



# **EMUL296™-PC**

## **User Guide**

We would appreciate any feedback about the product (including the manual) ranging from simple software defects to suggestions on how to improve the examples.

Thank You.

# EMUL296™-PC

## User Guide

Copyright © 1996 by ICE Technology

ICE Technology  
Tel: 650.375.0409 - 800.686.6428  
Fax: 650.375.0409  
BBS: 650.375.8666  
URL: <http://www.icetech.com>  
E-Mail: [support@icetech.com](mailto:support@icetech.com)

All rights reserved worldwide.  
Edition 1

<b>Development Team:</b> Jim Hayes Michael Bao Joey Zhuo Jörgen Andersson	<b>Documentation:</b> Jim Hayes Wolfgang Weller Jörgen Andersson

## Warranty Information

The EMUL296™-PC Emulator board, Trace board, Pods, Emulator Cable, and LanICE hardware are sold with a one-year warranty starting from the date of purchase. Defective components under warranty will either be repaired or replaced at ICE Technology discretion.

Pods that use a bond-out processor are also warranted for one year from the date of purchase except for the processor. The bond-out processor will be replaced once if support determines that the failure in the bond-out processor was not due to user's actions. This replacement limit does not apply to the rest of the pod.

Each optional adapter, cable, and extender is sold with a 90-day warranty, except that it may be subject to repair charges if damage was caused by the user's actions.

The EMUL296™-PC Emulation software is sold with no warranty, but upgrades will be distributed to all customers up to one year from the date of purchase.

ICE Technology makes no other warranties, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. In no event will ICE Technology be liable for consequential damages. Third-party software sold by ICE Technology carries the manufacturer's warranty.

<p><b>Warning:</b>    <i>Always turn on the emulator before applying power to the target. Always turn off the target power before turning off the emulator power.</i></p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------

# Table of Contents

<b>Table of Contents</b>	<b>5</b>
<b>Introducing EMUL296™-PC</b>	<b>9</b>
Introduction to EMUL296™-PC .....	9
How to use this manual .....	9
If you are new to emulators of any kind .....	9
If you have used emulators with other microprocessors .....	9
If you are familiar with emulators, MS Windows, and the chip, .....	10
Manual Conventions .....	10
Quick Installation Instructions .....	10
System Requirements .....	10
Quick Setup Instructions .....	10
Installing the Emulator .....	10
Installing the Trace Board (If Used) .....	11
Installing the Pod .....	11
Installing the Software .....	12
Initial Software Configuration .....	12
Confidence Test .....	13
Quick Start Instructions .....	13
<b>Chapter 1: Software User Interface</b>	<b>17</b>
Detailed Software Installation Instructions .....	17
Initial Software Configuration .....	17
Configuring the Software .....	18
Projects .....	18
Creating a Project .....	19
Setting the Paths .....	19
Mapping memory .....	21
Emulator Hardware Configuration .....	21
Setting the Chip Config Registers .....	22
Miscellaneous bits .....	24
Miscellaneous Configuration .....	25
Enable Code Space Limits .....	26
Window Colors .....	27
Reset vs. Full Reset .....	28
Trace Config Menu .....	29
Fast Break Write .....	29
Memory Coverage .....	30
Summary Memory Coverage Report .....	32
Detailed Memory Coverage Report .....	33
Performance Analysis .....	33
Menus .....	36
File Menu .....	36
Programming External FLASH Memory .....	37
Programming Algorithms .....	38
View/Edit Menu .....	40
Run Menu .....	41

Breakpoints Menu .....	42
Config Menu .....	44
Program Menu .....	45
Source Menu .....	46
Data Menu .....	46
ShadowRam Menu .....	47
Register Menu .....	47
Trace Menu .....	47
Stack Menu .....	47
Watch Menu.....	48
Window Menu.....	48
Help Menu .....	50
Dialog Boxes.....	50
Child Windows .....	50
Register Windows .....	50
Data and Shadow RAM Windows .....	51
Custom Display Format.....	52
Program Windows.....	52
In-line Assembler .....	53
Source Windows.....	53
Trace Window .....	54
Other Windows .....	54
Inspect Window.....	55
Watch Window.....	55
Evaluate Window .....	55
Stack Window .....	55
RTXC Window .....	56
Tool Bar.....	56
Help Line .....	56
Dynamic Data Exchange.....	57

## **Chapter 2: Emulator Macro User Guide 59**

Introduction .....	59
General description of the emulator macro setup .....	59
Visual Basic Supplemental User Guide.....	59
General information .....	60
Procedure for writing a macro .....	60
Subroutine Reference.....	64
Windows API.....	64
Nohau Subroutines .....	64

## **Chapter 3: Emulator Board 69**

EMUL/LC-ISA Emulator Board.....	69
Detailed Installation Instructions.....	70
Setting the I/O address jumpers -- J1 .....	70
Setting the Target Communication Rate -- Header JP1 .....	70
Communication Rate Jumper.....	71
Trace Clock Rate .....	71
The PWR Header -- JP2.....	71
Power Supply to Pod / Target .....	71

## **Chapter 4: Trace Board 73**

Trace Board Introduction .....	73
Trace Board Detailed Installation Instructions .....	73
External Inputs and Controls .....	73
Introduction to Tracing .....	75
Triggers and Hardware breakpoints .....	75
Trace Window .....	76
Pipeline Effects .....	76
Bus Cycle Order .....	76
Bus Width .....	76
Trace Menu.....	76
Find Frame number .....	76
Search Address .....	76
Search Next Address .....	77
Search Previous Address .....	77
Find Trig point.....	77
Save trace as text .....	77
Show misc. data .....	78
Show timestamp.....	78
Benchmarking Using Timestamp .....	78
Relative timestamp .....	78
T = 0 at Cursor.....	79
Convert cycles to time .....	79
Synchronize program window .....	79
Trace setup .....	79
Toggle trace (stop/run) .....	80
Trace Setup Dialog Box .....	80
Board Installed.....	80
Address .....	80
Trace Memory .....	80
Triggers .....	80
Filter Mode: Normal.....	82
Extend Recording.....	83
Filter Mode: Window .....	84
Editing the Trigger Conditions .....	84
Break Emulation? Box.....	85

## **Chapter 5: Pod Boards 87**

Features Common to All Pods .....	87
How It Works .....	87
Stack Pointer .....	87
Indicator Lights .....	87
How to Break Two Emulators Simultaneously .....	88
Trace Input Pins .....	88
Duplicate Resources .....	88
Configuration Requirements .....	89
Internal Addressing or Single Chip Mode.....	89

## **Chapter 6: POD-296-256-SA-50 91**

Introduction .....	91
Dimensions .....	92
POD-296-256-SA Emulation Memory .....	93
Wait States .....	93

Breakpoints .....	94
POD-296-256-SA Headers.....	94
<b>Chapter 7: Accessories</b>	<b>99</b>
Overview .....	99
Surface-mount QFP adapters - SA Family .....	99
Surface-mount SQFP Adapters.....	104
Compilers .....	108
BSO/Tasking.....	108
IAR Systems Software, Inc. ....	108
<b>Chapter 8: Troubleshooting</b>	<b>109</b>
Troubleshooting Overview .....	109
Step 1: Board I/O Addresses .....	109
Step 2: .INI Editor.....	109
Step 3: PWR and XTAL jumpers.....	110
Step 4: I/O On Addresses Pins.....	110
Step 5: Chip Configuration Bytes (CCB's) .....	111
Step 6: The Stack Pointer .....	111
Step 7: Interrupt Vectors .....	111
Step 8: Sample User Program.....	111

## Introducing EMUL296™-PC

### ***Introduction to EMUL296™-PC***

The EMUL296™-PC is a personal computer based in-circuit emulator for Intel's 80C296 16-bit family of microcontrollers. EMUL296™-PC consists of an emulator "plug-in" board, a five foot (1.5 m) cable, various pod boards and an optional trace board. The EMUL296™-PC design supports all Intel microcontrollers that are based on the 80C296SA core.

The EMUL296™-PC software is a *Microsoft Windows 3.x / '95 / NT* application. It follows the *MS Windows* Multiple Document Interface Standard, resulting in the same look and feel as applications produced by *Microsoft* and others for *MS Windows*.

The EMUL296™-PC user interface is consistent with most other *MS Windows* applications and includes dynamically changing menus, moveable and scrollable "child" windows, function key shortcuts for menu items, and context sensitive help. Anyone familiar with *MS Windows* applications will be able to use EMUL296™-PC with little or no other assistance. It also supports the *MS Windows* Dynamic Data Exchange protocol and can export data written to RAM to other *MS Windows* applications.

The EMUL296™-PC hardware is modular. The software user interface implements an effective high level debugger. It has support for local variables, C typedefs, and C structures. The Trace board options add bus cycle tracing, triggering and filtering functions.

### ***How to use this manual***

This manual was written with different kinds of users in mind. All users should have *MS Windows* installed and have learned the skills taught in the Basic Skills chapter of the *Microsoft Windows User's Guide*. It also assumes a basic familiarity with the chip you are using. Many of the EMUL296™-PC features are designed around the features of the supported chips. Being familiar with the chip is a prerequisite to understanding how to use the emulator productively.

#### *If you are new to emulators of any kind*

read the manual completely, including the reference chapters. You may skip the sections that describe pods items you do not have.

#### *If you have used emulators with other microprocessors*

and understand the difference between a hardware breakpoint and a software breakpoint, but are not familiar with the chip being emulated, you are strongly encouraged to review the features of the chip you have, then thoroughly read the section that describes the applicable pod before running the emulator.



*If you are familiar with emulators, MS Windows, and the chip,*

read the Emulator Board and Software User Interface Chapters, skim the section that describes the pod you are using, and then begin using EMUL296™-PC, referring to on-line help, when needed. After a few days of use, skimming the Reference chapters may highlight useful features.

## **Manual Conventions**

Type the words in double quotes exactly as shown (without the quotations) except for the <Enter>, <Ctrl>, <Tab>, and <Alt> keys. Use the <Alt> and <Ctrl> keys like shift keys. Hold them down while you press the key that follows them in the text. For example, if the text instructs you to type <Alt>F, press down and hold down the <Alt> key then press the F key.

Window names and labels that appear on the screen are printed in **bold** to set them apart from the rest of the text.

Notes and hints are printed in italics, and warnings have a box around them to set them apart from the rest of the manual text. Pay careful attention to them.

