM PC or compatible, 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.

Default

COM1	for PC
/dev/ttyb	for UNIX

Description

This command specifies the serial link for LoadICE communication with PromICE. Optionally, the port address may be specified for PC systems.

Notes

When using an AI communications session, this option specifies the link to use for host communication after the AI session is established.

The precedence of the **output** command and the **pponly** command, which specifies exclusive use of the parallel port, is as follows:

1. A command line switch overrides all other occurrences.
2. The last occurrence in the loadice.ini file overrides earlier occurrences in loadice.ini.

Examples

-o com3

Use COM3.

ppbus

Specifies that the parallel port be used only for downloads and that the serial port will be used for all other LoadICE/PromICE communications.

Command Forms

ppbus	loadice.ini file
-pb	Command line

Syntax

ppbus ***parallel_device_name*** [***port_address***]

where

parallel_device_name	(string) The LPT number or the device name (Solaris) where PromICE units are attached.
port_address	(optional hex) Non-standard port address.

Description

The **ppbus** command specifies that the parallel port be used only for downloads and that the serial port will be used for all other LoadICE/PromICE communications.

When daisy chaining PromICE units via parallel ports, this command is mandatory. The serial ports must be daisy chained because the control information is sent via the serial port.

Notes

This option works with the Ethernet print servers.

Examples

ppbus lpt1

Specifies the ppbus port as 'lpt1' in the loadice.ini file.

-pb /dev/bpp0

Specifies the ppbus port as '/dev/bpp0' from the LoadICE command line under Solaris.

ppmode

Sets the parallel port's communication mode.

Command Forms

ppmode	loadice.ini file
-P	Command line

Syntax

ppmode ***num***

where

num (integer) parallel port download mode:

- 0 Standard mode, around 30KBytes per second
- 1 Fast mode, about 60KBytes per second, with verification.
- 2 Turbo mode, about 90KBytes per second, no verification.

Default

0 - Standard mode for units with micro-code version 5 or older.
2 - Turbo mode for all units with micro-code version 6 or greater
When LoadICE connects to PromICE, it automatically determines the default parallel port mode based on the micro-code version.

Description

The **ppmode** command allows you to select the parallel port transfer mode. Turbo mode is only available on units that support fast mode, i.e. micro-code versions 6 and above.

Notes

Turbo mode is the default for units that support fast mode, i.e. micro-code versions 6 and above. If you have difficulty in connecting to PromICE and communicating reliably with turbo mode, you may want to try fast mode.

Turbo mode does not verify the downloaded data in order to achieve faster transfer rates. The **verify** command can enable data verification.

Examples

-P 1

Select fast transfer from LoadICE command line.

pponly

Specify PromICE communications to use only the parallel port in bi-directional mode. This feature is not available to Ethernet users and most UNIX users.

Command Forms

pponly	loadice.ini file
-q	Command line

Syntax

pponly ***parallel_device_name*** [***address***]

where

parallel_device_name	(string) The LPT number or the device name (Solaris) where PromICE units are attached.
address	(optional hex) Non-standard port address.

Description

The LoadICE will communicate with a single PromICE only using the parallel port.

Notes

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 to the installation section for more information about daisy chaining.

The precedence of the **pponly** command and the **output** command, which specifies the host COM port address, is as follows:

1. A command line switch overrides all other occurrences.
2. The last occurrence in the loadice.ini file overrides earlier occurrences in loadice.ini.

The **pponly** command cannot be used with most UNIX systems or Ethernet print servers. You must use the serial and optionally the parallel port. See the **ppbus** command.

Examples

pponly lpt2

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

-q /dev/bpp0

Specifies the **pponly** port as '/dev/bpp0' from the LoadICE command line under Solaris.

promiceid

Display the PromICE Identification number.

Command Forms

promiceid Dialog mode.

Syntax

promiceid

Default

Display the PromICE Identification number for the first PromICE unit.

Description

Each PromICE processor can have a 32 bit Identification number or serial number to uniquely identify the unit. This command displays the ID number.

The **promiceid** command allows PromICE to be bundled with other tools that will only work with PromICE specific units. If you would like specific ID numbers in units you purchase, contact Grammar Engine and request this service. Normally PromICE ID numbers are not unique.

Examples

promiceid

Report the PromICE Identification number on the first unit.

reset

Initiate a target reset. Also can specify the duration of target reset signal (RST).

Command Forms

reset	loadice.ini file, Dialog mode
-R	Command line
R	Dialog mode

Syntax

reset [***milliseconds***]

where

milliseconds	(optional integer) The length of the PromICE target reset signal in milliseconds (0-3000) for this and future reset events.

Default

The default reset time is 500 milliseconds.

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 emulation mode, then the reset is released (goes tri-state). However, you can cause a target reset using this command.

Notes

Specifying reset with a time of '0' will disable auto-reset. Then the target can be reset either manually or by issuing the **reset** command. Target reset is not automatically asserted while in Load mode.

Examples

r 500

Reset the target with a 500 millisecond pulse. Remembers pulse length for future resets.

resetfp

Controls whether LoadICE resets the FastPort before connecting with PromICE.

Command Forms

resetfp	loadice.ini file
-rfp	Command line

Syntax

resetfp [***num***]

where

num (optional integer) ethernet print server reset control

- 0 Disables the FastPort reset.
- 1 Enables the FastPort reset.

Default

The FastPort is not reset. When *num* is omitted, FastPort reset is enabled.

Description

The **resetfp** command resolves the condition when the FastPort-to-PromICE link is hung and LoadICE cannot connect via the FastPort.

LoadICE will connect with the FastPort's built-in monitor and reset it. After the FastPort reset has completed, LoadICE then connects to the PromICE unit(s).

Notes

Whenever a **reset** command is used, it will take about 20 seconds for LoadICE to connect with PromICE, instead of the 1 or 2 seconds required without a FastPort reset. To break an AI communications session, the FastPort must be reset every time LoadICE connects. This will be remedied in a future FastPort.

Examples

resetfp 1

Reset the FastPort from the loadice.ini file.

-rfp 0

Disable FastPort reset from the LoadICE command line.

restart

Restart the LoadICE to PromICE communication link.

Command Forms

restart Dialog mode

Syntax

restart

Description

The **restart** command will reestablish the communication link with PromICE without the need to restart the LoadICE application.

Notes

If you have reset or power cycled PromICE then you need to reestablish the link with this command. If LoadICE has timed out waiting for PromICE, then use this command. However, LoadICE will try to recover the link automatically unless the **noautorecovery** command was specified.

Examples

restart

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

rom

Specify ROM emulation memory size.

Command Forms

rom	loadice.ini file, Dialog mode
-r	Command line

Syntax

rom ***size***[**k**]

OR

rom ***size***[**m**]

OR

rom **gen_*part_number***

where

size (integer) The size of emulated memory, measured in the current word size (see the **word** command.). The specified number must be a power of 2.

k (optional literal) Use to indicate the *size* in thousands of the current word size.

m (optional literal) Use to indicate the *size* in millions of the current word size.

gen_part_number
(string of digits) JEDEC standard ROM part number for either 8 or 16 bit word size. The number must begin with a 27, such as 27512, 27010, 27040, 27080, or 27160.

Default

The maximum emulation size is the amount of emulation memory in the smallest PromICE unit in the current PromICE ROM configuration.

Description

The **rom** command specifies the size of ROM for PromICE to emulate. It must be less than or equal to the amount of emulation memory in the smallest PromICE unit in the current PromICE ROM configuration.

When emulating word sizes greater than 8 bits, state the target memory size as *nnn* K by the *iii* word size. For example, an 8 megabit device in 16 bit mode is stated 512K by 16 where *nnn* is 512 and *iii* is 16. So the commands for this memory are: word=16 and rom=512k.

Notes

Note that *gen_part_numbers* are scanned as decimal integers and all letters are ignored. The part AM27C020 is the same as 27020.

If your target is wired for a socket larger than the size of ROM you are emulating, then refer to the description of the **socket** command. Typically this can be the case for 1, 2 and 4 Mbit ROMs. A socket wired for 4Mb can be used with 1, 2 or 4 Mbit ROMs without any jumper changes.

Examples

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

These three examples all specify a 128K Byte (1Mbit ROM) to be emulated.

```
rom=256k
```

Lets you emulate a 256K Byte ROM (27020).

```
word 16 1 0
rom=256k
```

Lets you emulate a 256K by 16 ROM (28F400).

save

Save PromICE emulation memory contents to a binary file on the host.

Command Forms

save Dialog mode

Syntax

save ***file_name*** [***start end***]

where

file_name	(string) The name of the host file in which PromICE memory contents are to be saved. Path names may be included.
[*start end*]	(optional hex pair) Specifies an address range for the save (see below). Note: Both addresses are PromICE ROM-relative addresses.
start	(hex) Specifies the starting address of the range to save.
end	(hex) Specifies the ending address of the range to save.

Default

Save the entire current ROM configuration's memory contents.

Description

The **save** command reads the contents of PromICE emulation memory and saves it to a binary file. Optionally, it can save the contents in the PromICE ROM-relative address range *start* to *end*. The resulting binary file can be reloaded with the **image** command.

Examples

save newfile.bin 100 3fff

Save the data from the range 0x100 to 0x3FFF to a host file called newfile.bin.

search

Search PromICE emulation memory for an ASCII data pattern.

Command Forms

search	Dialog mode
S	Dialog mode

Syntax

search [***start*** ***end***] ***pattern***

where

[*start* *end*]	(optional hex pair) Specifies an address range (see below). Note: Both addresses are PromICE ROM-relative addresses.
start	(hex) *start* is the beginning address of the range to search.
end	(hex) *end* is the ending address of the range to search.
pattern	(quoted string) The text string to be located. Enclose the string in double quotation marks (" ").

Default

Search all current PromICE ROM configuration.

Description

The **search** command will search the current PromICE ROM configuration's memory for an ASCII string.

Notes

The *start* and *end* address range is optional. The entire specified range is searched for multiple occurrences of the string.

Examples

S 0 1000 "Enter new value:"

Search from 0x0 to 0x1000 for the string.

setup

Access a menu-driven tool that can be used to configure PromICE. You can generate a loadice.ini file to store your settings.

Command Forms

setup	Dialog mode
-setup	Command line

Syntax

setup

Description

The **setup** command invokes a menu-driven interface for LoadICE/PromICE configuration. Please follow the on-screen instructions. Be sure to save your settings to the loadice.ini file before exiting the menu interface.

Examples

setup

Invoke a menu-driven interface for configuring PromICE.

socket

When unused address lines cannot be pulled high, you may need the **socket** command to modify the PromICE address mask. Unused address lines are those that are available from the target ROM socket but are not needed to emulate the ROM.

Command Forms

socket loadice.ini file, Dialog mode

Syntax

socket ***size***[**k** | **m**]

OR

socket **gen*_part_number***

where

size (integer) The socket size, measured in the current word size (bytes or words) as defined by the most recent **word** command. The specified number must be a power of 2.

k (optional literal) Use to indicate the *size* in thousands of the current word size.

m (optional literal) Use to indicate the *size* in millions of the current word size.

gen_part_number (string of digits) JEDEC standard ROM part number for either 8 or 16 bit word size. The part number must begin with a 27 or a 29.

Default

The default socket size is the same as ROM size. As a result, all address lines beyond the emulation ROM address space must be high.

Description

Please see the *Unused Address Lines* chapter, in the back of this manual, which deals with the **socket** command in detail.

Examples

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

Lets you emulate a 128K Byte ROM in a socket wired for a 512K Byte ROM.

status

Displays the status of the target system as detected by PromICE.

Command Forms

status	loadice.ini file, Dialog mode
-st	Command line
st	Dialog mode

Syntax

status

Description

The **status** command is a quick way to find out if the target has power and if the target is accessing the ROM being emulated by PromICE.

Notes

The status is reported only from the master 8 bit unit of the first PromICE.

Sometimes you may need to issue the **status** command multiple times if the target has recently changed state such as after a reset. Check the status several times to determine if it is running. Use <CR> to repeat the previous command.

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.

stop

Cause PromICE to exit emulation mode and enter Load mode.

Command Forms

stop	loadice.ini file, Dialog mode
<ESC>	Dialog mode (toggles Emulation/Load modes)

Syntax

stop

Description

Force PromICE to stop emulating and enter Load mode. The target reset signal is asserted and held while in Load mode, unless the auto-reset feature has been de-activated by the **reset 0** command. To reactivate auto-reset, exit LoadICE and invoke it again.

Notes

The target will crash if it is executing out of the PromICE ROM emulation memory, unless the reset line is attached to the target.

When the **stop** command is issued, the buffers inside the PromICE unit(s) are shut off. If the target attempts to access code in PromICE memory, it will not receive correct data. If the reset line is connected, the reset will be held until a **go** command is issued.

Examples

stop

The load light should come on so you can execute commands that modify PromICE emulation memory contents.

test

Test PromICE ROM emulation memory on a particular PromICE 8 bit unit.

Command Forms

test	Dialog mode
t	Dialog mode

Syntax

test [***id*** | **c**] [***pass_count***]

where

[***id*** | **c**]	(optional integer or literal) Select the PromICE unit to test OR test the current configuration (see below).
id	(optional integer) A valid PromICE unit ID number (0-255). Do **not** follow *id* with a colon. Tests all of unit's memory with an iterative pattern.
c	(optional literal) Test the current PromICE emulation configuration with a non-repeating 32 bit pattern.
pass_count	(optional integer if first argument is present) The number of times to perform the memory test.

Default

Test PromICE 8 bit unit ID=0 memory once.

Description

The **test** command runs a simple RAM test on the memory of a PromICE 8 bit unit. If the test fails, then the addresses of the failures are reported.

Notes

This command destroys all data in PromICE emulation memory.

A failed test may indicate: 1) a damaged data buffer, 2) a problem with the non-volatile emulation memory controller (weak battery or failed Super Capacitor), or 3) a failure of a RAM chip inside PromICE.

Examples

t

Test memory once on PromICE unit ID=0.

version

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

Command Forms

version	Dialog mode
v	Dialog mode

Syntax

version

Default

Display the LoadICE version number and micro-code version of the first PromICE.

Description

The **version** command reports the version of software and micro-code you are running. This information is helpful when seeking technical support.

Notes

Please have this information readily available when contacting Technical Support.

Examples

version

Report version of the LoadICE and micro-code version numbers.

word

Specify the emulation data word width for the current PromICE ROM emulation configuration.

Command Forms

word	loadice.ini file, Dialog mode
-w	Command line
w	Dialog mode

Syntax

word ***word_width*** [***unit_ID*** { ***unit_ID_n*** }]

where

word_width (integer) defines the word width for the file. It must be an integral multiple of 8 and cannot be larger than the maximum data bus width determined by the total number of PromICE 8 bit units.

unit_ID { ***unit_ID_n*** }
(optional variable length integer list) The list of PromICE 8 bit unit IDs to define the byte order.

unit_ID (optional integer) The first PromICE 8 bit unit number corresponding to the byte at the lowest memory address within the word.

unit_ID_n (optional integers following *unit_ID*) The nth PromICE 8 bit unit number specifying the byte order within the word size defined by *word_width*.