## **Quick Installation Instructions**

### System Requirements

EMUL296™-PC requires a personal computer with at least one free ISA (or EISA) bus slot. The PC must also have at least 2 megabytes of RAM, a CPU that is either 80386, 80486, or Pentium compatible, a hard disk with at least 3 megabytes of unused space and *Microsoft Windows 3.1* (or higher), *Windows '95*, *Windows NT* or *OS/2 2.1* (or higher) installed. A mouse is not required, but is strongly recommended.

### Quick Setup Instructions

The hardware and software are designed to be easily installed and quickly running on most personal computer systems. Users can normally begin using their emulator (without yet connecting to the target) after following these initial steps. However, if you are new to personal computers, if you are unsure about what to do after reading the quick installation instructions, or if your emulator does not work after you follow these instructions, follow the steps for installing and configuring each board and the software as outlined in their respective chapters.

## **Installing the Emulator**

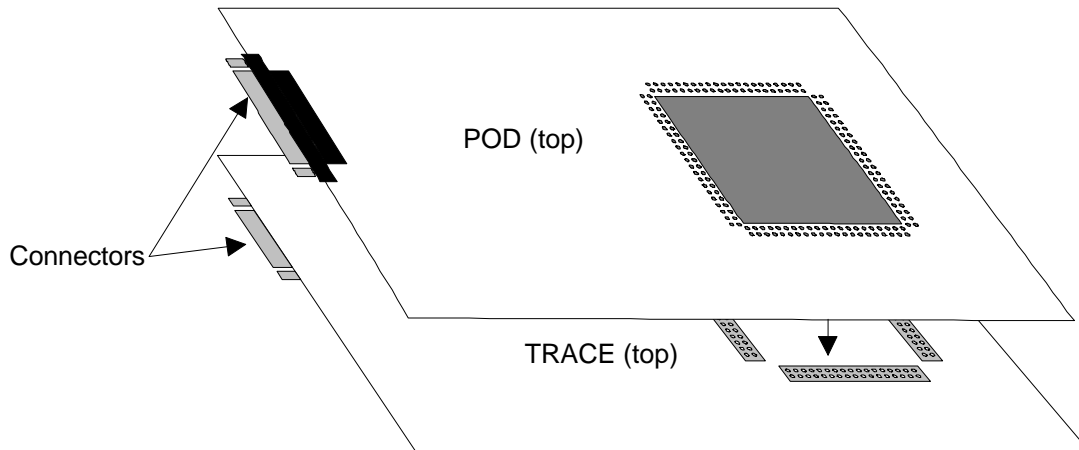
Installing the emulator board is much like installing most other AT-style boards:

1. Turn off the power.
2. Remove the PC cover.
3. Remove the slot cover (if present) for an available 8 bit slot.

4. Insert the emulator board into the slot and use a screw to secure the emulator.
5. You can now close the cover, and attach the cable to the emulator and the pod.

### ***Installing the Trace Board (If Used)***

1. Turn off the power.
2. When you assemble the pod and the trace, make sure that you have the boards aligned as shown below:



**Figure 1. Assembling Pod and Trace Boards**

*Note: Please be aware that you'll have to apply a fair amount of even force on the two boards to make them snap together. Make sure that all connectors and pins are aligned so they will fit together properly.*

1. Finally start the EMUL296™-PC program.

---

*Note: It is not intended that you, as a customer, should disassemble the trace from the pod board once they have been assembled. ICE Technology is not responsible for personal injury caused by trying to separate the EMUL296-PC pod board from the EMUL296-PC trace board.*

---

### ***Installing the Pod***

With the PC power off, line up the cable connector with the slot on the emulator board, and insert. There is no lock, but friction will secure the cable adequately. On the other end of the cable, insert the cable connector firmly, and tighten the screws. Remove any antistatic foam from pins. Before attaching the pod to your target, it is a good idea to power up the PC, install the software, and follow the procedures described in the section titled "Confidence Test" on page 13.

## ***Installing the Software***

To install this software, run SETUP.EXE by typing "WIN A:SETUP" at the DOS prompt or, from within *MS Windows*, by selecting the RUN item in the Program Manager File menu and typing "A:SETUP" as the file to run. A dialog box will ask for a directory for the EMUL296™-PC software. You will be asked if you have a Win NT PC. Either accept the suggested directory or type a different one. SETUP will uncompress and copy the files from the floppy to the hard disk directory specified and change the paths in the ".ini" file. When installed, there will be a **EMUL296** program group containing the EMUL296 icon. Double-clicking on this icon will start the EMUL296™-PC application.

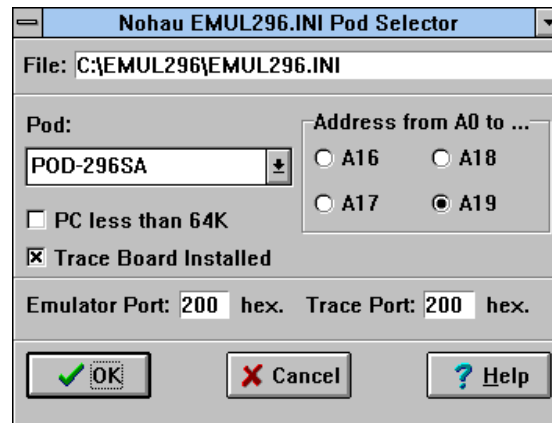
The program group will also contain icons for several .wri files. These files contain important information about what has been fixed in the latest revisions of the software and problems that we know about that have not been fixed yet. Please take the time to read these files.

## ***Initial Software Configuration***

The *Windows* software is used for all EMUL296-PC products. The type of target processor in the software configuration must agree with the type of pod you are using. If not, you may see an error message. To ensure that you do not get this error, we include a utility that you should run when you first install the emulator, and possibly again when you change your pod type. This utility is called **INI296**. (You can also run this utility any time you want to check the values in the initialization file.)

To invoke **INI296**, double click on the icon in the **EMUL296** program group labeled **INI296**. If the EMUL296.INI settings are not self-consistent, you will see a warning message, otherwise you will see the window shown in Figure 2.

To correctly configure your software to match your hardware, start by selecting the pod that matches your pod type. If your pod cannot address memory above 64K, put a check mark in the PC<64K box. If the pod you are using can address memory above 64K, either select PC<64K (because you are not using the extra addressing) or leave the box unchecked and make sure that the jumpers TRA16 through TRA19 match the field labeled Address from A0 to ... For example: if TRA16 is in the EA16 position but the reset is in the GND position, click on the button labeled A16. If all the TRA headers are in the EAxx position, click on the button labeled A19.



**Figure 2: Choosing a target processor with INI296**

Make sure that the **Trace Port:** field is the same as the **Emulator Port:** field, click on the **OK** button or press <Enter>. The **Emulator Port:** field must agree with the values you set in the J2 jumper on the emulator board, as described in the section, "Setting the I/O address jumpers – J1" on page 70.

After you have set up the initial processor type and I/O addresses, you can start the emulator application. You are done with installation.

### **Confidence Test**

Before starting the emulator software and before connecting the pod to your target, run the confidence test installed along with the emulator program. An icon labeled "Confidence Test" will be in the Nohau group. Double-click on it to start the Confidence Test.

The confidence test will read the Pod name and I/O address from the EMUL296.ini file. If these values are incorrect, you must run ini296.exe. It will take less than a minute to complete the tests. Many of the tests repeat with slight variations. If any tests report unexpected errors, call or email support@icetech.com customer support for assistance.

### **Quick Start Instructions**

This section describes how to quickly start using EMUL296™-PC to debug an existing program or target board once the EMUL296™-PC hardware is installed and the user interface software is running.

To load and execute a program:

1. Select **Load code ..** from the **File** menu and identify the "absolute" file to load by using the dialog box
2. Click the **Reset** button.

---

*Note: If you do not have code to load, do the following to enter your own small user program:*

---

click in the **Program** Window

type <Ctrl>-A

type in address 2080 (FF2080 if PC > 64K)

type <Ctrl>-N to force the program counter to address 2080

hit <Enter>

type:       NOP <Enter>

          NOP <Enter>

          LJMP 2080 <Enter>

then continue with item 3 below.

3. Click on the **GO** button in the tool bar

To set a software breakpoint:

1. Click twice on the desired instruction in any **Program** window, or
  2. Click once on the address in any **Program** window, or
  3. Click once on the line number for the desired instruction in any **Source** window.
- To make a software breakpoint inactive either:

1. Click on the desired breakpoint in any **Program** or **Source** window, then press F2, or
2. Select Setup .. from the **Breakpoints** menu, click on the breakpoint, click on the **Toggle** button, or
3. Highlight (click once on) the breakpoint and select **Toggle Breakpoint** from the **Program** or **Source** menu, or
4. Highlight (click once on) the breakpoint and select Toggle from the **Breakpoints** menu.

To delete a breakpoint either:

1. Select **Setup ..** from the **Breakpoints** menu, click on the breakpoint, click on the **Delete** button, or
2. Select **Delete All** from the **Breakpoints** menu.

To use the in-line assembler to change the program loaded:

1. Scroll a **Program** window until it shows the address to be changed or hit <Ctrl>A and type in the desired address
2. Highlight the instruction to be changed with the cursor or arrow keys
3. Type the desired mnemonic (this will open an **Enter new instruction:** dialog box) and hit <Enter>.

To change a RAM value:

1. Scroll a **Data** window until it shows the address to be changed
2. Highlight the address to be changed with the cursor or arrow keys
3. Type the desired value (this will open an **Enter data** dialog box) and hit <Enter>.

**Warning:** *Always turn on the PC before applying power to the target. Always turn off the target power before turning off the PC power.*



## Chapter 1: Software User Interface

### *Detailed Software Installation Instructions*

Before installing the software, it is important to have a basic understanding of how to operate *MS Windows*. For help mastering *MS Windows*, please refer to the *Microsoft Windows User's Guide*.

The EMUL296™-PC floppy disk includes an *MS Windows* compatible SETUP.EXE program. To install this software, run SETUP.EXE by typing "WIN A:SETUP" at the DOS prompt before entering *Windows* or, from within *MS Windows*, by selecting the **RUN** item in the **Program Manager File** menu and typing "A:SETUP" as the file to run. A dialog box will ask for a directory for the EMUL296™-PC software. You will be asked if you have a Win NT PC. Either accept the suggested directory or type a different one. SETUP will copy files from the floppy to the hard disk directory specified and change the various *MS Windows* ".ini" files as needed. When installed, there will be a **EMUL296** program group with several icons such as an **EMUL296** icon and an icon for the confidence test, **CONF.EXE**. Double-clicking on the emulator icon will start the EMUL296™-PC application. If you wish to move the icon to another group, you may do so by using the **Move...** menu item in the **Program Manager's File** menu or by dragging the icon to the new group.