Default

The default word size is 8 bits. Unless the byte order is specified, the default byte order is: all known PromICE 8 bit units arranged in ascending unit ID order.

Description

The **word** command is used to set the word size for the current PromICE ROM emulation configuration. The 8 bit unit IDs to be used, as well as their byte order within the word, can also be specified.

The word size specification and the ROM specification are used by most LoadICE commands that reference PromICE emulation memory.

Notes

The **word** command should precede the **checksum**, **dump**, **edit**, **fill**, **load**, **move**, and **rom** commands when used in the loadice.ini file. These commands depend on the data width of the target system.

When using a dual PromICE (models starting with "P2") to emulate a single 8 bit ROM, be sure to specify a single unit ID as the byte order. For example:

```
word 8 0
```

Otherwise, any load data overflow from the single 8 bit ROM's address space will not be reported.

The bus width is determined by the ROM bus width, not the processor's bus. If you are using a 32 bit processor with one 8 bit ROM, you should specify the word size as 8 bit for the **word** command.

Examples

```
w 16 0 1
```

This specifies a 16 bit word size to be used. The first byte of data (all the even numbered bytes) will be loaded into the bottom or master unit (unit ID0). The second data byte (odd numbered bytes) will be loaded into the top or slave unit (unit ID1). This is typical for Intel-style targets, based on how GEI target cables connect to dual PromICE units.

```
w 16 1 0
```

This specifies a 16 bit word size to be used. The first byte of data (all the even numbered bytes) will be loaded into the top or slave unit (unit ID1). The second data byte (odd numbered bytes) will be loaded into the bottom or master unit (unit ID0). This is typical for Motorola-style targets, based on how GEI target cables connect to dual PromICE units.

```
pponly=lpt1
rom=27010
file myfile.hex 8000=0
begin
word 8 1
fill AA
load
word 8 0
fill BB
load
```

Works with one 8 bit unit at a time. This example loads myfile.hex into each 8 bit unit but uses different fill patterns. It works with 8 bit unit number 1 followed by 8 bit unit number 0. Unit number 0 remains the current unit for subsequent LoadICE commands.

xmask

When unused address lines cannot be pulled high, you may need the **xmask** command to specify an arbitrary address mask. Unused address lines are those that are available from the target ROM socket but are not needed to emulate the ROM.

Command Forms

xmask loadice.ini file

Syntax

xmask mask_byte

where

mask_byte (hex) The address mask byte for target address lines A16 to A24.

Default

The default *mask_byte* is 0xFF.

Description

Please see the *Unused Address Lines* chapter, in the back of this manual, which deals with the **xmask** command in detail

Examples

xmask F3

Set address mask to 0xF3FFFF, which sets address lines A18 and A19 to zero.

5. The LoadICE Environment

Contents

OVERVIEW

The LoadICE environment includes the information on command syntax details, how loadice.ini is processed, and a basic explanation of PromICE ROM configuration models.

- LoadICE Syntax Details
  - LoadICE commands are case sensitive
  - Comments in loadice.ini
  - Numeric Arguments
  - ROM-Relative Address Arguments
- Introduction to how LoadICE processes the command line and loadice.ini file commands and the scope (or duration) of their actions.
- ROM Configuration Types

Be sure to visit **www.promice.com** for a free upgrade to the latest LoadICE software and documentation. If you don't have web access, contact your Sales Representative to obtain a copy.

Note: LoadICE version 4.0 or higher is required to use the commands documented in this version of the PromICE User Manual.

LoadICE Syntax Details

Case Sensitivity

LoadICE commands are case sensitive. All entries should be lower case unless otherwise specified.

Comments

Comments in loadice.ini begin with asterisk "*" and hide the remainder of the line.

Numeric Arguments

Numeric arguments of LoadICE commands each have an implicit radix defined by the particular command's syntax. LoadICE does not support explicit prefix or suffix letters to indicate the base of numeric arguments. For instance, all address arguments must be entered using hexadecimal digits.

Note that when LoadICE scans a numeric argument, all letters are ignored. For instance, generic ROM parts are scanned as decimal integers. The part AM27C020 is the same as 27020.

ROM-Relative Address Arguments

Most LoadICE commands that refer to PromICE emulation memory require PromICE ROM-relative address arguments. PromICE ROM-relative addresses are based at the beginning of ROM being emulated and are always hexadecimal.

For instance, the PromICE ROM-relative address 10 refers to the target address 80010H when the ROM starts at address 80000H.

Introduction to LoadICE Command Processing

LoadICE commands can be issued three ways: on the command line at LoadICE invocation time, in the loadice.ini initialization file during program startup, and in the interactive Dialog mode

Certain commands, when used on the command line or in the loadice.ini, will have lasting effects on the system. The duration of a command's effect is referred to as it's scope.

Commands in this section are listed by scope. Also loadice.ini processing is briefly explained.

INI Command scope: the entire LoadICE session

These commands issued on the command line or in the loadice.ini file will act upon the current PromICE ROM configuration emulation memory each time a load command is issued. To cancel the actions of these commands, end the LoadICE session by exiting from the program.

fill – write pattern into the ROM emulation memory (before load)

fillall – write pattern into the entire PromICE emulation memory (before load)

checksum – calculate checksum and write into memory (after load)

INI Command scope: the first load command

These commands issued on the command line or in the loadice.ini file will act upon the current PromICE ROM configuration emulation memory only for the first **load** command issued after LoadICE is executed. These actions expire automatically once completed.

edit – write bytes into emulation memory (after load)

move – move bytes to new location within emulation memory (after load)

Command scope: the current set of files to load

These commands, whenever issued, accumulate into a set of files to be loaded. When the **load** command is issued, all files belonging to the load set are sent to the PromICE according to the current PromICE ROM configuration in conjunction with their individual settings. To clear the current load set, issue the **clearfiles** command.

file – setup a hex record file for ROM emulation

image – setup an image record file for ROM emulation

INI File Processing

Normally LoadICE begins communicating with PromICE after loadice.ini file processing has finished. The commands in the loadice.ini file serve as configuration settings, not as a general-purpose scripting language. For instance, a **fill** command used in loadice.ini affects every load during the LoadICE session (see above).

You can use the **begin** command to cause LoadICE to immediately initiate communication with PromICE and treat the remaining commands in the loadice.ini file one at a time, much like Dialog mode commands. Commands following **begin** do not have INI Command Scope.

When you need to load more than one set of files inside a loadice.ini file, you can use the **begin** command followed by the **load** command to load the current set of files. Once loaded, the loadice.ini file processing resumes. Next use the **clearfiles** command to empty the current load set. Then you can issue **file** and/or **image** commands to specify new files and load them.

Example of INI File Loading

```
bank 2                          * define number of banks
bank 1                          * switch to second bank (#1)
file=myfile1.hex 400000=0
begin
load
clearfiles
bank 0                          * switch to first bank (#0)
file=myfile2.hex 401000=0
load
```

During the loadice.ini file processing, the **begin** command initiates connection to PromICE and loads *myfile1.hex* into logical bank 1. Then it loads *myfile2.hex* to into logical bank 0 at the conclusion of loadice.ini processing.

Configuration Dependencies

During loadice.ini file processing, the order of certain commands is significant. The placement of the **word** command and the order of the **pponly** and **output** commands will affect the success of your configuration.

The **word** command should precede the **checksum**, **dump**, **edit**, **fill**, **load**, **move** and **rom** commands when used in the loadice.ini file. These commands depend on the **word** command, which defines the data width of the target system.

The **output** command specifies the host COM port address. The **pponly** command specifies that LoadICE use the parallel port exclusively for all communications with PromICE. Both of these commands can be specified as switches on the command line that invokes LoadICE.

The precedence of the **pponly** and **output** commands are as follows:

1. A command line switch overrides all other occurrences.
2. The last occurrence in the loadice.ini file overrides earlier occurrences in loadice.ini.

All commands that configure Host-to-PromICE communications should precede the **begin** command, which initiates LoadICE communication with PromICE.

ROM Configuration Types

LoadICE supports two types ROM emulation configurations:

1) The current PromICE ROM emulation configuration

2) PromICE 8 bit unit configurations

The current PromICE ROM emulation configuration

The current PromICE ROM emulation configuration is the central, general purpose configuration that can be built with any number of PromICE 8 bit units. Most LoadICE commands refer to this configuration by default.

The **word**, **rom**, and **socket** commands define the basic configuration architecture. The emulation word size can be any multiple of 8 up to 64 bits, such as 8, 16, 32, or 64 bits, limited only the number of PromICE 8 bit units in the system. The emulation ROM size is limited to the smallest memory capacity of all the 8 bit units included in the word's byte order definition.

PromICE 8 bit unit configuration

See the chapter entitled *PromICE Unit IDs.*

Application Note: Emulating multiple 8 bit ROMs

When using dual PromICEs (Models starting with "P2"), LoadICE, by default, allocates the second 8 bit unit of the dual PromICE into an alternate bank with the same addresses but presumably a separate target chip select signal. PromICE ROM-relative addressing places the second 8 bit unit addresses above the first unit's address range. For example:

```
rom=27010
word 8
```

Is the same as:

```
rom=27010
word 8 0 1
```

Unit 0 has PromICE ROM-relative addresses 0 to 1FFFF and Unit 1 has 20000 to 3FFFF.

To work exclusively with one 8 bit unit, issue the word command with a word size of 8 followed by the unit ID. This example loads myfile.hex into each 8 bit unit but uses different fill patterns. It works with 8 bit unit number 1 followed by 8 bit unit number 0.

```
pponly=lpt1
rom=27010
file myfile.hex 8000=0
begin
word 8 1
fill AA
load
word 8 0
fill BB
load
```

6. Troubleshooting

CONTENTS

Introduction

Often additional information is helpful in resolving PromICE configuration problems. The interactive Dialog mode has commands that report useful data. For more information on using Dialog mode, see the introduction in the *LoadICE Environment* chapter.

Once in Dialog mode, the following commands are particularly useful. Use the **go** command to access Emulate mode before using these commands.

status Reports if target is powered and if the target is accessing PromICE memory. Also, it will report trace status information if the PromICE has the Trace Option.

config Reports the current PromICE memory configuration and load file information.

analyze Reports information about a hex or image file.

log Report on LoadICE / PromICE communications that should be sent to Grammar Engine for analysis.

Once in Dialog mode, the following commands can only be used in Load mode. Use the **stop** command to access Load mode.

test Test PromICE emulation memory (overwrites contents).

compare Compares contents of PromICE memory with the original files.

dump Displays the contents of PromICE emulation memory.

fillall Fills PromICE emulation memory with a fixed pattern. Use to prevent execution of old code fragments.

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 emulates to a point in code and hangs up repeatedly, 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.

HOST TO PROMICE

Windows / 95 / NT

Make sure you are using the latest version of LoadICE for your version of Windows. If you are using the NT parallel port driver, make sure you logged in as administrator to install the driver and that the PromICE.ini and BIOS settings are correct.

LoadICE Version

Check your LoadICE version. Upgrades can be downloaded from www.promice.com.

Switch box / extension cable / software key

Connect PromICE to the host using only the supplied cables. Disconnect any switch boxes, extension cables and/or software keys (hardware security keys or dongles). These are not recommended for use with PromICE. Software keys may intercept codes intended for PromICE and stop communication. Once the problem has been solved, you can experiment with reconnecting these devices.

Port specification

Make sure that you are connected to the correct port.

If the port has a non-standard address, specify it using a command line switch or in the loadice.ini file as follows:

```
output=com1:3f8
```

OR

```
pponly=lpt1:378
```

Baud rate

Make sure you have specified a valid baud rate in the **baud** command. PromICE supports 1200, 2400, 4800, 9600, 19200 and 57600 baud. If a slower baud rate works, then noise may be the problem.

Serial/parallel daisy chain

Make sure the daisy chain modules are connected properly. Refer to the "*Hardware Installation*" Section of this manual for more information.

Noise

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

Damaged serial/parallel port

If communication is impossible through the serial port, try using another port, preferably on another host system. (The TTY interface feature has been retired.)

PromICE TO TARGET

This section covers emulation problems including configuration, connectivity, and run-time issues. The target either fails to emulate, or stops emulating. Problems experienced with emulation may show up in a number of ways.

Grammar Engine Inc. Technical Support is available to help with emulation problem resolution.

Load File Mapping

If the file specification in the loadice.ini or command line is incorrect, the target code will not run or will run to a point and crash.

If your load file does not load any code or data into target RAM, make sure NOT to use the **noaddrerr** command in your loadice.ini or on the command line. This way, if data loads outside the PromICE 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 your file's starting address and where you want it to load into ROM.

Example: Assume you have a file called tst.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=tst.hex 400000=0 16 1 0
```

The "16 1 0" suffix is optional when the **word** command has already specified the byte order as in "word=16 1 0".

Example: Assume you have a file called tst.bin that has a 14 byte load header. Assume that the file contains 16 bit data and the low order byte is emulated by unit ID0 (master) and the high order byte by unit ID1 (slave). The specification would be:

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

Again, the "16 0 1" suffix is optional when the **word** command has already set the byte order using "word=16 0 1". When in doubt, specify all command parameters.

Load Light Stays On

When the **Load** light on the PromICE front panel remains On, the main cause is that PromICE is not sensing target power on the target ROM socket or probe. PromICE will not emulate if it cannot sense target power.

Check your target cables and jumper settings. Inspect the target ROM socket and the probe for damage. Most sockets are rated for a limited number of insertions. Minimize the number of insertions when using the older ET probes included with the PL32C target cables.

When using the 3 Volt Adapter, be sure that you have jumpers on **EXT**, **ROM** and **32** on the back of PromICE. Failing to use both the **EXT** and **ROM** jumpers will cause the Load light to remain on. Note that when using the 3 Volt Adapter properly configured with the jumpers listed above, PromICE monitors its own power source.

Emulation Size

In resolving configuration problems, begin by emulating the largest size that your PromICE can emulate and work down to the size the target's ROM socket is wired for, if smaller. If the ROM you are emulating is a 27512, but your target's socket can handle a 27020, try to emulate the 27020.

If you are not sure whether your target can use larger ROM sizes, then 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.

Noise

Check the cabling between PromICE and the target. Make sure the cable is not near any power supplies, monitors or cables. If your target is sensitive to noise or is itself noisy, it may not emulate reliably. Keep the cables as short as possible and route them as far away as possible from noise sources. Grammar Engine ships shielded cables.

Parasitic Power

PromICE should be powered using the external power supply shipped with it. Your target should not parasitically power the PromICE unit.

Incorrect Byte Order

If you are emulating more than one ROM and/or are using a target with a 16 bit or larger word size, define the byte order using the **word** command in the loadice.ini file or on your command line. If the byte order is not correct, the code will not load into memory correctly. When the target reads the disorganized code, it will fail.

Stopped Working After A Move

Whenever handling PromICE or your target be sure to use proper anti-static protective procedures. See the *Installation* chapter for details. If PromICE has taken static damage or been exposed to high voltages, the buffers within the PromICE are the first to be damaged, by design. Call for a Repair RMA number.

Be advised that replacing the buffers yourself will void your warranty and is done entirely at your own risk.

Check your target cables. Inspect the target ROM socket and the probe for damage. Always disconnect any adapter boards from the ROM probe before pulling it from the target socket. Be sure to pull the probe straight up. Failing to do so can damage the probe and/or the socket.

Most ROM sockets are rated for a limited number of insertions. Especially minimize the number of insertions when using the older ET probes included with the PL32C target cables. A probe for each target board is a good investment.

EMERGENCY REPAIRS

WARNING:

OPENING THE PROMICE CASE WILL VOID ALL WARRANTIES.

When you are on a tight schedule, you may choose to repair the PromICE yourself. For such cases, these instructions are provided.

Warning: This will definitely void any warranty. You assume the full risk of applying any repairs described 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 can reduce the chances of damaging your unit. Grammar Engine will not be liable for any damages incurred, even 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.

Replacing ROM Interface Buffers

If the PromICE will not emulate or LoadICE is reporting Memory Size Zero errors when you connect with PromICE, then you may have damaged buffers on the PromICE ROM interface. One or more damaged buffers can block address or data lines and cause the previously mentioned problems.

To replace the buffers:

1. To open the PromICE unit, use a #1 Phillips screw driver and remove the two screws from the underside of the PromICE box. Remove the flat head screw on the back panel that is attached to a heat-sink.
2. Slide out the circuit boards and the panel. Keep them all together as you take them out or they may bind in the case. If you have a duplex unit and/or AI option installed, then remove the panels and gently pry apart the circuit boards.
3. The address input buffers, marked U11, U12 and U13, are located behind the ROM cable header. These are 74ALS244 chips, which are 20 pin socketed devices. They should be replaced with identical chips. If you lack enough chips then replace them one at a time, until the defective chip is replaced.

4. The data buffer, marked U14, is a 74ALS245. Replace it with an identical 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 onto the master board).
6. After the chips are replaced, reassemble the boards by carefully lining up the 44 pin headers on both sides of the board. You may test the unit now to see if the problem is fixed.
7. To completely reassemble the unit, 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 on the top half of the box and close it with the Phillips on the underside of the box.

Replacing the Parallel Port Buffers

If you are having problems with the parallel port, and you have tried the regular troubleshooting section, then the parallel port buffers may be damaged.

While the unit is open, check the two buffers right behind the parallel port connector. These are 74ALS244 and 74LS244 chips, which are also 20-pin socketed devices. Replace them with identical chips.

PRIORITY FAX

To: **GRAMMAR ENGINE TECHNICAL SUPPORT**
Fax: (614)899-7888

From: _____
Company: _____
Voice: ( ) _____
Fax: ( ) _____
Model #: **P**_____ (Please include complete number)
Serial #: _____ (Below model number)
(The *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:

_____
_____
_____
_____

RMA INFORMATION

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

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

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:
_____
_____
_____
_____

7. Error Messages

Most messages from LoadICE inform you of PromICE's status and display information you have requested. The following are the messages displayed when errors occur. Unless you are in Dialog mode, errors will terminate LoadICE. Dialog mode will prompt for more command input.

Error messages are identified by their header. Additional data is often displayed to provide more information on particular errors. Sometimes a system error message is also displayed (if the global variable 'errno' has been set). The system error message usually indicates a failed system call and may be cryptic.

LoadICE will also display the user input that was being processed when the error occurred. In some cases, the error condition may not correspond to the current command input. Negative error numbers are failure codes returned from the PromICE emulation unit(s). Positive error numbers are from LoadICE itself.

Errors reported by LoadICE at a glance:

(-6) Interface is not available / active
(-5) Interface is busy
(-4) Timer expired while waiting
(-3) DataOverflow - lost host data
(-2) No data available from the target
(-1) Feature not implemented
(1) LoadICE parser internal error #1
(2) Illegal command
(3) LoadICE parser internal error #2
(4) Too many arguments supplied
(5) Expected argument not supplied
(6) Filename error
(7) Invalid baud-rate
(8) Invalid ROM size
(9) Invalid word size
(10) Invalid ID list
(11) Open failed
(12) Unable to skip file data
(13) Device I/O error
(14) Bad port name
(15) End-o-File
(16) Bad parameters to picmd()
(17) Communication error
(18) Name too long
(19) Timed out waiting for response
(20) Feature Not Implemented Yet!
(21) Verify failed
(22) Invalid unit ID
(23) Address out of range
(24) LoadICE internal error #3
(25) Bad arguments
(26) Bad checksum in record
(27) Feature not supported on this unit
(28) Bad argument for driver call
(29) Bad data in the hex record
(30) Unit is LOCKED
(31) Not enough units to emulate word-size
(32) Memory Size Zero
(33) Operation terminated by user
(34) Data over-run
(35) Key NOT assigned
(36) Can't do while Emulating!
(37) Not emulating! - do a 'go' first
(38) No operation!
(39) No link with units...
(40) Must have AI Rev3.1...
(41) Must include 'word=' statement...

(-6) Interface is not available or not active

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

(-5) Interface is busy

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

(-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.

(-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.

(-2) No data available from the target

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

(-1) Feature not implemented

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

(1) LoadICE parser internal error #1

Cause:
This error should only occur if LoadICE has been modified.
Solution:
Correct the loadice.ini file and retry the command.

(2) Illegal command

Cause:
Spelling or capitalization error.
Solution:
Check spelling and retry. All commands are case sensitive.

(3) LoadICE parser internal error #2

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

(4) Too many arguments supplied

Cause:
Too many arguments were specified on the command.
Solution:
Check command syntax and retry the command.

(5) Expected argument not supplied

Cause:
Required input was omitted.
Solution:
Check the command syntax.

(6) Filename error

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

(7) Invalid baud-rate

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

(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). See the **rom** command.

(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...). See the **word** command.

(10) Invalid ID list

Cause:
Word size is too large.
Solution:
Set word size to a smaller value. See the *Software Configuration* chapter for more information.

(11) Open failed

Cause:
File name not specified correctly or is not in the current path.
Solution:
Make sure the file and it's extension are correctly spelled and in lower case. The file should either be in the same directory as the loadice.ini or the path should be specified.

(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.

(13) Device I/O error

Cause:
Error internal to the host operating system.
Solution:
Check your host system for problems. Check the host system message or error log for a more detailed description.

(14) Bad port name

Cause:
A serial or parallel port does not exist or has a bad port address.
Solution:
Check the port name and address. See the *Software Configuration* chapter for more information.

When using the **ailoc** command, this error will occur if an **output** command is not specified. The AI option initialization uses the serial port defined by the **output** command.

(15) End-o-File

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

(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 or download a new copy from www.promice.com.

(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 vice versa. Check the cable connections at both the host and PromICE sides.

(18) Name too long

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

(19) Timed out waiting for response

Cause:
Noisy connection or incorrect version of LoadICE.
Solution:
Move cables away from monitors and/or power supplies. If you are using Windows 95 or NT, download the latest version of LoadICE from www.promice.com.

(20) FEATURE NOT IMPLEMENTED YET!

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

(21) Verify failed

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

(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.

(23) Address out of range

Cause:
Incorrect file offset or ROM size set too small. Some of the code may be intended for RAM.
Solution:
Correct the file offset. See the **file** command in the *Command Reference* chapter for more information on file offsets. Set the ROM size to emulate a larger part. To ignore data addresses outside of the PromICE emulation space, add the **noaddrerr** command to the loadice.ini file.

(24) LoadICE internal error #3

Cause:
Internal error.
Solution:
Recopy LoadICE from the distribution disks or download it from www.promice.com.

(25) Bad arguments

Cause:
A command has invalid input.
Solution:
This usually occurs when the **ailoc** and **pponly** commands are included in the loadice.ini file. Add "*output=<com>*" (where <com> is the serial port to which PromICE is connected) as the first line in the loadice.ini file. See the *Command Reference* chapter to verify syntax for a particular command.

(26) Bad checksum in record

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

(27) Feature not supported on this unit

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

(28) Bad argument for driver call

Cause:
Corrupt LoadICE application.
Solution:
Recopy LoadICE from the distribution disks or download it from www.promice.com.

(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.

(30) Unit is LOCKED

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

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

Cause:
Emulating a word size larger than the current number of PromICE units.
Solution:
Select a smaller word size. 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 units and "P1" units count as one.

(32) Memory Size Zero

Cause:
The PromICE memory controller hasn't charged yet.
Solution:
PromICE may contain 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. Replace the power supply, if necessary.

If you are using the **ROM** jumper and are not using a 3 Volt Adapter, remove the **ROM** jumper and use an **EXT** jumper in order to connect the external power supply.

If jumpers were on both **ROM** and **EXT** without a 3 Volt Adapter attached, then be advised that PromICE power was delivered to your target.

(33) Operation terminated by user

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

(34) Data over-run

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

(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.

(36) Can't do while Emulating!

Cause:
A command was issued that requires the PromICE to be in Load mode.
Solution:
Use the **stop** command or press 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 will need 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.

(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 press the escape key (escape not supported under UNIX). If the reset line is not connected from PromICE to the target, the target will have to be reset after the "*go*" is issued.

(38) No operation!

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

(39) No link with units...

Cause:
Attempted to issue a command while the Host/PromICE link was down due to timeout, etc.
Solution:
Issue a **restart** command. Alternately, exit LoadICE and run it again.

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

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

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

Cause:
You are using a command that requires the **word** command to already be defined.
Solution:
Place the **word** command into your loadice.ini file. Refer to the *Software Configuration* chapter for more information.

8. AI Configuration

Download Options

LoadICE Example: AI Setup

Introduction

The PromICE Analysis Interface option, referred to as "AI," implements a virtual serial channel, which is accessible through the target's ROM socket. This virtual channel operates like a ROM-based UART. The host connects to the PromICE serial port in place of a target serial port. PromICE functions as a pass-through data-forwarding device.

The PromICE AI option is available on units with "AI" or "AI2" as part of the model number. If you want to be able to debug via the target ROM socket and your PromICE lacks this option, contact your Grammar Engine Sales Representative for an upgrade.

On a dual PromICE, AI is only accessible through the master PromICE 8 bit unit, ID=0. The master unit connects to your target through the lower connector on the back of a dual PromICE. Internally, the AI option is a PromICE daughter board.

How AI Works

The target system can communicate with the host computer using the AI option like a serial port. The AI control registers are mapped into a region within the PromICE ROM emulation memory. The target can accomplish bi-directional communication with the host entirely by performing reads of the AI interface. AI does not require the ability to write to ROM addresses to communicate.

With minimal code on the target system, you can communicate bi-directionally with no cost to target hardware resources. The AI option can provide interrupt driven communications. If your target allows write access to ROM addresses, your debugger monitor can write into ROM emulation memory.

How Debuggers Work With the AI

An embedded debugger has two components, the host-side front-end and the target-based monitor. The front-end runs on your host, manages the target, and provides debugging with the help of its target monitor program. This monitor communicates with the front-end via a connection between the target and the host system.

Conventionally, the link is a serial channel connected to COM ports on the host and the target. Adding a PromICE with the AI option is like adding a spare serial port to your target. PromICE AI operates similar to a UART on your target system.

The only required modification to your target monitor is to install support for the AI virtual UART. Many third party debugging packages support PromICE/AI as a communication option.

LoadICE Software for AI

Be sure to visit **www.promice.com** for a free upgrade to the latest LoadICE software and documentation. If you don't have web access, contact your Sales Representative to obtain a copy.

Note: LoadICE version 4.0 or higher is required to use the commands documented in this version of the PromICE User Manual.

AI Versions

The PromICE AI option has two versions: AI1 and AI2. To determine which version is present in a PromICE, use the "**config rom**" command to display as a status similar to the following:

```
EMULATION UNITS PRESENT:
 PromICE ID0 Memory=524288 Emulating=524288 FillChar=0xFF Master/AI2
```

A PromICE 8 bit unit with the AI Option will have either "AI1" or "AI2" as the last item on its line. Be sure to note which version you have before reading the following section.

AI Configuration Reference: Overview

1. Configure PromICE so that your target emulates
2. Configure the debugger monitor with a target serial port, if present
3. Reconfigure the debugger monitor to work with AI.
4. Consider these debugger monitor configuration issues
5. AI Register Offsets for word sizes 16 bits and greater
6. AI Register Offsets for targets with burst mode ROM accesses OR long processor read cycles
7. Combine the AI Register layout analysis done in the preceding two steps
8. On AI 1 units: setup the address mask, if needed
9. Miscellaneous Configuration Issues
10. Configure the debugger monitor to work with AI in polled mode
11. Optionally, configure the debugger monitor to work with AI interrupts
12. Optionally, connect the write line for debugger monitor use

AI Configuration Reference

Follow this procedure to setup your debugging configuration:

1. **Configure PromICE so that your target emulates**

   The first step is to configure PromICE ROM emulation to work with your target. Once PromICE emulation is working, then work with the debugger. Follow the instructions in the beginning of the manual. If you have some known working code, use it to test your emulation setup. Once the target is emulating with PromICE connected, be sure to store that configuration in your loadice.ini file.

   At this point, the only changes you will need to make to your loadice.ini file will be to change the files to be loaded and to add AI commands. You should avoid changing your **rom**, **word**, or **socket** commands.

2. **Configure the debugger monitor with a target serial port, if present**

   If your target has a serial port, temporarily configure the debugger's monitor to work with that target serial port. Verify that the debugger works.

3. **Reconfigure the debugger monitor to work with AI.**

   Most debuggers that support AI will offer convenient reconfiguration.

4. **Consider these debugger monitor configuration issues**

   While configuring your debugger's monitor, consider the following items:

   Step 5: On target systems with word sizes of 16 bits or greater, determine the offsets for the AI control registers, as seen by the target.

   Step 6: On target systems which use burst mode accesses to ROM addresses (or when the target processor read cycle is greater than the ROM width), determine which burst mode is in use and adjust the offsets for the AI control registers.

   Step 7: Combine the analysis in the preceding two steps, both of which can affect the code used in the debugger monitor, to locate and read the AI control registers

5. **AI Register Offsets for word sizes 16 bits and greater**

   The AI control registers, as seen by 8 bit targets, occupy four consecutive byte addresses within the memory of PromICE 8 bit unit ID=0.

   On systems with larger word sizes, the AI control registers occupy byte positions within four consecutive words. The location of the control registers within the words depends on the position of the PromICE unit ID=0 in the current ROM configuration's word layout.

Most debugger code for working with the AI control registers assumes that the registers occupy the low order byte within the word. On Intel-style systems (little endian), the **word** command commonly looks like:

```
word 16 0 1        *where unit 0 has D0 to D7
```

On Motorola-style systems (big endian), the **word** command commonly looks like:

```
word 16 1 0        *where unit 0 has D8 to D15
```

If the **word** command cannot be made to resemble the above, then the code in the debugger monitor will likely have to change. The monitor must access PromICE 8 bit unit ID=0 where it appears in the target's data word.

On various targets, the AI control register offsets can be as follows:

Word	Offsets
8	0, 1, 2, 3
16	0, 2, 4, 6 or 1, 3, 5, 7

Word	Offsets
32	0, 4, 8, 12 or 1, 5, 9, 13; etc.
64	0, 8, 16, 24 or 1, 9, 17, 25; etc.

6. AI Register Offsets for targets with burst mode ROM accesses OR long processor read cycles

On targets with burst mode ROM access, add the AI **burst** command to your loadice.ini file. The AI **burst** command is also needed if the target processor read cycle is greater than the ROM width, such as a long word processor booting from an 8 bit ROM.

Also, adjust the locations of the AI control registers to compensate for the multiple reads generated by each burst read. AI1 PromICE units support burst modes of 0, 4, 8, and 16 bytes. AI2 PromICE units support burst modes of 0, 2, 4, 8, 16, and 32 bytes. For each power of two increase in burst length, PromICE ignores a low order address bit as follows:

Burst	Ignores	Offsets
0	none	0, 1, 2, 3
2	A0	0, 2, 4, 6
4	A1 – A0	0, 4, 8, 12

Burst	Ignores	Offsets
8	A2 – A0	0, 8, 16, 24
16	A3 – A0	0, 16, 32, 48
32	A4 – A0	0, 32, 64, 96

The target monitor code must account for the control register offsets due to the burst mode accesses. The effect is to insert additional filler bytes between the control registers. This is in addition to the register spacing due to target word size.

7. Combine the AI Register layout analysis done in the preceding two steps.

The previous two steps can both affect the memory layout of the AI control registers as seen by the target. Now combine the offset information from the two previous steps into a single register layout. Modify the debugger monitor to use these offsets to locate and read the AI control registers.

For example, on a system with 2KB ROM (2716) on a 16 bit word and 4 byte burst mode, the AI register layout is as follows:

A 11	A 10	A 9	A 8	A 7	A 6	A 5	A 4	A 3	A 2	A 1	A 0	Address Lines
..1	X	X	X	X	X	X	X	X	X	X	X	rom 2kb
..1											X	word 16
..1									X	X		burst 4
..1	0	0	0	1	1	0	X	X	X	X	X	ailoc C0
..1	0	0	0	1	1	0	0	0	X	X	X	ZERO
..1	0	0	0	1	1	0	0	1	X	X	X	ONE
..1	0	0	0	1	1	0	1	0	X	X	X	HOST_DATA
..1	0	0	0	1	1	0	1	1	X	X	X	STATUS

As shown in the example above, the AI registers can be accessed within the following word address ranges on an Intel-style target:

Register	Word Address Range	word 16 0 1	word 16 1 0
ZERO	0xF8C0 to 0xF8C6	low byte	high byte
ONE	0xF8C8 to 0xF8CE	low byte	high byte
HOST_DATA	0xF8D0 to 0xF8D6	low byte	high byte
STATUS	0xF8D8 to 0xF8DE	low byte	high byte

During AI setup, it may be helpful to write a test program that can discover the AI locations. This is done by filling the region surrounding the AI control registers with 0xCC and scanning that region for the registers. Determine the size of the entire register region and divide by four to obtain the register spacing.

8. On AI 1 units: setup the address mask, if needed

AI2 users can skip this item because AI2 register address decoding ignores all address lines beyond the ROM emulation size.

On AI1 units, additional LoadICE commands may be necessary if it is not possible to pull high all unused address lines available from the target ROM socket. See the *Unused Address Lines* chapter for details.

9. Miscellaneous Configuration Issues

Cache

The AI option will not function if its memory range is cached. Either disable the cache during debugging or find a way of forcing a cache miss for every AI access.

Adjusting AI Timing

Depending on the memory access timing of the target processor, the **aicontrol** command may be needed. See the **aicontrol** command for details. The 683xx and DSP processors use this command most often. If the target works reliably, do not use this command.

Watchdog Timers

If you have a watchdog timer on your target that is not used by the debugger, then disable it in the monitor startup code. The debug monitor will not come up if a watchdog timer is enabled but not configured by the monitor.

Using the pponly command with Serial AI

LoadICE is used to create a serial AI communications session. When using the LoadICE **pponly** command to control PromICE only through the parallel port, you must still add the **output** command to specify the host serial port in the loadice.ini file:

```
output=com2
pponly=lpt1
..
..
ailoc 200, 19200
```

10. Configure the debugger monitor to work with AI in polled mode

Add the **ailoc** command to your loadice.ini file. Add other AI commands as needed.

Begin with polled mode communication. By using polled communications, you can verify that your target monitor configuration is working. Once properly configured, you will be able to debug through the ROM socket.

11. Optionally, configure the debugger monitor to work with AI interrupts

Interrupt driven communications is primarily used as a way for a debugger to establish efficient character-level communication with the target. When not used at the character level, interrupts can be used to regain control of a free-running application.

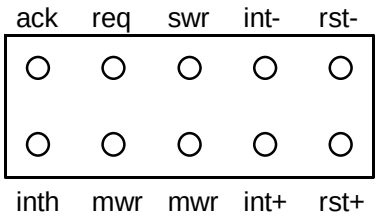
Interrupt setup includes connecting a line from the interrupt output pin, **int-** (low asserted) or **int+** (high asserted), to an interrupt input on your target. Usually, the **int** pin is connected to NMI on the target system to allow for debugging. Most debugger monitors require you to supply your own routines to handle interrupt driven communications.

Use the **aircvint** command to activate AI target receiver interrupts, which will cause an interrupt for every character received from the host. Only on AI2, you can use the **aixmtint** command to activate AI target transmitter ready interrupts, which interrupts each time the data transmitter becomes idle.

Interrupts do not occur until the target AI initialization is complete (see *AI Porting*, the **AIinit** routine in the code example). If target communication fails when you switch to interrupt driven communications, first check the interrupt handler for problems.

Connecting the Interrupt Line

The interrupt signals are among the auxiliary signals on the PromICE back panel.



int- and int+: (outputs) These interrupt signals are driven by PromICE whenever the HDA bit is set in the AI status register. This occurs during AI communications, or when caused by a host command. Both polarities of the signal are provided. They are driven by a 74LS125 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.

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.

Debuggers can request PromICE to interrupt the target system (using the DTR or INIT line). The debugger can use this feature to regain control of the target when your program is running. When connecting the interrupt signal, note the interrupt line on your target. Some debuggers will want to know which target interrupt line receives PromICE interrupt signals.

12. Optionally, connect the write line for debugger monitor use

Note: Most debuggers with AI support require the write line to be connected.

If your target allows write access to ROM addresses, you can enable write cycles to ROM emulation memory. This will allow your debugger to download code directly into PromICE emulation memory from the target side, set breakpoints in ROM addresses, and single step.

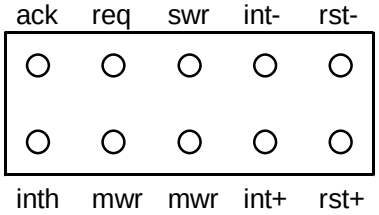
Newer PromICE units have three write pins: two **mwr** (master with D0 - D7) pins and one **swr** (slave with D8 - D15) pin. When used with one write line, use a jumper to connect one of the **mwr** pins to the **swr** pin and connect the line from the target to the remaining **mwr** pin. Note that on single PromICE units with a master but no slave board, the **swr** pin is not connected.

Older PromICE units have a single write called **wrt** pin (both master and slave), which does not support multiple write lines for byte-writes on 16 or 32 bit configurations. Older dual PromICEs can be upgraded to byte-write capability.

Connecting the Write Line

The following explains the details of connecting write lines to the newer PromICEs, which have the back panel connector as shown below.

Connect the write line inputs, **mwr/swr,** on the back of PromICE to the write line on your target.



mwr / swr: (input) These are low asserted inputs that are connected to the system write line on the target. The target can do write cycles to PromICE ROM emulation memory. The **mwr** pin is used to write to the master unit

(bottom back connector). The **swr** pin is used to write to the slave unit (upper back connector) of a dual PromICE (model prefix P2*xxx*).

Connecting the Write Line for a singe ROM

The following describes how to connect the target's write line into the PromICE for a single ROM:

8 bit ROM

Leave the jumper on the **swr** and **mwr** pins, and connect the target's write line to the spare **mwr** pin. Connect the target cable into the master unit (the lower connector on the back of a dual PromICE).

16 bit ROM using Dual PromICE

Leave the jumper on the **swr** and **mwr** pins, and connect the target's write line to the spare **mwr** pin. Connect the two cables from the 16 bit adapter board to the connectors on the back of the dual PromICE.

Variations on Connecting the Write Line for Multiple ROMs

Target systems implement write access to multiple ROMs in two basic ways:

- Each ROM chip has its own chip_select decoding and is connected to a system-wide write line.
- Multiple ROMs share a common chip_select and the system write line is decoded into chip-specific write lines.

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

8 bit word (two ROMs, system-wide write line)

Leave the jumper on the **swr** and **mwr** pins, and connect the target's write line to the spare **mwr** pin. Connect the target cables into a dual PromICE or daisy chained PromICEs.

16 bit word (two 8 bit ROMs, system-wide write line)

If your target has a system-wide write line, jumper the **swr** and **mwr** pins together, and connect the target's write line to the spare **mwr** pin. Connect the target cables into a dual PromICE or daisy chained PromICEs.

If your target supplies separate chip selects for each ROM, byte writes are possible. Otherwise all writes will be 16 bit word writes.

16 bit word (two 8 bit ROMs, decoded write lines)

If your target has one chip select and decodes the write line to determine which ROM to write, remove the jumper between **mwr** and **swr**. Then, connect the write line for ROM that is cabled to the master unit (bottom connector on the back of PromICE) to one of the **mwr** pins on the

PromICE. Connect the other write line to the slave unit input, which is the **swr** pin on the PromICE. It is important that the correct ROM write line be connected to the corresponding write pin.

32 bit word (four 8 bit ROMs, system-wide write line)

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

If your target decodes separate chip selects for each ROM, byte writes are possible. Otherwise all writes will be 32 bit word writes.

32 bit word (two 16 bit ROMs, system-wide write line)

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

If your target decodes separate chip selects for each ROM, 16 bit word writes are possible. Otherwise all writes will be 32 bit word writes.

32 bit word (four 8 bit ROMs, four decoded write lines)

This configuration requires two dual PromICEs (models starting with "P2"). Remove the jumpers on **mwr** and **swr** on each of the PromICE units. Connect one of the target's four write lines to each of the pins on the PromICE units. Be careful to connect the write signal for each ROM to the corresponding PromICE master or slave unit, in accordance with the byte order specified in the **word** command.

32 bit word (two 16 bit ROMs, two decoded write lines)

This configuration requires two dual PromICEs (models starting with "P2"). Leave the jumpers on **mwr** and **swr** on each of the PromICE units. Connect one of the target's two decoded write lines to each PromICE unit. Be careful to connect the write signal for each ROM to the corresponding PromICE, in accordance with the byte order specified in the **word** command.

Note: Some target debuggers support writing only words. They do not depend on the availability of byte writes.

Conclusion

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 minor part of the overall monitor configuration. The next section will describe the basics of monitor based debugging - how it works and how PromICE AI fits into the solution.

Download Options

There are two ways to load your application into the target system for debugging:

1. Download the debugger monitor to PromICE directly and then use the debugger to load your application into target RAM or PromICE emulation memory. For the debugger monitor to load data into PromICE memory, the target write line(s) must be connected to **mwr/swr** on the PromICE.

   While this is the fastest way to setup your system, there are disadvantages:

   - Although you can download the debug monitor over the parallel port quickly, you still have to download your application via the serial port.
   - When loading your application into PromICE ROM emulation memory using the target monitor, your application must not overwrite the monitor. If this occurs, the target system will crash during the application download. To recover, reload the monitor using LoadICE and restart the system.

2. Several debuggers support using LoadICE to pre-load the debugger monitor and your application code into PromICE. These debuggers can then begin operation without having to load more code.

   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 using either the host serial or parallel port. This is the fastest way to load code to your target system.

   Once you have the target running with the monitor and application, you can command the debugger to load the symbol information corresponding to the pre-loaded code and begin debugging.

   Also, you can easily detect when a file is being loaded over the other by using the LoadICE **compare** command to verify the monitor and application data after they have loaded.

   To use the **compare** command, first setup Dialog mode (see the introduction in the *LoadICE Environment* chapter). Then use the **compare** ("c") command to verify the files that have been loaded against the files on your disk. If one of the files overlaps the other, the address that is overlapping will be listed. Relocate your application if necessary.

LoadICE Example: AI Setup

Example of a loadice.ini file that loads a file, programs an AI session, and exits.

```
C:\LoadICE>type loadice.ini
output COM1
baud 57600
pponly LPT1
rom 27512
word 8 0
file ai8051.hex 0=0
ailoc 7800 19200

C:\LoadICE>loadice

LoadICE version 4.0 for Windows 95/98
(C) Copyright 1989-99 Grammar Engine Inc.