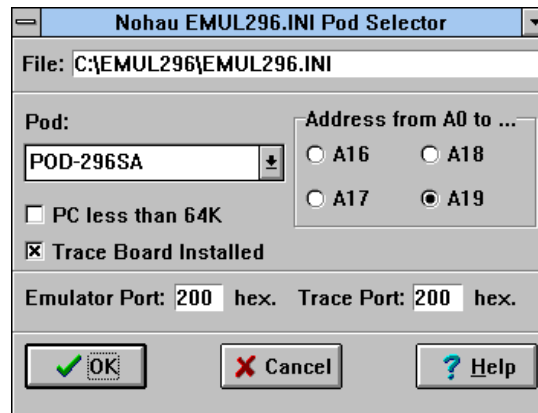
### *Initial Software Configuration*

The Windows software is used for all EMUL296™-PC products. The type of target processor in the software configuration must agree with the type of pod you are using. If not, you may see an error message. To ensure that you do not get this error, we include a utility that you probably want to run when you first install the emulator, and possibly again when you change your pod type. This utility is called INI296. (You can also run this utility any time you want to check the values in the initialization file.)

To invoke INI296, double click on the icon in the **EMUL296** program group labeled **INI296**. If the EMUL296.INI settings are not self-consistent, you will see a warning message, otherwise you will see the window shown in Figure 3.

To correctly configure your software to match your hardware, start by selecting the pod that matches your pod type. If your pod cannot address memory above 64K, put a check mark in the PC < 64K box. If the pod you are using can address memory above 64K, either select PC < 64K (because you are not using the extra addressing) or leave the box unchecked and make sure that the jumpers TRA16 through TRA19 match the field labeled Address from A0 to ... For example: if TRA16 is in the EA16 position but the reset are in the GND position, click on the button labeled A16. If all the TRA headers are in the EA position, click on the button labeled A19.





**Figure 3: Choosing a target processor with INI296**

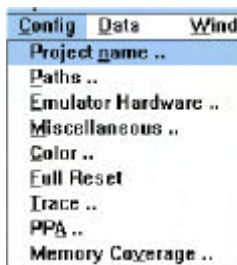
Make sure that the **Emulator Port** address and **Trace Port** address are the same. After you have set the **Emulator Port**: and **Trace Port**: fields, click on the **OK** button or press <Enter>. The **Emulator Port**: field must agree with the values you set in the J1 (address) jumper on the low cost emulator board, as described in the section , "Setting the I/O address jumpers – J1" in the Emulator chapter.

After you have set up the initial processor type and I/O addresses, you can start the emulator application. You are done with installation.

## Configuring the Software

If the Quick Installation instructions do not work, you will most likely need to adjust either the hardware jumpers, the software configuration, or possibly both. Please refer to the appropriate chapters for setting the jumpers on the Emulator board or the Pod board. The next few pages describe all of the items in the Config menu. Use these menu items to examine the software configuration in detail and to change it if needed.

## Projects



A project is a collection of software configuration settings that are all associated with a specific person, target, or software development project. This menu item opens a dialog box that allows you to set up named configurations or projects. This is first in the menu and described first because all of the other Config menu item settings will be stored as settings for the current project in a file with a ".PRO" suffix. There is an ".INI" file and those settings are used if there is no current project. But if the ".INI" file contains the name of the current project, all software settings are taken from ".PRO" file for that project.

Projects behave differently than, for instance, a word processing document. All software configuration settings are written to disk every time you change projects or whenever you exit the emulator software. There is no "exit without saving changes" option. Once you

make a change to the configuration, it is immediately effective and will, unless you manually undo the change, be saved to the disk in the project file.

## Creating a Project

Users who change the software settings and THEN change the name of the project may believe the old project will remain unchanged. In fact, the moment a new project is created, the current settings will be saved to the old project, not the new project. The new project will be saved when exiting the debugger or when changing projects (again).

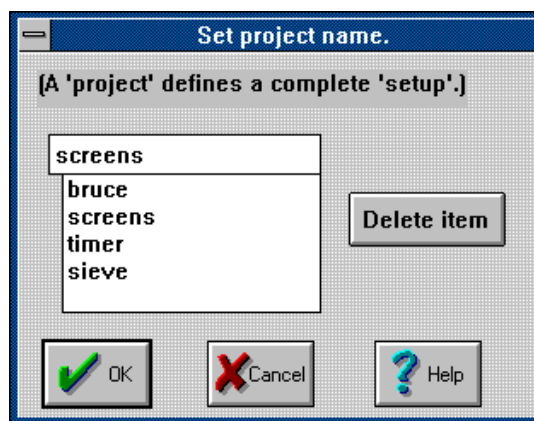


Figure 4: Set Project Name Dialog Box

To add a project, type the new name over the current name. Because the project name is used as the body of a DOS file name, do not use characters in the name that cannot be used in a file name (like a space character). The new project will inherit all the settings from the old project. Projects are deleted by highlighting the project name you want deleted and then clicking on the Delete item button.

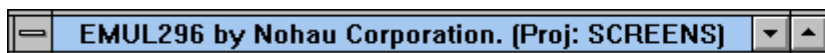


Figure 5. EMUL296™-PC Title Bar

The name of the current project appears in the emulator software title bar. Figure 5 is an example of the EMUL296 title bar for the project named SCREENS (used to create the screen shots for this manual).

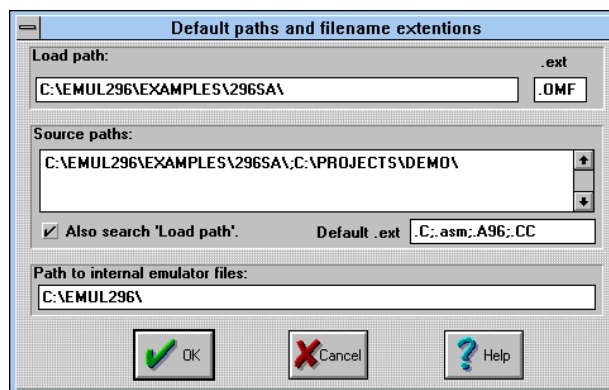
## Setting the Paths ..

The next item in the **Config** menu is **Paths** .. which opens the dialog box shown in Figure 6 of the manual. The emulator uses these directories to find the files it needs.

Each of these fields can hold up to 1024 characters. Each directory in the path must be separated by a semi-colon (;) just like MS DOS path names. By default, The **User load**

**modules:** field will contain the directory from the last loaded object file, and the Emulator internal files: field will contain the directory where the emulator files were installed.

The **Load path:** directory is the default directory searched for Intel Hex files and absolute object files. Any directory can be specified when loading a module, but the directory shown here is the default. The **.ext** field specifies the default file extension. Files in a directory with this extension will be shown in the **Source** Window.



**Figure 6: Paths Dialog Box**

With many compilers, the full path name of the source file is contained within the object file. Linked object files consisting of several linked objects will, correspondingly, have several source file names and paths. If that source file name exists in the object file that EMUL296™-PC is loading, the debugger will look for that source file when updating the Source window.

The second field, **Source paths:** identifies other directories to search for missing source files not identified in the object file or files moved since the compile. The directories in this field must be entered by the user. Once entered, directories will stay here until removed by the user. The small check box, when checked, will tell EMUL296™-PC to look for source files in the Load path: directory as well. Simple projects may have all the source and object files in the same directory (the **Load path:**) and may not need any directories in the Source paths: field.

---

*Note: The ".ext" field specifies the source file extension. If your C modules have the extension ".c", enter that. To see assembler source (.asm) in the **Source** window, enter ".asm" .*

---

The Emulator internal files: field will be set during the installation and probably will not need to be changed. Emulator internal files: is the directory the application uses to find the various support files that are part of the EMUL296™-PC software such as register definition files and dynamically loaded libraries. Normally, the installation program will set this to the proper directory. If you copy or move EMUL296™-PC to a new directory or disk drive, remember to change this field also.

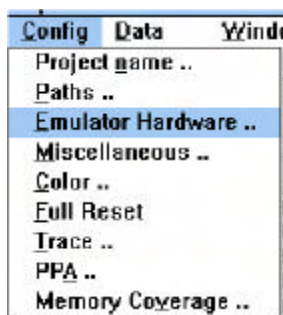
## Mapping memory

ROM and RAM on the target can be emulated by RAM on the pod board. This RAM is called emulation RAM. The entire address range for both ROM and RAM can be mapped to either the target or the emulator in blocks as small as the chip selects specifies. This is controlled by jumpers and the chip selects on the CPU.

When an address is mapped to emulation RAM on the pod, all READ, WRITE, and instruction fetch cycles to that address are directed to emulation RAM. Target RAM, target ROM, and memory mapped devices on the target at that address are ignored. If your target has a memory mapped I/O device within a block mapped to emulation RAM, this mapping will prevent your application from accessing that device. To avoid this, map the blocks that contain target devices to the target.

*Note: On reset, the emulator software will write to the CS0 registers so that CS0 will cover the entire 16 Mb memory range. This will map all memory to the pod when a jumper shunt is on JP24 (CS0) and ensure that the code can be loaded to the pod.*

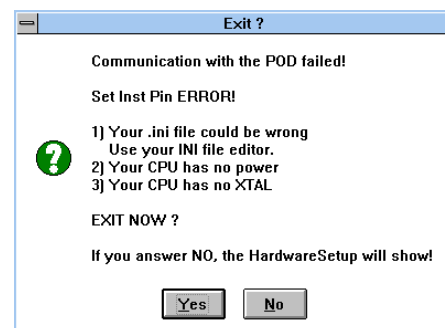
## Emulator Hardware Configuration



**Warning:** *The settings in this dialog box must agree with the emulator jumper settings, the pod processor type, and the application startup code. If this is not the case, EMUL296™-PC will not work properly.*

The **Emulator Hardware ..** menu item configures the software to correctly communicate with the hardware.

The **Emulator Port** value must agree with the jumper settings on the low cost emulator board header J1. (See "Setting the I/O address jumpers – J1" in the Emulator chapter). If they do not agree, the EMUL296™-PC software will not be able to communicate with the hardware, and the dialog box in Figure 7 will automatically be displayed, as a reminder that communication has failed and some change is needed.