 Opening initialization file 'loadice.ini'

Connecting.. Please WAIT..
 Opening Serial Port COM1 @BR-57600
 Opening Parallel Port LPT1 (@0x378)
Connecting to PromICE via the Parallel Port.
 Connection established
 AI virtual serial channel to be enabled on exit

EMULATION UNITS PRESENT:
 PromICE ID-0 Memory=512KBytes Emulating=64KBytes FillChar=0xFF Master/AI2
 PromICE ID-1 Memory=512KBytes Emulating=64KBytes FillChar=0xFF Slave of 0
Opening file `ai8051.hex` for processing./Done
Transferred 1856 (0x740) data bytes
 PROMICE ID-0 putting AI in transparent mode
 AILOC = 0x7800 (0xFF7803)
 HostLink - serial port @ 19200 baud
 Binary transparency - no breakCharacter
 Host interrupts (DTR/INIT toggles) to ignore: ALL

LoadICE Exiting with NO Errors
C:\LoadICE>
```

9. AI Command Reference

Overview

AI setup begins with the **ailoc** command. This command programs a PromICE with the AI option to emulate a serial link between a host-based debugger and its target resident monitor. This allows you to debug your target code without using additional target hardware resources. The rest of the AI commands condition and modify how the AI link operates.

The host and target communicate as if they were connected using a conventional serial link. The only exception is that the target monitor will need code to access the AI control registers. This mode of communication is called “transparent” because PromICE does not modify or interpret the data exchanged between the host and the target.

In addition, the host debugger can use PromICE to interrupt or reset the target. The AI interface can support target systems that make burst accesses to the ROM (refer to the preceding setup section).

aicontrol	Controls the AI's access timing
aidirt	Enables AI transmitter to send data written to register ZERO (AI2 only)
aifast	Selects higher speed (unbuffered) transparent communications
ailoc	Specify the address of AiCOM virtual channel
ainorci	Disables per-character receive interrupts
aircvint	Control AI per-character receiver interrupts
aireset	Allows host debugger to reset the AI interface (via DTR or INIT)
aitint	Allows the host debugger to directly interrupt the target
aitreset	Allows debugger to reset the target interface (via DTR or INIT)
aixmtint	Control AI transmitter ready interrupts (AI2 only)
burst	Allows support for burst mode access to ROM interface
intlen	Set AI interrupt length

aicontrol

Control AI timing characteristics

Command Forms

aicontrol loadice.ini file, Dialog mode
-aic Command line

Syntax

aicontrol ***ai1num***

OR

aicontrol ***ai2num***

where

ai1num (integer) Specify an AI1 timing mode:

- 0 Default no delay. Timing derived from target system.
- 1 Insert a 50ns delay for target data sent to host.
- 2 #1 above plus use 50ns delay on target read of data received from the host.

ai2num (integer) Specify an AI2 timing mode for both transmit and receive data:

- 0 Default 20ns delay.
- 1 Use 40ns delay.
- 2 Use 60ns delay.
- 3 No delay. Timing derived from target system.

Default

For AI1, use no delay. For AI2, use 20ns delay.

Description

The AI circuit clocking is based on the timing of target memory accesses. Specifically, clocking is derived from *chip_enable*, *output_enable* and the addresses selected by the Ailoc address mask.

Depending on the target system's memory cycle timing, these signals may need additional settling time to properly clock the AI circuit. The **aicontrol** command selects among delay times for AI memory accesses related to transmitted and received data. This enables the AI clock to be decoded correctly.

Notes

Avoid selecting a delay that is greater than the PromICE emulation access time. The longer delay can cause AI to fail completely.

Examples

aicontrol 1

On AI1, use a 50ns delay for transmitted data. On AI2, use a 40ns delay for transmitted and received data.

aidirt

Enables direct writes for data transmission on AI2 units.

Command Forms

aidirt loadice.ini file, Dialog mode
-aiw Command line

Syntax

aidirt

Default

The target transmits data to the host using the technique that reads the AI registers to transmit bits.

Description

On AI2 units, the **aidirt** command enables the ability for the target to transmit data to the host by writing the data to AI register ZERO.

Examples

aidirt

Enable AI transmission of data written to AI register ZERO.

aifast

Selects non-buffered AI communications.

Command Forms

aifast loadice.ini file, Dialog mode
-aif Command line

Syntax

aifast ***num***

where

num (integer) Specifies an AI timing mode:

0 Disable non-buffered communications

1 Enable non-buffered communications

Default

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

Description

The **aifast** command allows you to disable I/O buffering. Only data from the host for the target can be buffered. When buffering is disabled, communication speed increases but places reliable communications at risk.

Notes

If you expect the target to be busy servicing an interrupt or to receive bursts of traffic from the host, then use the default, buffered method. If buffered I/O works, you can try the non-buffered option for faster throughput.

Examples

aifast

OR

aifast 1

Enable non-buffered communication from loadice.ini file.

-aif 0

Disable non-buffered communication from the LoadICE command line.

ailoc

Setup the AI communication channel between the host and the target.

Command Forms

ailoc loadice.ini file, Dialog mode
-ai Command line

Syntax

ailoc [***id***:] ***address baud*** [***break_char*** [***int_count***]]

where

id (optional integer) A valid PromICE 8 bit unit ID number (0-255). Follow *id* with a colon. This number is the unit ID of the PromICE unit with the AI option. On Dual PromICEs, AI is on the master module. If you are using multiple PromICEs for 16 or 32 bit emulation, the other serial daisy chained units are automatically programmed for "pass through" mode.

address (hex) The base address of the AI control registers as seen by the target system, specified as an offset from the beginning of PromICE ROM-relative emulation memory. This value has these constraints:

On 8 bit systems, the AI interface occupies four consecutive bytes so *address* must be on a quad boundary (i.e. lowest two bits are zero). On 16 bit systems, *address* must be on an octal boundary (lowest three bits are zero). For a 32 bit system, *address* must be on a 16 byte boundary (lowest nibble is zero).

On systems with burst mode ROM accesses, the address must be aligned both by word size and burst count. See *AI Configuration Reference*.

baud (integer) A valid baud rate (1200, 2400, 4800, 9600, 19200, or 57600) for the link between the host serial port and PromICE during AI communications. Debuggers can use a baud rate of zero (0) to invoke AI communications over the parallel port.

break_char (optional hex) A character used to break PromICE out of the AI communication session. Use with ASCII host-target communication protocols. Use -1 to select binary

transparency and use the next argument to specify the condition for breaking the AI communication session.

int_count (optional integer) A number of consecutive host interrupt signals to ignore before PromICE breaks out of the AI communication session. Used with binary host-target communication protocols. Use -1 to ignore all interrupt signals.

Some debuggers will toggle DTR line on startup and thus cause PromICE to restart and break the session. By increasing *int_count*, you can overcome this problem.

Default

The default *id* is 0. The default for the *break-char* is none, which is binary character transparency. The default for *int_count* is to ignore all interrupts from the host. In most cases, you only need specify the address and the baud rate.

Description

The **ailoc** command is used to establish an AI communications session in PromICE at the time the LoadICE application exits. During an AI communications session, PromICE functions as a pass-through data-forwarding device, relaying information between the PromICE serial port and target monitor (via the AI control registers).

In the loadice.ini file, place the **ailoc** command at the end. The *address* and *baud* arguments must be specified. The *address* offset specified is where the target will access the AI control registers. The *baud* argument sets the serial baud rate between the host and PromICE.

Use LoadICE to download a debugger monitor and create the AI session before starting the debugger's host-side front end. Once the AI communication session is established, the host-side debugger can communicate with the target monitor as if they were connected with a standard serial link.

If your debugger supports the PromICE bi-directional parallel port AI protocol, you can have faster AI communication.

Notes

The bus width is determined by the ROM bus width, not the processor’s bus. If you are using a 32 bit processor with one 8 bit ROM, your bus width would be 8 bit for the **word** command.

The *address* is normalized to map to the master PromICE unit according to word size specification (see word size in *AI Configuration Reference*).

In order to run the LoadICE application again, any existing AI session must be broken. LoadICE breaks the session by toggling the serial port DTR line (or parallel port INIT line) 4 times within a sensing period. The sensing period is 5 seconds on PCs and 30 seconds on UNIX. This tells PromICE to break the session.

Examples

ailoc 8 19200

The AI control registers are mapped in the target ROM memory at location (offset from beginning of ROM) 0x8. The host and PromICE will communicate at 19200 baud during the AI session.

ainorci

Disables per-character receive interrupts during AI communications.

Command Forms

ainorci loadice.ini file, Dialog mode
-ainri Command line

Syntax

ainorci

Default

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

Description

By default, the AI receiver will act like an interrupt driven device, i.e. every time a character is available from the host (HDA set), both the interrupt lines, **int-** and **int+**, are asserted. The target clears the interrupt condition by reading the host data. However, the **ainorci** command can be used to disable receiver interrupts. Then the target must poll the status for HDA. Via loadice.ini, 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.

Notes

Use this command when the per character interrupt overhead is not desired or not useful. The debugger can then control the reset explicitly via DTR/INIT and **aitint** command. You may choose not to connect the **int** line to your target.

Examples

-ainri

Disable interrupts from the LoadICE command line.

aircvint

Use this command to control the AI target receiver interrupts, which will cause an interrupt for every character received from the host during AI communications.

Command Forms

aircvint loadice.ini file, Dialog mode
-aircvi Command line

Syntax

aircvint ***num***

where

num (integer) Enables or disables AI receiver interrupts

0 Disable AI receiver interrupts

1 Enable AI receiver interrupts

Default

By default, the AI receiver will act like an interrupt driven device, i.e. every time a character is available from the host (HDA bit set in the AI status register), both the PromiCE interrupt lines, **int**- and **int**+, are asserted. The target clears the interrupt condition by reading the host data.

Description

The **aircvint** command can be used to enable or disable receiver interrupts. When disabled, the target must poll the status for the HDA bit. Via loadice.ini, 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.

Notes

Use this command to disable receiver interrupts when the per character interrupt overhead is not desired or not useful. Then the debugger must control the reset directly via DTR/INIT and **aitint** command.

Examples

-aircvi 0

Disable receiver interrupts from the LoadICE command line.

aireset

Controls a host debugger's ability to reset the PromICE AI option.

Command Forms

aireset loadice.ini file, Dialog mode
-aire Command line

Syntax

aireset [***num***]

where

num (optional integer) specify the mode for AI option reset.

0	Disable debugger-controlled AI reset
1	Enable debugger-controlled AI reset

Default

When not specified, the **aireset** feature is disabled. When specified, the default *num* is 1 for enabled.

Description

The **aireset** command allows the host debugger to signal a reset of the AI option. The debugger signals by toggling the serial DTR or the parallel INIT line. A debugger can use this capability only during an AI communications session. An AI reset clears all AI status bits and discards any data in the AI system. The AI session continues following AI reset.

Notes

Debugger-controlled AI reset is only available during an AI communications session. AI sessions begin when the LoadICE application instructs the PromICE to begin a session and then LoadICE terminates.

This command can be used with **aitreset** command but is not suitable to use with the **aitint** command.

Examples

aireset

OR

aireset 1

Enable the debugger AI reset feature

-aire 0

Disables the debugger AI reset feature from the command line

aitint

Enable a host debugger's ability to signal an interrupt of the target system

Command Forms

aitint	loadice.ini file, Dialog mode
-aiti	Command line

Syntax

aitint

Default

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

Description

When the host debugger toggles the serial DTR line or the parallel INIT line, this PromICE AI feature will generate an interrupt on the target system. The PromICE interrupt output is the **int-/int+** pins on the back of PromICE (see the *AI Configuration Reference*). The **aitint** command allows the host-based debugger to interrupt the target application and return control to the debugger monitor.

Generally you will need to use the **ainorci** command to disable receive character interrupts. Then this command permits the debugger to request interrupts on demand.

Notes

The interrupt is generated for a fixed duration. The default is 100 microseconds and can be changed using the **intlen** command.

This command is not suitable to use with the **aireset** or **aitreset** commands.

Examples

aitint

Enable host debugger-controlled interrupt from the loadice.ini file.

-aiti

Enable host debugger-controlled interrupt from the LoadICE command line.

aitreset

Enables a host debugger to directly reset the target system.

Command Forms

aitreset loadice.ini file, Dialog mode
-aitr Command line

Syntax

aitreset

Default

No target reset can be done.

Description

The **aitreset** command enables the AI feature that allows a host-based debugger to signal PromICE to briefly assert target reset. The debugger signals by toggling the DTR or the INIT line.

Notes

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

This command can be used with the **aireset** command but is not suitable for use with the **aitint** command.

The duration of the reset pulse can be changed (up to 500ms) using the **reset** command in loadice.ini.

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.

aixmtint

Use the **aixmtint** command to activate AI2 target transmitter ready interrupts, which interrupt each time the data transmitter becomes idle during AI communications.

Command Forms

aixmtint loadice.ini file, Dialog mode
-aixmti Command line

Syntax

aixmtint ***num***

where

num (integer) Enables or disables AI transmitter interrupts

0 Disable AI transmitter ready interrupts

1 Enable AI transmitter ready interrupts

Default

Both the interrupt lines, **int-** and **int+**, on the back of PromICE are not asserted to the target when the transmitter becomes idle (TDA bit cleared in AI status register).

Description

By default, the AI2 transmitter will act like a polled device. The target clears the interrupt condition by writing a new character to be sent to the host or reading the status register. The **aixmtint** command can be used to enable or disable transmitter ready interrupts. When disabled, the target must poll the status for the TDA bit.

Notes

This command is available only on AI2 units. Use this command to disable transmitter ready interrupts when the per character interrupt overhead is not desired or useful.

Examples

-aixmti 1

Enable transmitter ready interrupts from the LoadICE command line.

burst

Program the AI option to handle burst mode ROM accesses.

Command Forms

burst	loadice.ini file
-B	Command line

Syntax

burst ***num***

where

num	Specifies the number of reads executed during the cycle. AI1 PromICE units support burst modes of 0, 4, 8, and 16 bytes. AI2 PromICE units support burst modes of 0, 2, 4, 8, 16, and 32 bytes.

Default

Burst mode is disabled.

Description

Some target systems do burst mode reads from ROM addresses. The processor holds chip-select and output-enable low, and reads either 2, 4, 8, 16 or 32 bytes of data. During this time, chip-select and output-enable are held low and the address lines change.

The **burst** command is also needed if the target processor read cycle is greater than the ROM width, such as a long word processor booting from an 8 bit ROM.

This command adjusts the locations of the AI control registers to compensate for the multiple reads generated by each burst read. AI1 PromICE units support burst modes of 0, 4, 8, and 16 bytes. AI2 PromICE units support burst modes of 0, 2, 4, 8, 16, and 32 bytes. For each power of two increase in burst length, PromICE ignores a low order address bit as follows:

Burst	Ignores	Offsets
0	none	0, 1, 2, 3
2	A0	0, 2, 4, 6
4	A1 – A0	0, 4, 8, 12
8	A2 – A0	0, 8, 16, 24
16	A3 – A0	0, 16, 32, 48
32	A4 – A0	0, 32, 64, 96

The target monitor code must account for the control register offsets due to the burst mode accesses. The effect is to insert additional filler bytes between the control registers. This is in addition to the register spacing due to target word size.

Notes

This command is only needed when using the **ailoc** command. Target burst reads work correctly in normal emulation.

Examples

burst 4

Specifies a 4 byte burst in the loadice.ini file.

intlen

Sets the duration of AI interrupt signals.

Command Forms

intlen	loadice.ini file, Dialog mode
-il	Command line
il	Dialog mode

Syntax

intlen ***num***

where

num	(integer) Specifies the length for all AI interrupt signals in milliseconds.

Default

The default interrupt duration is 100 microseconds.

Description

The **intlen** command specifies the duration of all interrupt requests generated by the AI option.

Examples

intlen 150

Set AI interrupt length to be 150 milliseconds.

10. AI Porting

Contents

Overview

Your target sees the AI option as a memory-mapped peripheral located within PromICE emulation memory at an address defined by the LoadICE **ailoc** command.

The AI target-side interface consists of four 8 bit control registers. The AI registers are mapped into the PromICE ROM emulation memory. On dual PromICEs, all four registers are on the master (bottom) PromICE 8 bit unit. Once an AI communications session begins, the AI control registers are accessed by reading these four locations.

AI communications sessions begin when the LoadICE application instructs the PromICE to begin a session and then LoadICE terminates. Once established, the AI session transports data between the host and the target systems.

AI Register Description

This section contains the bit-level 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 ‘zero bit’ to the AI transmit buffer.

ONE
(read only) Offset Address: 1

7							**0**
-	-	-	-	-	-	-	-

Bit #	**Mnemonic**	**Function**
7-0	-	Reading this register sends a ‘one bit’ to the AI transmit buffer.

HOST_DATA

(read only) Offset Address: 2

7							0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Bit #	Mnemonic	Function
7-0	RB7:0	Received Character Buffer, a byte of data from the host

STATUS

(read only) Offset Address: 3

7							0
X	X	X	X	FLG	OVR	HDA	TDA

Bit #	Mnemonic	Function
7-4	-	Reserved. These bits are undefined; for compatibility with future devices. Mask off these bits.
3	FLG	Used to detect AI Registers at startup, value of 0 (see below)
2	OVR	Host data overflow bit
1	HDA	Valid host data in buffer
0	TDA	Data Transmitter Busy

AI Operation

Typically, a host debugger initiates the connection to a target monitor. LoadICE is used to configure the AI option upon its termination. The host debugger then begins trying to contact the target and establishes communication. When the AI communication session begins, the target will see the AI control registers instead of the contents of emulation memory at those addresses.

PromICE AI Initialization

For complete details, see the *AI Configuration* chapter and the **ailoc** and **burst** commands.

Target Initialization

The target software must know where the AI control registers were positioned by the interaction between the LoadICE **ailoc**, **word**, and **burst** commands (see the *AI Command Reference* chapter). Note that on systems with word size 16 or greater, the target software must extract the AI register contents from the correct byte within the word.

Initialize the emulation memory location to be occupied by the AI **STATUS** register with the pattern 0xCC, which has a "1" in **FLG**, bit #3. (When the AI control registers become available, **FLG** in the AI **STATUS** register will be zero.) Use the LoadICE **fill** command or the target software to set the 0xCC pattern.

When the target boots or restarts, the target software should wait for the AI session to be established before attempting to talk to the host. The target monitor enters a loop that polls for the presence of the AI **STATUS** register. When **FLG**, bit #3, in the AI **STATUS** register becomes zero, the AI control registers are now available. The target should discard the first received character.

After an AI communications session begins, the host debugger and the target monitor can now communicate.

Target Data Reception

When **HDA**, bit #1, in the AI **STATUS** register becomes one, the target can read host data from the **HOST_DATA** register. The target should discard the first received character. AI receiver buffers up to 39 characters of host data. If the target does not keep up with received data, then the buffer overflows. When host data is lost, **OVR**, bit 2, is set to one to signal the need for error recovery. Reading **HOST_DATA** will clear **HDA** and **OVR**, if set.

Target Data Transmission

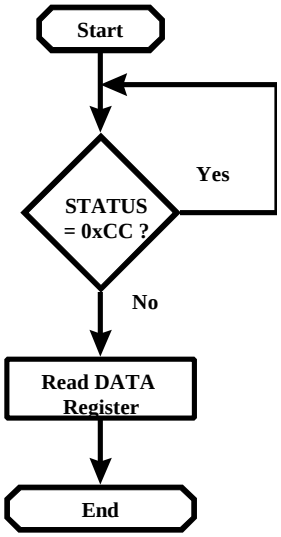
When **TDA**, bit 0, in the AI **STATUS** register becomes zero, the transmitter is available to send data to the host. The procedure for sending data to the host involves a series of read accesses to AI registers **ZERO** and **ONE**. See the following AI Write Algorithm flow chart for details.

While the transmitter is busy, **TDA**, bit 0, in the AI **STATUS** register will remain one.

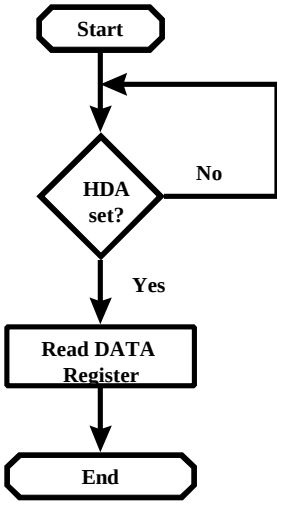
On AI2 units, when the target write line has been connected to PromICE, the target can write byte data to register **ZERO** for transmission to the host. Use the **aidirt** command to enable transmission by register **ZERO** writes.

AI Algorithms

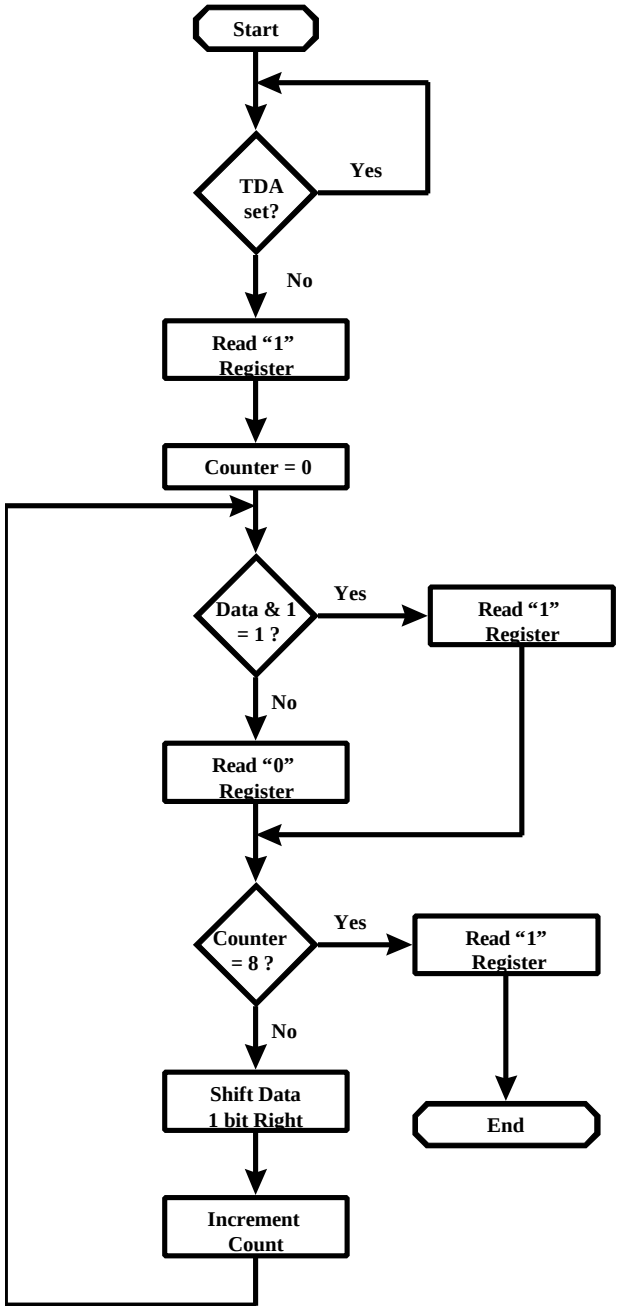
Initialization Algorithm



Read Algorithm



Write Algorithm



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. This example provides simple AI Register data structures for various word size configurations where the PromICE 8 bit unit containing the AI Option is in the lower order byte of the word size.

Note: If the **burst** command is used or the AI Option is not on the PromICE 8 bit unit corresponding to the byte at the lowest memory address within the word, you will need to create a custom data structure (see *AI Configuration Reference*).

```
#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 AI regs. to become  *
 * available and for data to be received from  *
 * the host.  The first byte read from the AI  *
 * at startup is not valid.                    *
 ***********************************************/
void AIinit(void)
{
 unsigned char dummy;
 while(AI->STATUS == 0xcc){}  // Wait for bit #3
                              // to become zero
 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 while data
                              // transmitter busy
 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
fillall cccc
noverify
file=aitest.hex 80000=0
ailoc 2000, 57600
```

11. AI Troubleshooting

This chapter should help to resolve communications problems you may encounter when using a debugger with PromICE. Following these simple issues is a section on debugger diagnosis.

Common Problems

Target using Burst Mode access to ROM

If your target makes burst mode accesses to ROM, add the **burst** command to your loadice.ini file. Refer to the **burst** command in the *AI Command Reference* chapter.

Receiving Bad Data Back from Target

Add the **aicontrol** command to your loadice.ini file. Depending on your target system, you may want to try other values. Refer to the **aicontrol** command in the *AI Command Reference* chapter.

Debugger crashes during download

If your target debugger crashes at some point while downloading your application into PromICE memory, your application may be overwriting the debugger monitor.

For an alternative, see *Download Options* in the *AI Configuration* chapter.

Watchdog Timer

If you have a watchdog timer on your target that is not used by the debugger, then disable it in the monitor startup code. The debug monitor will not come up if a watchdog timer is enabled but not configured by the monitor.

Failure to establish an AI Communications Session

This can occur if you are trying to connect your debugger through an AI communications session in one window while LoadICE is running in another window. AI sessions are only established at the time that the LoadICE application exits.

Byte Order Swapped

Debuggers usually expect the AI option, which is on the master PromICE 8 bit unit, to be mapped to the low order byte within the target word. See the *AI Configuration* chapter for details.

Can't Interrupt Target

In order to bring control back to the monitor from a free-running application or to enable interrupts for character-level communications, you need to hook the PromICE interrupt line. See the *AI Configuration chapter*.

The target monitor must be configured to handle the interrupt. Refer to your debugger's monitor documentation for information on interrupt driven communications.

Can't Write, Single Step or Set Breakpoint in ROM Emulation Memory

In order for the target to write into the ROM emulation memory, the PromICE write input pins must be connected to the target's write line. See the *AI Configuration Reference*.

Some target systems will not allow write cycles to the ROM addresses. Usually a PAL controls this. Determine if the target supports write cycles to ROM addresses and make the appropriate modifications, if necessary.

DEBUGGER DIAGNOSIS

Never Worked

Step 1: Verify your loadice.ini configuration. First configure PromICE to emulate some known working code, if available. 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 next items to check in your loadice.ini file would be the **socket**, **file**, and AI control commands.

Otherwise repeat 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. 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** command is probably incorrect. Verify that the **ailoc** address is on the correct boundary and is at the beginning of the AI mapped area.

Verify that the baud rate specified in the **ailoc** command matches the baud rate specified to the debugger. If you are still getting incorrect characters back from the target, add the **aicontrol** command to your loadice.ini file.

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.

Note: You should be able to see the Rx and Tx lights flashing on the front panel of PromICE as the host and target attempt to communicate with each other.

Worked previously

If something fails (i.e. The debugger can't connect and/or the monitor crashes) then carefully review all recent changes to your configuration and/or source files. Go back over your configuration, step by step. Look at even the simplest changes.

AI Technical Support

If you have a question, feel welcome to call Grammar Engine. We will answer your question or direct you to someone who can. Here is a referral index:

Call The Debugger Vendor for:

Compiler and/or locator questions.
Target specific monitor configuration questions.
Debugger switches and options.

Call Grammar Engine Technical Support for:

PromICE / LoadICE questions.
AI specific configuration questions.

12. PromICE Unit IDs

This chapter supplements the LoadICE *Command Reference* by showing how to use PromICE 8 bit unit numbers or IDs with LoadICE commands. For all other information on LoadICE commands, see the *Command Reference* chapter.

Contents

PromICE 8 bit units

PromICE technology is built on emulation memory units, which are 8 bits in data width and the memory size of the particular unit. Ordinarily, the 8 bit units need only to be included in the **word** command's units list, as a part of the general purpose ROM emulation configuration. Occasionally it is useful to work with individual 8 bit units.

Uses for 8 bit unit configurations

The following examples illustrate the benefit of working directly with the PromICE 8 bit units.

Split 16 bit files

On 16 or 32 bit word size ROM configurations, the load files may have been split into byte width files by a utility. The load files, having been split into 8 bit versions, are loaded by specifying the individual PromICE 8 bit units. Then the general 16 bit ROM configuration can be used to work with the emulation memory as whole words.

For example:

```
word 16 1 0
file lowbyte.hex 1000=0 8 1     * load low order data into unit #1
file hibyte.hex 1000=0 8 0      * load high order data into unit #0
load
dump                            * displays 16 bit words
```

16 bit system with 8 bit boot ROM

The target system has an 8 bit boot ROM in addition to a main memory (8, 16, or 32 bits wide). The boot ROM is connected to a PromICE 8 bit unit that is not included in the **word** command's unit list. The **word** command is used to specify the main memory's word width.

For example:

```
word 16 0 1
file maincode.hex 1000=0        * load main code into 16 bit ROM
file bootcode.hex 0=0 8 2       * load System boot code into unit #2
load
```

PromICE 8 bit unit configurations

Each 8 bit unit can have its own configuration information. The **rom** and **socket** commands, used with the unit "*id*:" argument, modify that unit's configuration information.

Note: When using the **rom** or **socket** commands without an unit ID argument, **all** PromICE 8 bit unit configurations are changed, including those not listed in the **word** command. Also, 8 bit units included in the **word** command's unit ID list must not be reconfigured at the 8 bit unit level.

To setup a general configuration along with unit-specific configurations:

1. First build the general PromICE ROM emulation configuration and specify the 8 bit units list on the **word** command.
2. PromICE 8 bit units, not used in the general configuration (not in the **word** command's units list), can be individually configured by specifying their *id*: as an argument.

The individual units as well as the units belonging to the general configuration can be individually referenced using the 8 bit unit *id*: argument with the commands documented in the following section.

Using one 8 bit unit at a time

Working with 8 bit units individually is accomplished with the **word** command. While the **rom** and **socket** commands can modify an 8 bit unit's configuration, the **word** command controls which units are in the current configuration.

The following example repeats the 'Split 16 bit files' example above but works with each unit, one at a time.