**Figure 7. Failed Communication Dialog Box**

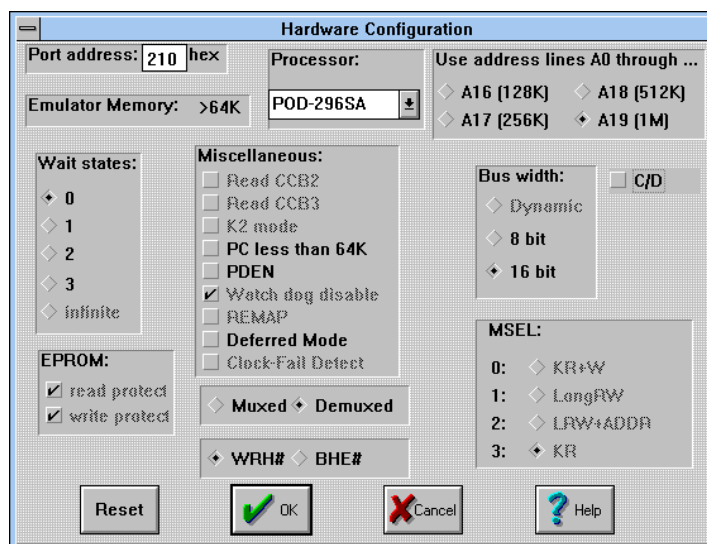


Figure 8. Hardware Configuration Dialog Box

### Setting the Chip Config Registers

Most of the **Hardware Configuration** dialog box controls how EMUL296™-PC manipulates the Chip Select registers. allowing the controller to read them after each reset. Normally, EMUL296™-PC must update the Chip Config registers before each reset. If it did not, the emulator might not be able to write to emulation RAM. It might be unable to load code, or toggle breakpoints, etc.

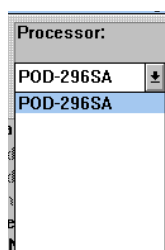


Figure 9. Processor List Box

The processor list box lets you choose the kind of processor being emulated. This setting must agree with the kind of processor you are using in the target or the emulator will not work correctly. If you must change the processor type, select the correct processor from the list, ignore any errors, ignore any warnings that appear at this point, exit the emulator **Hardware Configuration** dialog box, and select **Full Reset** from the **Config** menu. Choose the **Full Reset** menu item before you do anything else, or push **Reset** in the dialog box and then click **OK**.



Figure 10. Wait States field

The box labeled **Wait States:** will let you select the number of wait states (for CS0) the chip will use when accessing external memory. Emulation RAM is fast enough to respond with 0 wait states. Target RAM may or may not be fast enough, depending upon the speed of the chips on your target. Selecting the infinite button requires that the target hardware correctly assert the READY signal when the data on the bus is valid.



Figure 11. Setting the Bus Width

In the **Bus width:** box, selecting either 8 bit or 16 bit bus width forces the controller to use only the specified bus width for CS0.

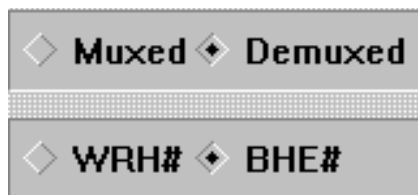


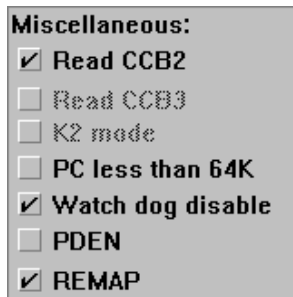
Figure 12. Bus Control Settings

The P5.5 pin carries the BHE/WRH signal. Selecting one of these two radio buttons controls which signal the controller sends on that pin. BHE stands for Bus High Enable and will indicate that there is valid data on the high byte of the bus (D8 through D15) in 16 bit mode for either WRITE or READ bus cycles. WRH stands for WRite High and is used as a Write Strobe to the devices connected to the high byte of the data bus during WRITE cycles.

### Miscellaneous bits

As of this writing, the Miscellaneous bits feature is not functional.

The Read CCB2 and Read CCB3 check boxes, if checked, will force the controller to read CCB2 and CCB3. If they are not checked, they will prevent the controller from reading their respective control bytes. If you have selected a controller from the Processor List box that uses only 2 CCB locations, (CCB0 and CCB1) these check boxes will be gray.



**Figure 13. Miscellaneous CCB bits**

The 8xC296 controllers have more than 16 address bits. With a check mark, this field prevents the controller from addressing anything above 64k. The Use address bits A0 through .. field (in the upper right corner of the dialog box) will be inactive. Address lines A16 through A19 (port E bits) may then be used for I/O. When you remove the check mark, the address bits above 15 become configurable as address bits and the address line configuration field becomes active.

The PDEN check box, when checked, allows the controller to go into Power Down Mode if the IDLPD #2 instruction is executed.

Once all the check boxes and radio buttons are set the way you want them, click on the **OK** button. This will:

1. Update the CCB RAM locations.
2. RESET the controller.
3. Exit from the dialog box.



**Figure 14. Exiting the Hardware Config dialog box**

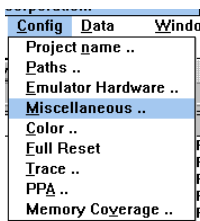
To exit the dialog box without writing any of the changes just made, click on **Cancel** button. To write the changes and reset the controller without exiting the dialog box, click on the **Reset** button.

---

*Note: When the EMUL296™-PC software is started, the controller will be released from a reset state and it will read the CCB values from locations in the POD EPROM and execute code that writes the .ini file CCB values to the CCB RAM locations. Then the controller is reset again and this time it will read the CCB values from RAM.*

---

## Miscellaneous Configuration



The **Miscellaneous** item in the **Config** menu opens a dialog box that controls special features of EMUL296™-PC:

- (1) when and if automatic resets occur.
- (2) optional reset vector values.
- (3) the source code address range for limiting where breakpoints are set.
- (4) the memory scroll range used for **Data** and **Program** window scroll bars.
- (5) writing values to memory while the application is running.

By default, the emulator resets the controller when the EMUL296™-PC software is started and after an object file is loaded. The **Reset chip at start up:** and the **Reset chip after load file:** radio buttons can disable either of those resets which may be helpful during particularly difficult or unusual debugging circumstances.



**Figure 15. Miscellaneous Setup Dialog Box**

For example, if you have a code file you want to load but you are unsure if the CCB values in that code file are correct, you may not want the emulator to reset the controller right after loading the file. Instead, you may want to load the file manually and then check the CCB values before they are used.

Saved watchpoints are not implemented yet. For information about the status of this feature, call or email customer support a [support@icetech.com](mailto:support@icetech.com).



**Figure 16. One Feature Not Yet Implemented**

The next field in the **Miscellaneous** menu dialog box, the **DDE sampling interval**, controls how often Shadow RAM is updated on the screen and how often a DDE link is updated.

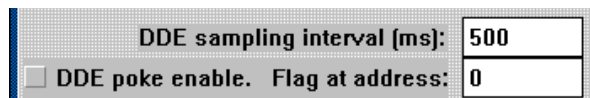


There is a lower limit to how often the screen and the DDE link can be updated. This limit depends upon the speed of your machine, how much RAM your machine has, and how many applications are running. Due to delays from inter-application messaging in *MS Windows* and possible problems caused by a message backlog, the lowest setting allowed is 100 milliseconds. The upper limit is 32767 milliseconds.

---

*Note: DDE is not supported as of this printing. Please check your current version of software.*

---



The image shows a graphical user interface for DDE control. It consists of two rows. The first row has the label "DDE sampling interval (ms):" followed by a text input field containing the value "500". The second row has a checkbox labeled "DDE poke enable." followed by the text "Flag at address:" and a text input field containing the value "0".

**Figure 17. Controlling the DDE Sample Interval**

Occasionally, while the emulation is running, writing a value to a RAM location can help debugging. For example, it might be useful to simulate a memory mapped input device this way. The EMUL296™-PC DDE interface supports "poking" a value into an address. The DDE link can be a two-way connection and some applications may send a value to a particular address in emulation or target RAM. However, updating RAM at pseudo-random times can be dangerous for the program and possibly the hardware you are controlling while you are emulating. With the next field, the DDE poke flag address, you specify an address of a two byte flag that, when polled and found to be set to hexadecimal 1234, indicates that it is safe to write the DDE value to RAM.

The check box to the left turns poking on or off: a check turns it on. When EMUL296™-PC has a value to "poke" into RAM (every DDE sampling interval), it first polls this box. If it is checked, the emulator application will read the contents of the poke flag address. If the contents are set to 1234H, then EMUL296™-PC will suspend the emulation for approximately 200-250 microseconds while it updates RAM. Finally, EMUL296™-PC will clear the poke flag (set to 0) and restart the application. The software must reset the flag to 1234 when it is again safe to update the poke addresses.

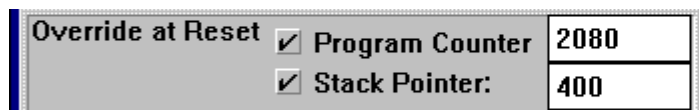
---

*Note: Displaying Shadow RAM and sending a value to another application does not slow the emulation.*

---

### **Enable Code Space Limits**

In some symbols files, symbols are not identified as Program space variables or as Data space variables. If this is true of your file, you may see the EMUL296™-PC software try to set breakpoints in your variable address range. To prevent this, check the **Enable code space limits** box and set the Low and High addresses to encompass just the instructions. Configured this way, EMUL296™-PC will not put breakpoints outside that address range.



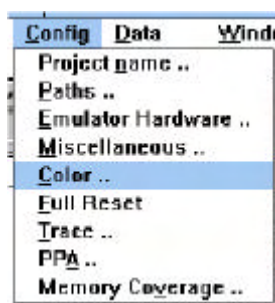
**Figure 18. Forcing Reset Vectors**

When the **Override at Reset** boxes are checked and the fields contain addresses (in hexadecimal notation), those values will be written to the controller's program counter and stack pointer every time EMUL296™-PC resets the controller. If you have some test code at an address other than 2080H that you want to execute, filling in this field will force the program counter to the specified value each time you reset the controller. Similarly, to run the test code right after a reset, the stack pointer register must have a legitimate value. This field will conveniently force the stack pointer to a specified value after reset without having to run your start-up code.