```
begin
word 8 1
file lowbyte.hex 1000=0
load                    * load low order data into unit #1
word 8 0
file hibyte.hex 1000=0
load                    * load high order data into unit #0
word 16 1 0
dump                    * displays 16 bit words
```

Commands Supporting PromICE 8 bit Unit Ids

This section is a supplement to the *Command Reference* chapter. This section specifically shows how to use PromICE 8 bit unit IDs with these selected LoadICE commands.

For all other command documentation, see the *Command Reference* chapter.

Most of these command share this attribute:

id (optional integer) A valid PromICE unit ID number (0-255). Operate only on PromICE 8 bit unit *id*, overriding the current PromICE ROM configuration. Follow *id* with a colon.

The default for unit ID is 0.

checksum

Perform checksum on ROM and store result in PromICE emulation memory.

Syntax

checksum [***id*** **:**] ***start end store*** [***function***] [***sum_size***] [***order***]

Examples

checksum 1: 0 fffb fffc a 32

Compute the checksum on 0 through FFFB on unit ID=1.

dump

Display contents of PromICE emulation memory on the screen.

Syntax

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

Examples

d 1:100 ff

Dump data from unit ID-1 from 0x100 to 0x1FF.

edit

Modify PromICE ROM emulation memory.

Syntax

edit [***id*****:**] [***address*** { ***value*** }]

Examples

edit 1:500 ab cd

Set locations 0x500 and 0x501 in PromICE unit 1 to values 0xAB and 0xCD.

file

Setup a hex record file for ROM emulation or a comparison operation.

Syntax

file ***file_name file_address = ROM_offset*** [***word_width unit_ID*** { ***unit_ID_n*** }] [**(** ***range_start, range_end*** **)**]

where

word_width (integer) defines the word width for the file. It must be an integral multiple of 8 and can not be larger than the maximum data bus width determined by the total number of PromICE 8 bit units.

unit_ID (integer, required if *word_width is present*) The first PromICE 8 bit unit number corresponding to the byte at the lowest memory address within the word.

unit_ID_n (optional integers following *unit_ID*) The nth PromICE 8 bit unit number specifying the byte order within the word size defined by *word_width* for this file configuration.

Notes

You need not use the arguments listed above unless you are loading files "per ROM" (i.e. one file gets loaded into a PromICE 8 bit unit and another file gets loaded into another PromICE 8 bit unit). If you have a word size of 16 or larger, then specifying these arguments forces the file to load into the PromICE 8 bit unit with that ID. For example, forcing 8 bit loads is necessary if your 16 bit hex file has already been split by a utility into two 8 bit files.

On targets with word size greater than 16, you can use other values for *word_width*. For instance, if the target's 32 bit word is made up of 16 bit ROMs then use *word_width* equal to 16 and list two unit IDs to load 16 bit slices of the target code.

The old **file** command syntax using the *id:* argument has been retired. Use the syntax shown above to specify PromICE 8 bit unit IDs.

Examples

```
word 16 1 0
file=myfile.hex 400000=0 8 1
```

On a 16 bit target, load data from '*myfile.hex*' with addresses starting at 0x400000 in the file to be mapped to 0x0 into emulation memory of PromICE 8 bit unit number one. The file must contain 8 bit data.

```
word 32 0 1 2 3
file=myfile.hex 400000=0 16 2 3
```

On a 32 bit target, load data from '*myfile.hex*' with addresses starting at 0x400000 in the file to be mapped to 0x0 into emulation memory configuration including of PromICE 8 bit unit numbers two and three. The file must contain 16 bit data.

fill

Write a fill pattern into emulation space in the individual PromICE 8 bit unit.

Syntax

fill [***id*:**] ***start end data*** [***data2***] ***fill_size***

Notes

You can specify unique fill parameters for each ROM using the *id*: parameter.

Examples

```
f 1:200 300 ab
```

Fill unit ID=1 from 0x200 to 0x300 with data value 0xAB

fillall

Fill the entire 8 bit PromICE memory unit capacity with a repeating data pattern.

Syntax

fillall [***id*:**] ***data*** [***data2***] ***fill_size***

Examples

fillall 0:ab

Fill all unit ID=0 PromICE memory with data value 0xAB

find

Find binary data pattern in the PromICE 8 bit unit's emulation memory.

Syntax

find [***id*:**] ***start end find_count data_byte*** { ***data_bytes*** }

Examples

F 0:0 1ffeo 4 de ad fe ed

Looks on unit ID=0 for the byte pattern 0xDE 0xAD 0xFE 0xED in a 128kB ROM space, in the ROM-relative address range 0x0 to 0x1FFE0. For example:

0000021C: DEADFEED
00000340: DEADFEED

image

Setup a binary file for ROM emulation or a comparison operation.

Syntax

image ***file_name skip_count =*** [***id*:**] ***ROM_offset***

OR

image ***file_name skip_count = ROM_offset*** [***word_width unit_ID*** { ***unit_ID_n*** }] [**(*start_offset, end_offset*)**]

where

.id	(optional integer) A PromICE 8 bit unit ID number (0-255). Specifying *id* forces a load to that unit *id* using an 8 bit word size. This temporarily overrides the system word size defined by the **word** command. Follow *id* with a colon. **Do not use *id* with *word_width*** (see below).
word_width	(integer) defines the word width for the file. It must be an integral multiple of 8 and can not be larger than the maximum data bus width determined by the total number of PromICE 8 bit units.

Notes

You need not specify the ID (*id:*) unless you are loading files "per ROM" (i.e. one file gets loaded into a PromICE 8 bit unit and another file gets loaded into another PromICE 8 bit unit). If you have a word size of 16 or larger, then specifying *id*: forces the file to load into the PromICE 8 bit unit with that ID. For example, forcing 8 bit loads is necessary if your 16 bit image file has already been split by a utility into two 8 bit files.

Examples

image=myfile.bin 20=1:0

Load data from *myfile.bin* with addresses starting at 0x21 in the file to be mapped to 0x0 into emulation memory of PromICE 8 bit unit number one. The file must contain 8 bit data.

move

Copy bytes within the PromICE 8 bit unit's emulation memory.

Syntax

move [***id*:**] ***start end destination***

Examples

m 1:100 120 300

Move data from 0x100 to 0x120 to 0x300 (to 0x320) on 8 bit unit #1.

promiceid

Display the PromICE Identification number.

Syntax

promiceid [***id*:**]

Examples

promiceid 2:

Report the PromICE Identification number on the third unit (ids are zero-based).

rom

Specify ROM emulation memory size.

Syntax

rom [***id*:**] *size* [k]

OR

rom [***id*:**] gen_*part_number*

Examples

```
word 16 0 1
rom=2:128k
```

Specify that unit #2 emulate a 128K Byte (1Mbit ROM) separately from the general memory configuration which uses units #0 and #1.

save

Save PromICE 8 bit unit's emulation memory contents to a binary file on the host.

Syntax

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

Examples

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

Save the data from unit-0 from 0x100 to 0x3FFF to a file called “newfile.bin” on the host.

search

Search PromICE 8 bit unit's emulation memory for an ASCII data pattern.

Command Forms

search	Dialog mode
S	Dialog mode

Syntax

search [***id*:**] [***start end***] ***pattern***

Examples

S 1:0 1000 "Enter new value:"

Search unit #1's 8 bit memory from 0x0 to 0x1000 for the string.

socket

Specify handling for unused upper address lines. See the *Unused Address Lines* chapter.

Syntax

socket [***id*:**] *size* [k]

OR

socket [***id*:**] gen_*part_number*

Examples

socket=1:27040
rom=1:27010

Lets you emulate a 128K Byte ROM in a socket wired for a 512K Byte ROM using PromICE unit #1.

test

Test a PromICE 8 bit unit's ROM emulation memory.

Syntax

test [***id*** [***pass_count***]]

where

id	(optional integer) A valid PromICE unit ID number (0-255). Do **not** follow *id* with a colon.

Examples

t 3

Test unit-3 once.

version

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

Syntax

version [***id:***]

Examples

version 3

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

13. Unused Address Lines

This chapter explains how to configure PromICE to handle unused address lines using LoadICE commands. Floating address lines must be physically pulled high or low for PromICE to work properly.

The recommended approach is to use as much of PromICE emulation memory as possible and to pull up the remaining address lines on the target's ROM socket. Use 10K pull up resistors to Vcc.

If pull-ups are not possible, this chapter delves into the LoadICE commands that may be needed to get PromICE to emulate and the AI option to function properly.

This chapter contains information on the:

- **socket** command
- **xmask** command
- AI 1-related issues (uses all address lines up to A20)

NOTE: **PromICE will not emulate properly if there are any floating address lines** on the ROM socket. Likewise, AI1 will fail when there are floating address lines.

socket

Modifies the PromICE address mask when address lines, available from the target ROM socket, are not used by ROM emulation and are not pulled high.

Command Forms

socket loadice.ini file, Dialog mode

Syntax

socket ***size***[**k** | **m**]

OR

socket **gen_*part_number***

where

size (integer) The socket size, measured in the current word size (bytes or words) as defined by the most recent **word** command. The specified number must be a power of 2.

k (optional literal) Use to indicate the *size* in thousands of the current word size.

m (optional literal) Use to indicate the *size* in millions of the current word size.

gen_part_number (string of digits)
JEDEC standard ROM part number for either 8 or 16 bit word size. The part number must begin with a 27 or a 29.

Default

The default socket size is the same as ROM size. As a result, all address lines beyond the emulation ROM address space must be high.

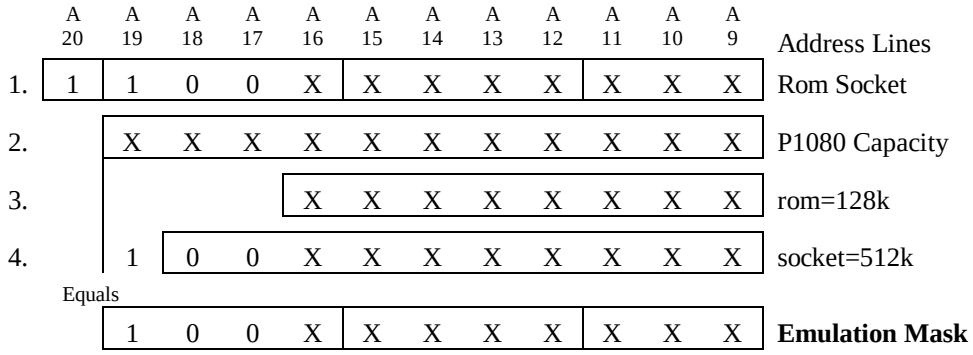
Description

If your target's socket has more address lines than ROM you are emulating, and those unused address lines are not pulled high, then you will need this command and/or the **xmask** command to set the PromICE address mask correctly. The **socket** command works by left shifting zeros into the PromICE emulation address mask beyond the emulation size. If the address mask created by shifting zeros is insufficient, use the **xmask** command to set an arbitrary address bit mask.

This command, along with the **xmask** command, affects both memory emulation and the AI option configuration by providing LoadICE the appropriate mask for the address lines beyond the emulation size.

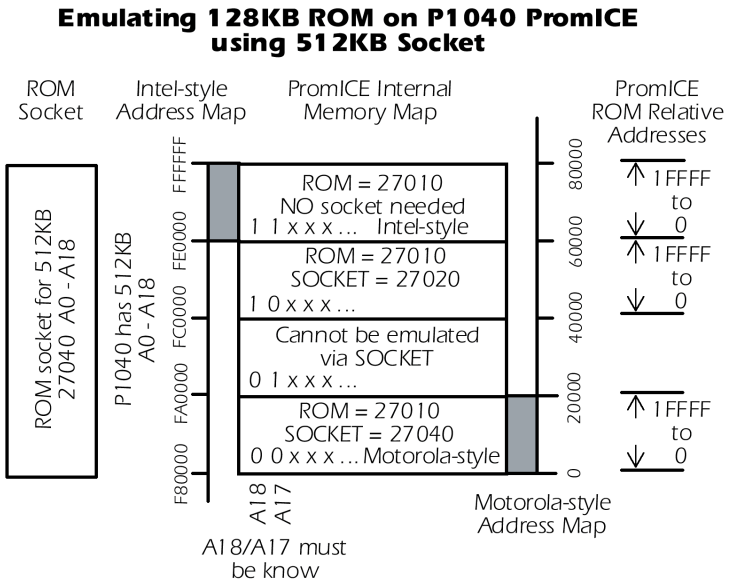
The following diagram illustrates the address line relationships between:

1) A target ROM socket with A0 to A20.
2) The capacity of a P1080 PromICE emulation memory with A0 to A19.
3) A 128KB ROM emulation size with a base address of 0x180000.
4) The use of the **socket** command to right shift two zeros beyond the emulation size.



The actual emulation address mask is created by a bitwise AND of the mask created by the **socket** command with the 8 bit mask specified in the **xmask** command. The emulation address mask must be accurate to the PromICE memory capacity.

Examples



socket=27040
rom=27010

Lets you emulate a 128K Byte ROM in a socket wired for a 512K Byte ROM.

socket=1m
rom=256k

Lets you emulate a 256K byte ROM in a socket wired for a 1M Byte ROM.

word 16 1 0
socket=27040
rom=27010

Lets you emulate a 128K Word ROM in a socket wired for a 512K Word ROM.

word 16 1 0
socket=1m
rom=256k

Lets you emulate a 256K Word ROM in a socket wired for a 1M Word ROM.

xmask

Specify an arbitrary address mask to be used by PromICE when address lines, available from the target ROM socket, are not used by ROM emulation and are not pulled high.

Command Forms

xmask loadice.ini file

Syntax

xmask mask_byte

where

mask_byte (hex) The address mask byte for target address lines A16 to A24.

Default

The default *mask_byte* is 0xFF size.

Description

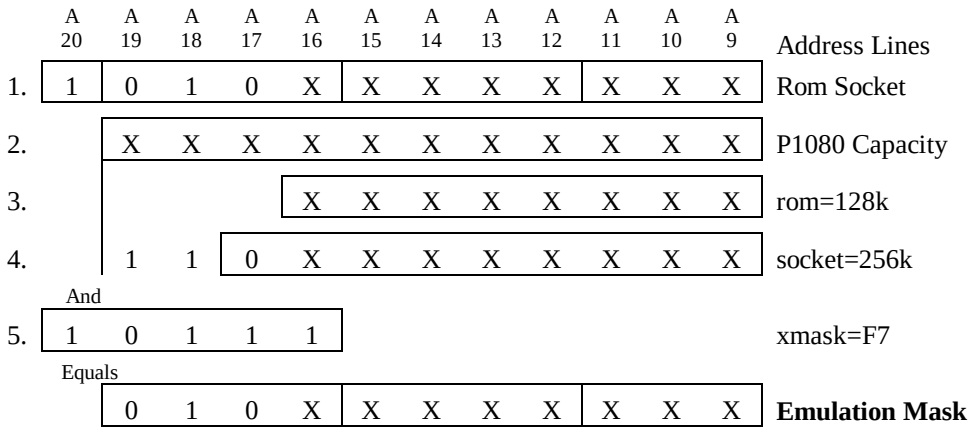
If your target's socket has more address lines than ROM you are emulating, and those unused address lines are not pulled high, then you may need this command to set the PromICE address mask correctly. The **xmask** command works by using a bit-wise AND of *mask_byte* onto the PromICE emulation address mask. In this way, address bits in the mask can be set to zero.

This command affects both memory emulation and the AI option configuration by providing LoadICE the appropriate mask for the address lines beyond the emulation size.

The address mask is controlled by two commands. The **socket** command can right shift zeros into the mask. The **socket** command can shift starting at A11 and up. The **xmask** command controls 8 bits, A16 to A24.

The following diagram illustrates the address line relationships between:

1) A target ROM socket with A0 to A20.
2) The capacity of a P1080 PromICE emulation memory with A0 to A19.
3) A 128KB ROM emulation size with a base address of 0x140000.
4) The use of the **socket** command to right shift a zero beyond the emulation size.
5) The use of the **xmask** to specify the upper address lines.



The actual emulation address mask is created by a bitwise AND of the mask created by the **socket** command with the 8 bit mask specified in the **xmask** command. The emulation address mask must be accurate to the width of the PromICE capacity.

Examples

xmask F3

Set address mask to 0xF3FFFF, which sets address lines A17 and A18 to zero.

On AI 1 units: setup the address mask, if needed

AI2 users can skip this item because AI2 register address decoding ignores all address lines beyond the ROM emulation size.

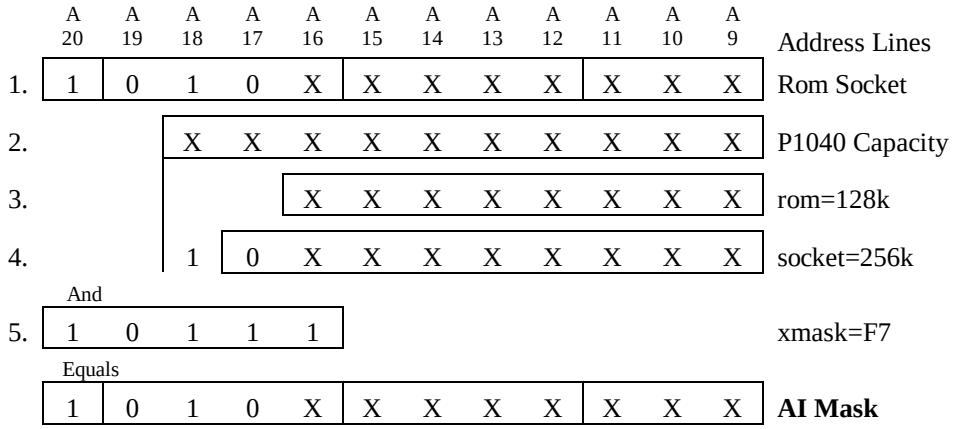
On AI1 units, the register address decoding uses all address lines up to A20. If all the unused address lines on the target ROM socket are not pulled high, then AI1 may not work even if PromICE memory emulation works correctly. In this case, use the **socket** and/or **xmask** commands as explained above to establish the correct address mask.

The AI1 address mask is controlled by two commands. The **socket** command can right shift zeros into the mask. The socket command can shift starting at A11 (2716 ROMs) and up. The **xmask** command controls 8 bits, A16 to A20. The AI1 address mask must be accurate to A20 or the highest order address line on the target ROM socket.

NOTE: 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 must be handled using the address mask for AI1 to work properly.

The following diagram illustrates the address line relationships between:

1) A target ROM socket with A0 to A20.
2) The capacity of a P1040 PromICE emulation memory with A0 to A18.
3) A 128KB ROM emulation size with a base address of 0x140000.
4) The use of the **socket** command to right shift a zero beyond the emulation size.
5) The use of the xmask to specify the upper address lines.



The actual AI address mask is created by a bitwise AND of the mask created by the **socket** command with the 8 bit mask specified in the **xmask** command. The resulting mask, combined with the address specified by **ailoc**, is used to detect target accesses to the AI registers.

14. Technical Specifications

Identification

The format of PromICE model numbers can be any of the following:

P*snnn-aa*

P*snnn-aa*-AI

P*snnn-aa*-AI2

P*snnn-aa*-AI2-*ttt*

where

s (integer) count of 8 bit units:

- 1 Single (master) 8 bit unit for emulating 1 ROM.
- 2 Dual 8 bit units (master and slave in one case) for emulating two 8 bit ROMs (with separate chip selects) or one 16 bit ROM.

nnn (integers, optional letter suffix) capacity of the unit's emulation memory as follows:

nnn	Emulation per 8 bit unit	Generic Part
512	2 KB to 64 KBytes	27512
010	2 KB to 128 KBytes	27010
020	2 KB to 256 KBytes	27020
040	2 KB to 512 KBytes	27040
080	2 KB to 1 Mbytes	27080
160	2 KB to 2 Mbytes	27160

The optional letter suffix is a record keeping designation.

aa (integers) Emulation memory access time in nanoseconds. Values include 35, 45, and 90ns. Older units may have values including 55, 70, 100, 120, and 150ns.

AI (literal) AI1 Communications Option. Older units may have AI0.

AI2 (literal) AI2 Communications Option

ttt (alphanumeric) Trace/Code Coverage Options include:

- DT 512KB Trace and 512KB Code Coverage memory
- DTE 512KB Trace and 512KB Code Coverage memory, external inputs

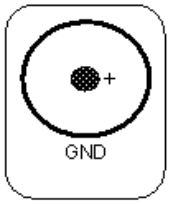
1E	128KB Trace and 128KB Code Coverage, external inputs

Older Trace/Code Coverage Options include:

1	128KB Trace and 128KB Code Coverage memory
128	128KB Trace and 128KB Code Coverage memory
2	128KB Trace and 256KB Code Coverage memory

External Power Supply

The original PromICE units use a 9Volt DC 1A (unregulated) wall plug-in power supply with a 2.1mm Pin and sleeve plug with positive in the center and sleeve as ground.



VDC

The newer PromICE units use a 5Volt DC 2.2A (regulated) desk top power supply with a 2.5mm Pin and sleeve plug with positive in the center and sleeve as ground. These units include the P2160-45 and P2160-35 and all AI2, Trace, and FlashICE units.

Power Consumption

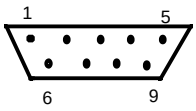
Power consumption will vary depending on the buffers in the ROM emulation interface. Also, faster models consume more power due to faster SRAMs and buffers. The figures below are based on a 4 Meg unit built with 4 1Mbit SRAMs and the ALS buffers. The external power supply is the 9VDC ~1A (unregulated) one.

PromICE Master: +5VDC < 200mA
PromICE Slave: +5VDC < 150mA
PromICE Analysis Interface: +5VDC < 70mA

Interfaces

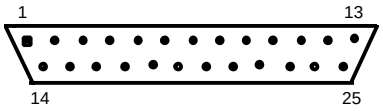
Serial Interface / DB9 female

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.



Parallel Interface / DB25 male header

It is a Centronics compatible parallel printer port configured for direct connection to DB25 connector on the back of PCs 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	14
2	D0	ERROR	15
3	D1	INIT	16
4	D2	SELECTIN	17
5	D3	GND	18
6	D4	GND	19
7	D5	GND	20
8	D6	GND	21
9	D7	GND	22
10	ACK	GND	23
11	BUSY	GND	24
12	PE	GND	25
13	SELECT		

The data transfer protocol is modified so that the parallel port can be used bi-directionally. 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

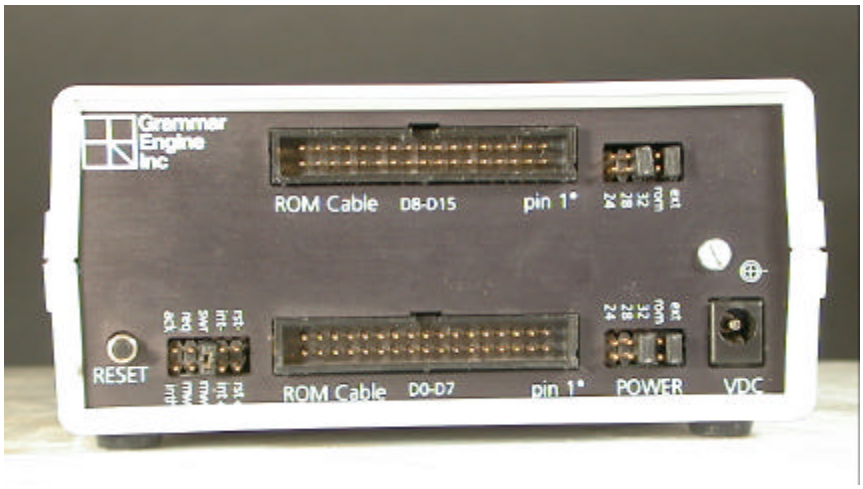
has not read the previous data from the host or it has data to send to the host. The SELECTIN signal is used to acknowledge the state of the BUSY line to PromICE. This ensures that the sense of the BUSY line is never confused by the host.

When sending data to the host, PromICE places data 4-bits at a time on the ACK, PE, SELECT, and ERROR lines and asserts the BUSY signal.

When two PromICE units are daisy-chained on a 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.

PromICE Back Panel

The back panel of a dual PromICE is shown below.

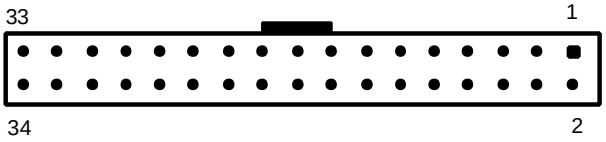


Note the dual set of ROM cable headers and power headers. Each set configures one of the two 8 bit units inside the dual PromICE.

The RESET button will reset PromICE's internal processor and the target, if the reset line labeled **rst** is connected to the target. Reset does not affect the contents of PromICE emulation memory.

ROM Cable Header

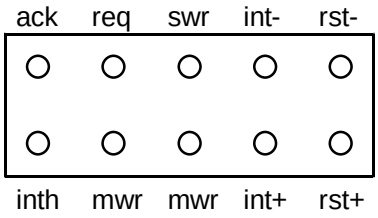
Pinout for PromICE ROM cable header as seen from the back of the unit.



PIN#	SIGNAL	SIGNAL	PIN#
1	GND	A0	26
2	A20	A1	24
3	VCC-32	A2	22
4	A19	A3	20
5	A18	A4	18
6	A16	A5	16
7	A17/vcc-28	A6	14
8	A15	A7	12
9	A14	A8	13
10	A12	A9	15
11	A13/vcc-24	A10	21
12	A7	A11	17
13	A8	A12	10
14	A6	A13/vcc-24	11
15	A9	A14	9
16	A5	A15	8
17	A11	A16	6
18	A4	A17/vcc-28	7
19	OE_	A18	5
20	A3	A19	4
21	A10	A20	2
22	A2	CS_	23
23	CS_	OE_	19
24	A1	D0	28
25	D7	D1	30
26	A0	D2	32
27	D6	D3	33
28	D0	D4	31
29	D5	D5	29
30	D1	D6	27
31	D4	D7	25
32	D2	VCC-32	3
33	D3	GND	34
34	GND	GND	1

vcc-*nn* -> is the vcc for the appropriate pin device

Optional Connection Header



The Optional Connect Header is on the PromICE back panel. It has auxiliary signals that allow you to control the target system from the host, as well as additional features that allow 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 74LS125 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 during AI communications, or when caused by a host command. Both polarities of the signal are provided. They are driven by a 74LS125 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.

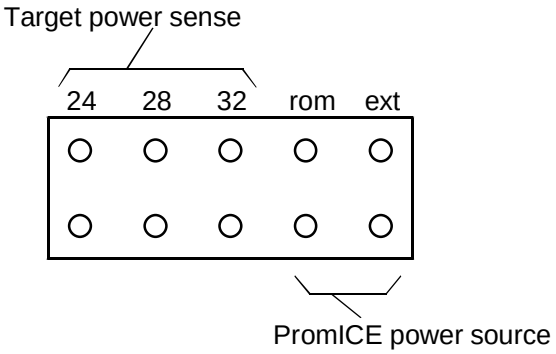
mwr / swr: (input) These are low asserted inputs that are connected to the system write line on the target. The target can do write cycles to PromICE ROM emulation memory. The **mwr** pin is used to write to the master unit (bottom back connector). The **swr** pin is used to write to the slave unit (upper back connector) of a dual PromICE (model prefix P2*xxx*). To do byte writes to 16 bit ROM emulation memory, you must attach two separate write signals to the **mwr** and the **swr** inputs.

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

req: (output) This signal is driven directly by the 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.

Power Header



Always use the **ext** jumper to supply external power to PromICE. Use the power supply shipped with that PromICE.

Use the **rom** jumper only when connecting the 3 volt adapter or to parasitically power a small, low power target. Your target should not parasitically power the PromICE unit.

Use the **32** jumper for all other cables except the 24 and 28 pin DIP. Use the **24** jumper when using a 24 pin DIP cable. Use the **28** jumper when using a 28 pin DIP cable.

When using the GEI 3 Volt Adapter, use the **EXT**, **ROM**, and **32** jumpers. The 3 Volt Adapter ships with a spare, long handled jumper on one of the **Write** pins that you can use for the **ROM** jumper.

Front Panel 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. PromICE will not exit Load mode if target power is not detected.

Enclosure

Dual PromICE with AI: 5.08" Wide, 2.5" High w/o rubber feet, 5.25" Deep, Impact-resistant, ABS-molded, Grade DFA/R Plastic.

All other PromICE models: 5.08" Wide, 1.5" High w/o rubber feet, 5.25" Deep, Impact-resistant, ABS-molded, Grade DFA/R Plastic.

Environmental Restrictions

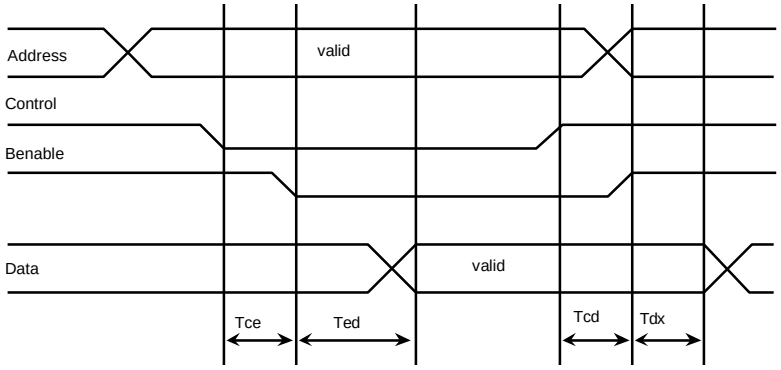
Operating Temperature: 5 to 32 degree C (41 to 90 degrees F)
Storage Temperature: -40 to 70 degrees C (-40 to 158 degrees F)
Humidity: 90% maximum without condensation.

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 - 22ns, Ted - 12ns, Tcd - 22ns, and Tdx - 12ns.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X