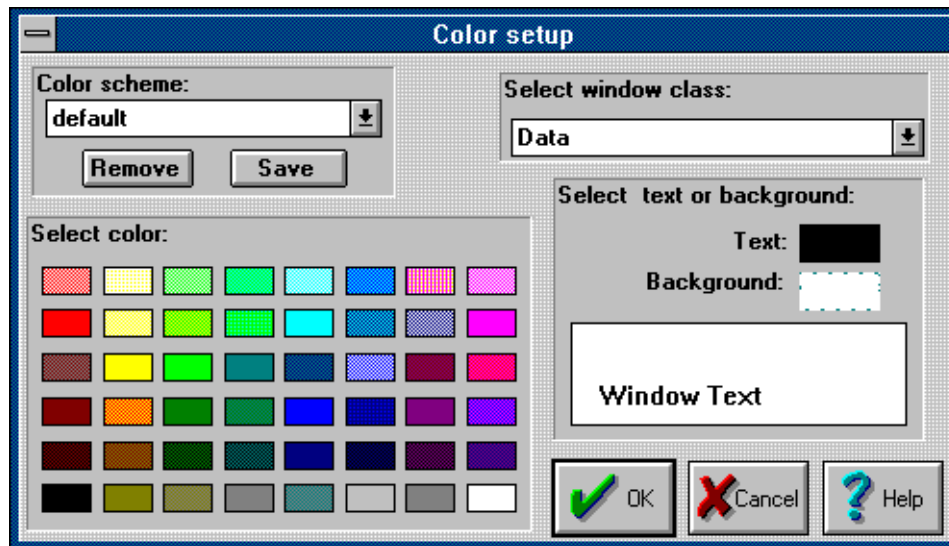
**Warning:** *Using the Stack Pointer field will set the stack pointer for you which will not happen on a stand-alone target. If you use this field, your start-up code must also set the stack pointer every time the controller is reset or your application will behave differently.*

In the lower left corner of the **Miscellaneous Setup** dialog box is a group of radio buttons that control the Memory scroll range. These buttons specify the highest address displayed in **Program** and **Data** windows when the elevator in the scroll bar is dragged to the bottom of the scroll bar. If the scroll range is set to 256K, the bottom of the scroll bar represents the address 40000 Hex. Similarly, halfway down the scroll bar represents 20000 Hex. If the scroll range is set to 1 Meg. the bottom represents 100000 Hex, and so on. .

## Window Colors



Under the **Config** menu is the **Color ..** menu item. Open this dialog box to set the colors of different kinds of EMUL296™-PC child windows. For example, all **Program** windows can be set to have a dark blue background with white text to differentiate them from other kinds of windows. At the same time, all **Data** windows can be green with Black text, and all **Source** windows set to have white background and red text. It is possible to make the screen quite attractive.



**Figure 19. The Color Setup Dialog Box**

For each window class that you wish to change, select the window class from the **Select window class** drop list. While that class name is showing in that field, the colors you select will be assigned to that class of windows.

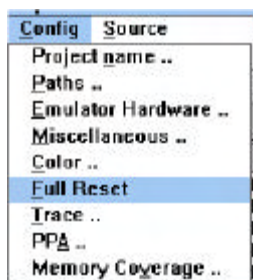
After you have set all the colors the way you want them, you can name a color scheme by typing the name in the **Color scheme** field and then click on the **Save** button. This color scheme can then be recalled by selecting it from the drop list of color schemes.

---

*Note: Not all combinations of background and foreground colors are possible. EMUL296™-PC is constrained by the same limits as MS Windows itself, and is affected by the color palette chosen in the Windows Control Panel program. No matter what colors you select from this palette, the example text pane will show you the colors that will actually be used by EMUL296™-PC.*

---

### **Reset vs. Full Reset**



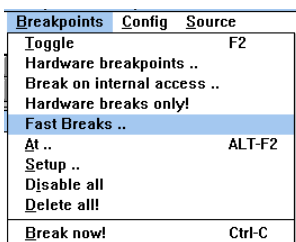
Under most circumstances that you will encounter, a Full Reset is the same as clicking on the **Reset** button in the speed bar, selecting **Reset Chip** from the **Run** menu, or pressing **<Ctrl>F2**. With both kinds of reset, the controller is reset by pulling the reset line low. The controller reads the CCB registers and the controller is immediately halted. When the emulator software is first started, or possibly after an accident on the pod or target, the states of the two large logic chips on the pod are not known. In a Full Reset, before the controller is reset, the large logic chips on the pod are reloaded with their configuration information (you can find the **Full Reset** menu item is under the **Config** menu). Under all circumstances you

MAY use the **Full Reset** menu item in place of clicking on the **Reset** button. Unless you are changing the pod type, there will be no NEED to use the **Full Reset** menu item.

### Trace Config Menu

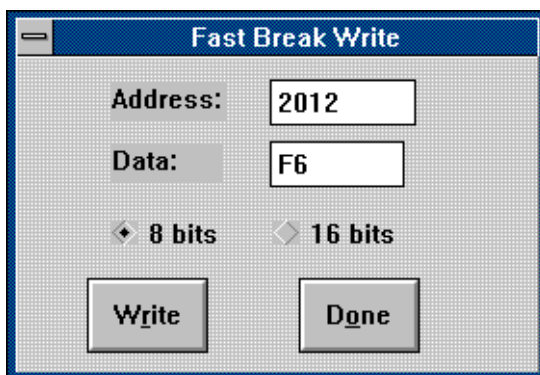
For information about the Trace .. menu item, please refer to the manual in **Chapter 4: Trace Board**.

### Fast Break Write



If the emulator is connected to a motor controller, breaking execution may be hazardous to the motor, or other parts of the target. At the same time, you may want, or even need, to update a memory location to run a certain test. The feature designed for these circumstances is the **Fast Break Write**.

A **Fast Break Write** breaks execution, updates one address with one 8 or 16 bit value, then resumes execution, all within 15 microseconds or less (at 16 MHz).



**Figure 20. Fast Break Write Dialog Box**

The address and data fields are both in hexadecimal notation. Execution will be paused and the address will be updated when (every time) you click on the **Write** button.

---

*Hint: To read data values without delaying the application, use a Trace board and make sure that your software regularly reads or writes to the address you wish to monitor. You can then set up a trace filter to only record the bus cycles to and/or from the desired address(es). Stopping and viewing the trace buffer does not affect the running application.*

---

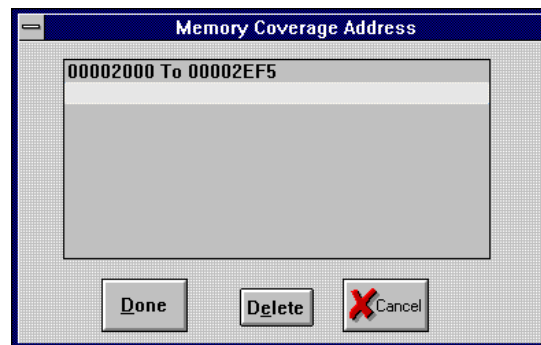
## Memory Coverage



The EMUL296™-PC trace option includes the hardware necessary to monitor memory and correlate its use with your C source code. If instructions are fetched, the trace board will mark those addresses.

Choosing **Memory Coverage** from the **Config** menu will open a **Coverage** window and put the trace board into a "Coverage Mode" that prevents normal tracing. As long as this coverage window is open, the Trace window contents will not change. If initially closed, the Trace window will open empty and stay empty until you close the **Memory Coverage** window.

If you load your application software before you open the coverage window, the **Memory Coverage** window will display rows, where each row has a starting address on the left and small black squares to the right of the starting address. The starting and ending addresses are taken from the object file you have last loaded. You may set any address range by selecting the **Edit item** in the **Coverage** menu.



**Figure 21. Editing the Coverage Address Range**

You may either edit the existing range or you may add one or more address ranges. To edit the existing range, double click on the line containing the address range. To add an address range, double click on the empty line just below the existing address range.

EMUL296™-PC supports any number of address ranges. The ranges do not need to be next to each other. They may be located anywhere in the address space. The only practical limit is that the sum of all ranges must be less than 256 kilobytes.

Once you have edited the address ranges (changed them from the default range set up when loading the file) you will want to save these settings for future use. The **Save** menu item in the **Coverage** menu will write the current address range(s) to the .INI file. The **Load** menu item will read them from the .INI file.

Each square represents a memory word: two bytes starting on an even byte. Squares are grouped into segments 8 squares across. For each address there are 4 rows of segments. "Figure 22. Typical Memory Coverage Window" shows a **Memory Coverage** window with 7 segments in each row. In this case, each row of dots represents 70 hex bytes of memory. Each row of blocks represents 1C0 hex bytes. As the window gets wider, each row contains more blocks of squares and the addresses will get farther apart.

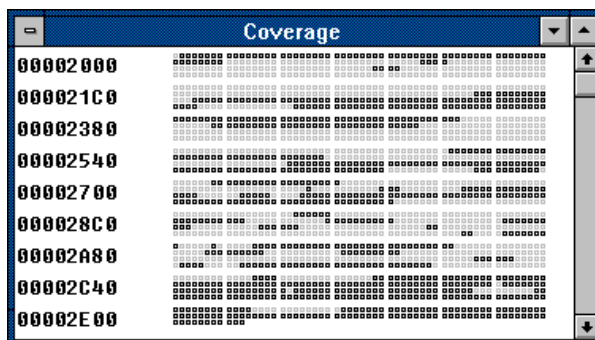


Figure 22. Typical Memory Coverage Window

A square (or memory word) that has been covered will be blue. If either byte of the word has been fetched, the square will change color. Untouched squares will be black. This gives you a visual estimate of how much of your code has been executed since the **Coverage** window was last reset. For an exact representation, look at the **Program** and **Source** windows.

As shown in Figure 23, there are new symbols in the **Source** and **Program** windows. In the **Program** window, you will find either a : or an x between the address and the rest of hexadecimal value at that address. The : means that the opcode has not been executed. The x indicates that it was either executed or prefetched.

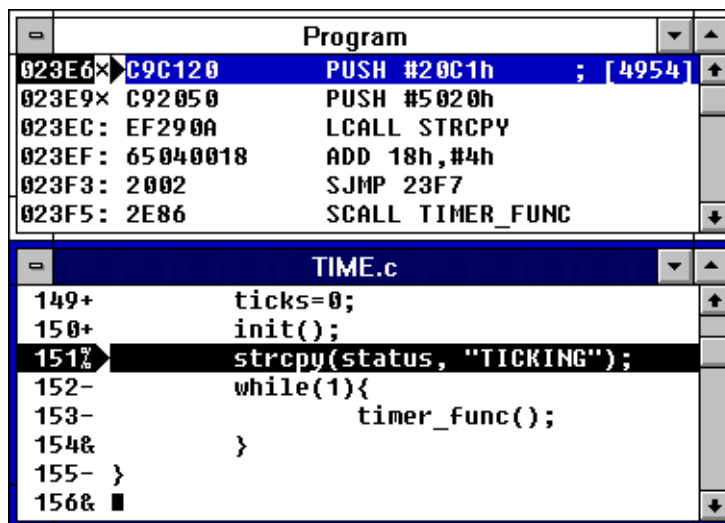


Figure 23. Program and Source Windows in Coverage Mode

In the **Source** window, there are 4 new indicators between the line number and the line text: + - & % , explained below:

- + All opcodes from this line were fetched.
- % Some opcodes from this line were fetched.
- No opcodes from this line were executed.

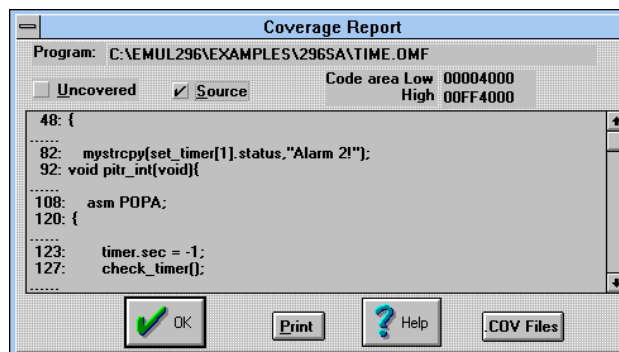
& This line generated no executable code.

Please note two important things in Figure 23. The strcpy() call is not completely covered, so in the Source window, it is marked with a percent sign. The other important thing to notice is that address 23E6 has a breakpoint. The instruction at address 23E9 has been marked as fetched (because it has been fetched) but it has NOT been executed. Instructions right after executed jumps will be shown as fetched. They may or may not have been executed.

### Summary Memory Coverage Report

A screen full of coverage information may be helpful, but it won't satisfy the FDA. or the FAA. They both want written evidence they can hold in their hands that show that your tests actually tested your code. EMUL296™-PC can help there.

When you select **Report** from the **Coverage** menu, you will see a dialog box similar to Figure 24.



**Figure 24. Summary Coverage Report**

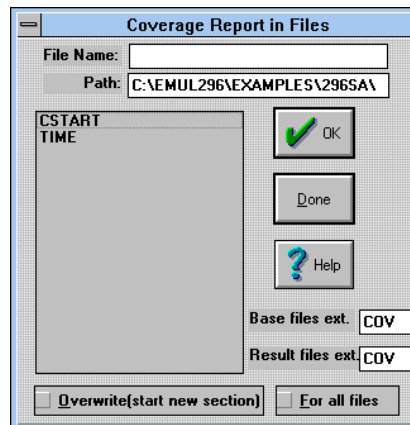
For every module loaded, you see a range of addresses that have been touched or fetched since that last time the coverage memory was reset. You may see the summary applied to source lines (as in the example) or to absolute addresses and opcodes (the default). You may also invert the sense of the summary report. Check the **Uncovered** box and the report will display only the uncovered addresses or lines.

To obtain a paper copy of the report, click on the **Print** button. This will send the summary to the current default printer. A different kind of report, a more detailed report, is available by clicking on the **.COV Files** button. Figure 25 is the dialog box that configures these reports.

The summary report is good for small test runs that can be completed without turning off your P.C.; without exiting the EMUL296™-PC software. However, your tests may be very large; so large that running all of them without exiting the emulator software may not be possible. The detailed coverage reports let you combine multiple test runs in a single document (for each source file).



### Detailed Memory Coverage Report



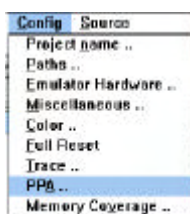
**Figure 25. Producing Detailed Coverage Reports**

The detailed coverage report produces a text file that looks like the **Source** window while in coverage mode. It reads in the source file (or a previous coverage file), and puts the line number and one of the four status characters at the beginning of each line. The output text file is written to the same directory as the source file and, by default, has .COV as a suffix to distinguish it from the .C file. You can change the output file suffix by changing the **Result files ext.** field.

If the source file scanned is actually the output from a previous coverage report, it will combine the two reports so that lines covered by either report will be marked as covered in the new report. This is the feature that allows you to run your tests over several days and still generate a single set of files that accurately reflects how well all of the tests together have covered the generated instructions.

Typically you will want the emulator to create a detailed coverage report for all source files so leave the **File name:** field empty and the **For all files** field checked. You can, of course, generate coverage reports for a single module, in which case you would double click on that module name in the list box. Checking the **Overwrite** box will ignore the coverage data in the input file, if there is any. Then the report files will only reflect the current coverage data. The **Base files ext** field selects the extension of the input text files. If there is no file with the specified extension, the source file for the current module (from the object file) will be used as the input file to generate the report.

### **Performance Analysis**



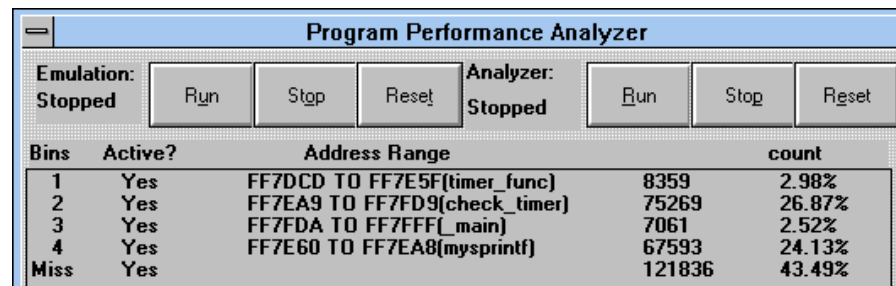
What portion of your application uses most of the CPU cycles? This is the question that Performance Analysis is designed to answer. You set up address ranges or bins, run your program, and then look at the results to see where (or which bin) the statistics say your program spent the most time.

Performance Analysis is a statistical analysis of execution behavior. Once every second, a percentage of the bus cycles are collected,



sorted into their respective bins, and the results are displayed on the screen. As you might guess, the percentage of the cycles that are collected depends upon the speed of your P.C., what other tasks are running, etc.

To get more accurate results, run your program for longer periods of time. If you watch the statistics on the screen, you will see them change quickly at first, then more slowly. When they change very slowly, you know that the statistics will probably not get any more accurate.



Bins	Active?	Address Range	count	
1	Yes	FF7DCD TO FF7E5F(timer_func)	8359	2.98%
2	Yes	FF7EA9 TO FF7FD9(check_timer)	75269	26.87%
3	Yes	FF7FDA TO FF7FFF(_main)	7061	2.52%
4	Yes	FF7E60 TO FF7EA8(mysprintf)	67593	24.13%
Miss	Yes		121836	43.49%

**Figure 26. Performance Analysis Control Window**

When you select **PPA Analyzer** from the **Config** menu, you will see a control window that looks like Figure 26. The application and the data collection will automatically be started every time you open the PPA control window. The six buttons at the top of the window control the application and the data collection separately.

---

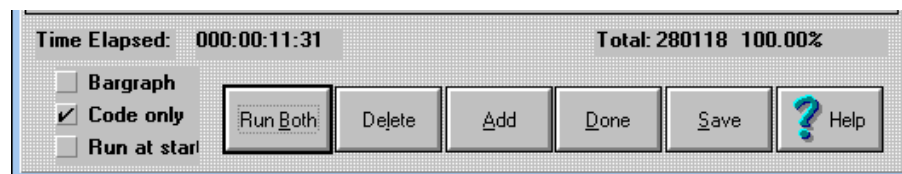
*Note: Clicking on the **SAVE** button saves the setup, not the results.*

---

Each bin is really an address range. If the address range corresponds exactly to the address range of a function, that function name will be displayed next to the address.

A fetch from an address that doesn't fall within any address range will be counted in the **Miss** bin. A fetch from within an existing but inactive address range bin will not be counted at all. It will not count in the inactive range and it will not be counted within the **Miss** bin. Statistics will not be kept for that inactive bin at all. The **Miss** bin cannot be made inactive.

The very first time you configure Performance Analysis you will find only one bin: the **Miss** bin. This bin cannot be deleted, or edited, or be made inactive.



**Figure 27. Performance Analysis Control Options**

Figure 27 shows the rest of the statistics from Figure 26, plus a few options. This data set took 11 minutes and 31 seconds to collect. A total of 280,118 frames were recorded.

Figure 27 shows that data read and write cycles were ignored; the Code only option is checked. Only instruction fetches are counted in the statistics.

Figure 27 also shows that the **Bargraph** option is turned off. If you prefer a graphic display, you may turn on the **Bargraph** option and see the data displayed in a form similar to that shown in Figure 28.

Bins	Active?	Address Range		count
1	Yes	(timer_func)	8359	2.98%
2	Yes	(check_timer)	75269	26.87%
3	Yes	(_main)	7061	2.52%
4	Yes	(mysprintf)	67593	24.13%
Miss	Yes		121836	43.49%

**Figure 28. Bargraph Display Option**

To add bins of your own, click on the **Add** button to open the list of functions shown in Figure 29. You can add any of the functions as an address range, or you may create an address range not on the list.

The **Address Range** dialog box contains the following elements:

- Start:** Text field with value `2F2`.
- End:** Text field with value `3A`.
- Length:** Check box (checked).
- Active?:** Check box (checked).
- Default length:** Text field with value `3A`.
- Function List:** A list box containing: `pitr_int`, `timer_func`, `check_timer`, `main`, `BeginTimerInterrupt`, `sprintf`, `_ofmt`, `conv`, `_numskip`, and `reverse`. The `main` entry is highlighted.
- Add:** Button to add the selected function to the bins.
- Done:** Button to close the dialog.

**Figure 29. Adding a Bin**

To add a bin corresponding to the `main()` function, double click on **main** in the function list, then click on the **Add** button. Note that clicking once on `main` will highlight it but not update the **Start** and **End** fields with the values for `main()`. Also note that double clicking does not actually add the bin. You must click on the **Add** button to actually add the bin. With the **Address Range** dialog box open, you may add as many bins as you like before clicking on the **Done** button to close it.

The **Length** field controls how the **End** field is used. With a check, the **End** field displays the length. Without a check in the **Length** field, the **End** field displays the end address in hexadecimal notation.

Using a very similar screen, any bin can be edited by double clicking on that line in the **PPA Control** window (Figure 26). This is how you activate and deactivate bins.

Once you have collected the data you want, EMUL296™-PC allows you to either save or discard the changes you just made to the list of bins. Only one bin configuration can be

saved, not one per project like most configuration settings. This bin configuration will be automatically restored the next time you use performance analysis.

## Menus

The primary means of controlling the debugger, thus the emulation, is through menus. The EMUL296™-PC menus conform completely with the *Microsoft MDI* standard. Only those menu items that have meaning or can be used with the current selection will highlight when the mouse is pointing to them. Menus are organized to hide items that are out of context.

Most menu items have "Hot Key" equivalents. That is, there is some combination of function keys, character keys, and modifier keys (Control, Shift, or Alt keys) to select most menu items. The Hot Key for each menu item is shown in that menu to the right of the item name, and are also shown below. Where you see "<Alt>FS" as the keyboard shortcut, you should type <Alt>F (hold the Alt key down while you then press the F key) to open the File menu, then press the S key (without the Alt key) to activate the portion of EMUL296™-PC that writes "S" record files. Holding down the Shift key or turning on CapsLock is not necessary. Even though the keyboard shortcuts are all shown in capital letters, the shortcuts are not case sensitive.

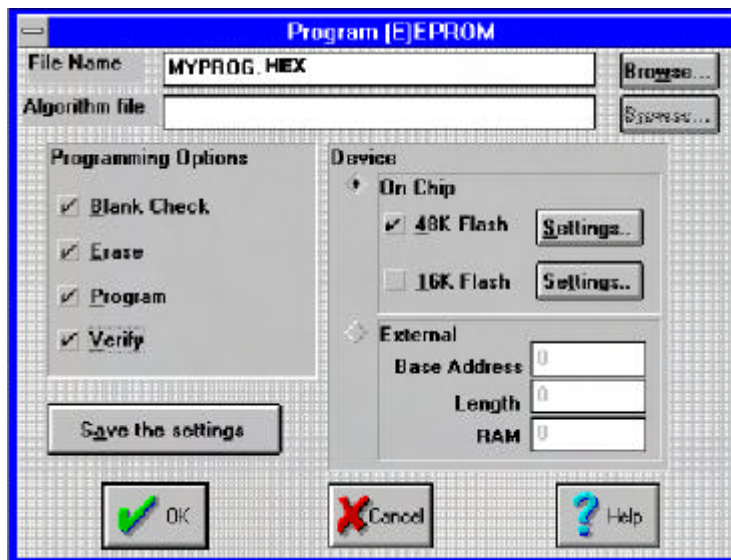
### File Menu

Menu Item	Hot Key	Function
<b>Load code</b>	F3	Load an absolute file. EMUL296™-PC supports many popular compiler object file formats. See the Accessories chapter for more information.
<b>Load default symbols ..</b>	<Alt> FL	Load symbols defined by the MCU manufacturer. Selecting this menu item will load the default symbols defined by the MCU manufacturer in their manuals. This will enhance the display in Program window by converting the addresses of registers into their respective names and bit descriptions. <i>(Note: Loading default symbols may take as long as 40 seconds on some machines.)</i>
<b>Load EEPROM ..</b> (See Note Below)		With some assistance, EMUL296™-PC can load user programs into FLASH memory devices, whether the memory is a module in the MCU or is a chip external to the MCU. Selecting this menu item will open the Program (E)EPROM dialog box shown in Figure 30.

---

*Note: As of this printing, Flash programming is not implemented. Please test it in your software version.*

---



**Figure 30. Programming On-chip Flash Memory.**

From this dialog box, clicking on the **OK** button will start executing the checked **Programming Options** from top to bottom.

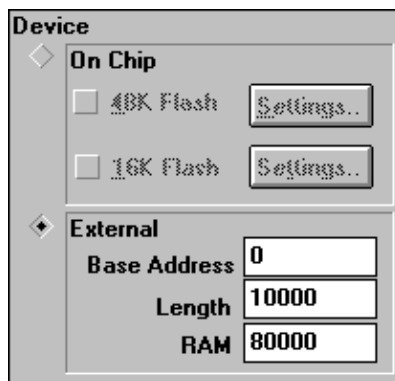
In the example in Figure 30, the file MYPROG.HEX is the file you want loaded into the FLASH memory. At this time, this must be an Intel hex record file. MYPROG.HEX must be linked to the same address range where the EEPROM is mapped.

The Algorithm file is used for programming external memory devices. It will be explained later.

The **Programming Options** are just what they seem to be. Blank Check makes sure that the EEPROM is blank. Erase makes the EEPROM blank. Program does the actual programming. Verify compares the file in the File Name field to the contents of the EEPROM. If they are not the same, a dialog box will tell you so.

#### Programming External FLASH Memory

If you have designed your target with an external FLASH chip, and you want EMUL296™-PC to program it, you must provide the algorithms for blank checking, erasing, programming, and verifying the contents of your specific EEPROM. The algorithms must be written in instructions executable on the MCU in your target.



**Figure 31. Programming an External Flash EEPROM**

When you click on the **External** button the controls for the on-chip FLASH are grayed. The three active fields become Base Address, Length, and RAM. The first two, Base Address and Length refer to your EEPROM. Set them to match the base address and size of the FLASH you want to program. The RAM field must point to active RAM large enough to hold the algorithms and temporary data. This is where the algorithms will be loaded and executed.

---

*Hint: Many MCU's have internal SRAM modules large enough to hold algorithms and temporary data. Use the Other Presets feature to activate and map the SRAM to the address shown here in the RAM field.*

---

### Programming Algorithms

In order to perform the 4 tasks Blank Check, Erase, Program and Verify, you must provide code that can run on your MCU to perform those tasks. The code is processor specific. Please refer to the template file F\_SKEL.ASM for skeleton examples of each function and specific register names.

To invoke any of these programming algorithm functions, the EMUL296™-PC debugger puts the start of RAM from the RAM field into a register puts the entry point (the RAM base address plus four for BlankCheck) into the program counter, and tells the MCU to start executing. This means that the programming algorithm file must start at the very beginning of RAM and the jump instructions from the template file must be first in the algorithm file.

Every function must end with a TRAP instruction to give MCU control back to the debugger. When the routine is complete, the debugger checks the CPU registers for error information and displays the status to the user.

In addition to the four obvious functions, your algorithm file must include four other functions: Calibrate, Initialize, EnableProg, and DisableProg. Examples of all these functions are also in the file F\_SKEL.ASM for the 296SA.

Calibrate turns off the watchdog timer, adjusts the clock speed, and then counts how many times it executes a small loop in two seconds. This count is divided by 20 and used as a

100 millisecond standard. Your algorithms can use this constant to produce desired delays independent of the target clock rate or your PC speed.

The Initialize function should contain any target or EEPROM specific setup code. It is called once after the calibrate routine.

If you have checked the Program box, the EnableProg function is called once, before the Program function is called. Typically, this function will turn on any necessary programming voltages. Typically, the function DisableProg turns off the programming voltages.

The Program function is called as many times as necessary to completely fill the EEPROM. Before it calls Program each time, the debugger fills the buffer in RAM with some of the data to be programmed and puts the start address in a register. The Program function must copy the data from the RAM buffer to the EEPROM while manipulating whatever bits are necessary.

Verify is similar to Program in that it is called many times in succession with different portions of the EEPROM in the RAM buffer during each call. If the verify function finds a difference, it puts a 1 into the status register, which will get reported to the user.

Menu Item	Hot Key	Function
<b>Save code as ..</b>	<Alt>FS	Write the contents of RAM or ROM to a HEX record file. Any region of memory can be saved to a file for reloading later. Selecting this menu item opens a dialog box that lets you select an address range. Please note that only the HEX file format is supported at this time.
<b>Remove Symbols</b>		Delete all line number and symbolic information, and close source files.
<b>Show load info ..</b>		Display a window describing the object file last loaded including number of variables, address range loaded, etc.
<b>Preferences</b>	<Alt>FP	Controls the way the emulator loads object files.
<b>Exit</b>	<Alt>X	Quit the EMUL296™-PC application. Exiting the EMUL296™-PC software will update the current debugger configuration to either the .ini file or to the current .pro file, if one is selected.

View/Edit Menu

Menu Item	Hot Key	Function
<b>Copy to clipboard</b>	<Ctrl><Ins>	Copy the text (without formatting or font information) of the entire active window to the clipboard.
<b>User defined symbols</b>		This item opens a dialog box that lets you select the module from which you can view symbols.
<b>Default CPU symbols</b>		View and edit memory-mapped registers by name and by the bit.
<b>DDE Status</b>		Open a window displaying the DDE interface status.

---

*Note: DDE is not supported as of this printing. Please check your current version of software.*

---

The **DDE Status** menu item opens a window that displays information about the DDE interface intended for development and debugging.

Please contact a technical support engineer for assistance using the **DDE Status** window. [support@icetech.com](mailto:support@icetech.com)

Menu Item	Hot Key	Function
<b>C call stack ..</b>	<Alt>VC	Opens a child window that displays the C call stack and passed parameters needed to reach the current Program Counter.
<b>Evaluate ..</b>	<Ctrl>E	Open a dialog box that evaluates C expressions. Expressions may contain variables. Assignment expressions may change the values of variables.

---

*Hint: To change the value of a variable, use the **Evaluate** window to evaluate a C assignment expression such as "i=75".*

---

Menu Item	Hot Key	Function
<b>Inspect ..</b>	<Ctrl>I	Open a dialog box that displays the contents of a single variable, structure, or array in detail.
<b>Add a watch point ..</b>	<Ctrl>W	Open a child window that displays groups of variables that is updated every time emulation halts.
<b>Search..</b>	<Ctrl>S	This menu item opens a dialog box that lets you search the active window for the kind of data displayed in that window. If the <b>Source</b> window is active, you can search for text strings within that file. If the <b>Trace</b> window is active, you can search for any trace record. (See the Trace chapter for more details.) In all other windows that support searching, the search is for a hex pattern.
<b>Search next</b>	<Ctrl>X	The last search defined will be performed again, from the cursor forward.
<b>Search previous</b>	<Ctrl>P	The last search defined will be performed again from the cursor backwards.

#### Run Menu

Menu Item	Hot Key	Function
<b>Step into</b>	F7	Execute one instruction, including a jump instruction. If a <b>Source</b> window is selected, execute all the instructions for one line of source.
<b>Step over</b>	F8	Execute one instruction or all the instructions in a subroutine. If a Source window is selected, execute all the instructions for one line of source. Due to some kinds of optimizations, this feature may not always be available.
<b>Animate ..</b>	<Ctrl>F7	Execute instructions continuously and slowly, highlighting each instruction or each line as it is executed.
<b>Go</b>	F9	Begin executing instructions from the current PC at full speed until the next breakpoint.
<b>Go to cursor</b>	F4	Execute the instructions from the PC to the current cursor position.



<b>Go to ..</b>	<Ctrl>F9	Execute the instructions from the PC to the specified address.
<b>Go to return address</b>	<Alt>F9	Execute the instructions from the PC to the next found function return. Due to certain optimizations, this feature may not always be available.
<b>Go FOREVER</b>		Execute instructions from the current PC after disabling all breakpoints.
<b>Break Emulation</b>	F9	Suspend execution as if a breakpoint was encountered.
<b>Reset Chip</b>	<Ctrl>F2	Reset CPU without executing any instructions.
<b>Reset and Go</b>		Reset CPU and begin execution from reset vector.

### Breakpoints Menu

Menu Item	Hot Key	Function
<b>Toggle</b>	F2	Disable or enable existing breakpoints.
<b>Hardware breakpoints ..</b>		Opens a dialog box that lets you set up address ranges for hardware breakpoints (that don't use the trace board).
<b>Break on internal access ..</b>		Opens a dialog box that lets you set up address and data masks that will cause a break on internal bus cycles of the right type.
<b>Hardware breaks only</b>		If this menu item is checked, all program/source window breakpoints will result in hardware breakpoints..
<b>Fast_Breaks ..</b>		Setup and execute a "fast break write" to memory.

---

*Note: "Break on internal access" is not supported as of this printing. Please check your current version of software.*

---

Under the **Breakpoints** menu, select **Break on internal access..** This uses the Intel special emulation chip breakpoint register to break execution when reading or writing bytes or words to register RAM and internal RAM. These read and write cycles cannot otherwise be seen externally with the trace card or in shadow memory. The data as well as address may be qualified and ranges are possible using **Don't Cares** in the mask bits.

Notice that there are 11 address bits and 16 data bits, each with a check box. The address range of Register and internal RAM varies depending on Sx chip member.

To select accesses 100 to 1FF you might enter 100 hex and then uncheck the lower 8 bits:

(✓✓✓✓    \_\_\_\_    \_\_\_\_)

An unchecked bit is "don't care" and a checked bit is significant for either address or data qualifiers. The address and data mask is then ANDed with the entered hex address or data value. A data range of 10 to 1F could be entered as 10 with all boxes except the rightmost 4 bits checked:

(✓✓✓✓    ✓✓✓✓    ✓✓✓✓    \_\_\_\_)

The 4 vertical boxes are checked as follows:

ARD	(Address Range Detect) normally checked
WR	(Write cycle) checked if a write to specified address
RD	(Read cycle) checked if a read from specified address
DRD	(Data Range Detect) normally checked

Leaving all four boxes empty disables any internal breakpoints.

#### EXAMPLE 1:

Use the supplied **time.omf** program and observe that the counter "ticks" is located in register RAM at location 36 hex (not fixed location). It increments from 0 to 19 decimal in the pitr\_int periodic interrupt function before being reset and the seconds incremented. Check boxes ARD, WR, and DRD, then enter "ticks" for the address and 5 for data. All mask boxes should be checked. Exit this pull-down and start the processor with F9. You should observe a breakpoint approximately 8 bus cycles after 5 was written to "ticks". Scroll up in the program disassembly window to see the "INC" opcode or view the last 8 bus cycles in the trace buffer.

## EXAMPLE 2:

Type "1234 hex" in word 36 (use **Data** window) and then type in a LD 20, 36 opcode in the **Program** window. Checking ARD, RD, and DRD, entering 36 as address and 1234 as data (all mask bits checked) results in a breakpoint approximately 8 bus cycles after the read of 1234 from register RAM location 36.

Menu Item	Hot Key	Function
<b>At ..</b>	<Alt>F2	Set a breakpoint by address, line, or line in module.
<b>Setup ..</b>	<Alt>BS	Open a breakpoint editing dialog box.
<b>Disable all</b>	<Alt>Bi	Disable all breakpoints from being active while remaining in the list.
<b>Delete All</b>	<Alt>BD	Clear all existing breakpoints.
<b>Break now!</b>	<Ctrl>C	Immediately halt the emulation.

Config Menu

Menu Item	Hot Key	Function
<b>Project name ..</b>		Choose a configuration or project from a list of existing projects, or create a new one.
<b>Paths ..</b>	<Alt>CP	Sets the default directories for finding load files, source files, and emulator files.
<b>Emulator Hardware ..</b>	<Alt>CE	Sets the emulator board address, controller type, and Chip Select registers reset values.
<b>Miscellaneous ..</b>	<Alt>CM	Sets automatic PC & SP reset value, DDE sampling interval, and memory scroll range values.
<b>Color ..</b>	<Alt>CC	Assign colors to windows.
<b>Full Reset</b>	<Alt>CF	Reloads on-pod logic & performs reset.
<b>Trace ..</b>	<Alt>CT	Please refer to the Trace Chapter for information about the <b>Trace Config</b> dialog box.
<b>PP Analyzer</b>	<Alt>CA	Open a <b>Performance Analysis</b> control window and start recording addresses.
<b>Memory Coverage ..</b>	<Alt>CV	Open the dialog box that controls Memory or Code coverage.

These next nine submenus share one location in the menu bar. The menu displayed corresponds to the kind of child window selected. Selecting a different kind of child window will change which menu is displayed. To select a different window, either use the **Window** menu, or just click the mouse on any part of the desired window.

### Program Menu

Menu Item	Hot Key	Function
<b>Address..</b>	<Ctrl>A	Scroll the selected <b>Program</b> window to the specified address.
<b>Origin (at program counter)</b>	<Ctrl>O	Scroll the <b>Program</b> window to display the PC address.
<b>Set new PC value at cursor</b>	<Ctrl>N	Set the Program Counter to the address at the cursor.
<b>Module</b>	<Ctrl>F3	Open a dialog box that allows quickly scrolling the <b>Program</b> window to the start of any module.
<b>Function</b>	<Ctrl>F	Open a window listing all the functions in all modules loaded. Selecting one will scroll the <b>Program</b> window to the start of that function.
<b>View source window</b>	<Ctrl>V	Scroll (or open) a <b>Source</b> window to show the source at the current Program window cursor.
<b>Toggle breakpoint</b>	F2	Enable or disable a breakpoint at the cursor.

### Source Menu

Menu Item	Hot Key	Function
<b>Address..</b>	<Ctrl>A	Scroll the selected <b>Source</b> window to the specified address, which may be a function name or a label.
<b>Origin (at program counter)</b>	<Ctrl>O	Scroll the <b>Source</b> window to display the Program Counter address.
<b>Set new PC value at cursor</b>	<Ctrl>N	Set the Program Counter to the address at the cursor.
<b>Module</b>	<Ctrl>F3	Open a dialog box that allows quickly scrolling the <b>Source</b> window to the start of any module.

<b>Function</b>	<Ctrl>F	Open a window listing all the functions in all modules loaded. Selecting one will scroll the <b>Source</b> window to the start of that function.
<b>Call stack ..</b>	<Alt>SC	Opens a window that displays the C call stack and passed parameters to reach the current Program Counter.
<b>View assembly code</b>	<Ctrl>V	Scroll (or open) a <b>Program</b> window to the current program counter (not source window cursor).
<b>Toggle breakpoint</b>	F2	Enable or disable a breakpoint at the cursor.

### Data Menu

Menu Item	Hot Key	Function
<b>Address..</b>	<Ctrl>A	Scroll the selected <b>Data</b> window to the specified address.
<b>Original Address</b>	<Ctrl>O	Scroll the selected <b>Data</b> window to the last address used in an <b>Address..</b> menu command.
<b>Edit ..</b>	<Enter>	Alter the contents of the highlighted location.
<b>Block move..</b>	<Ctrl>B	Move a segment of RAM to another location (in RAM).
<b>Fill..</b>	<Ctrl>F	Fill RAM with the specified value or pattern.
<b>DataDisplay as..</b>	<Ctrl>D	Set the data display mode (ASCII, hexadecimal bytes, long integers, etc. See page 50 of the manual for the complete list of formats).
<b>Address space ..</b>	<Ctrl>- <Space>	Set the address space for the selected <b>Data</b> window.

### ShadowRam Menu

---

*Note: ShadowRAM is not supported as of this printing. Please check your current version of software.*

---

Menu Item	Hot Key	Function
<b>Address..</b>	<Ctrl>A	Scroll the selected <b>ShadowRam</b> window to the specified address.
<b>Original Address</b>	<Ctrl>O	Scroll the selected <b>ShadowRam</b> window to the last

		address used in an Address.. menu command.
Display as..	<Ctrl>D	Set the data display mode (ASCII, hexadecimal bytes, long integers, etc. See page 50 of the manual for the complete list of supported formats).

### Register Menu

Either select a register then select this menu item, or more simply, select a register and type a new value. The first character typed will open the same dialog box as selecting the **Edit** menu.

### Trace Menu

Please refer to **Chapter 4: Trace Board** for all information regarding the Trace board and user interface.

### Stack Menu

Menu Item	Hot Key	Function
Parameters in Hex	<ALT>SP	Display the function parameters in hex instead of in their declared type.
Show function	<ALT>SS	Not implemented at this time.

### Watch Menu

Menu Item	Hot Key	Function
Add ..	<Insert>	Open a dialog box for adding a variable to the <b>Watch</b> window.
Edit ..	<Enter>	Open a dialog box for editing an existing variable in the <b>Watch</b> window.
Remove ..	<Delete>	Delete the selected variable from the <b>Watch</b> window.

### Window Menu

The **Window** menu items open new windows, close existing windows, select windows, and arrange windows on the screen.

Menu Item	Hot Key	Function
Open a new	<Alt>WNP	Open a new <b>Program</b> window.

<b>program window</b>		
<b>Open a new source code window</b>	<Alt>WNS	Open a <b>Source</b> window.
<b>Open a new data window</b>	<Alt>WND	Open a <b>Data</b> window.
<b>Open a new register window</b>	<Alt>WNR	If one is open, it will ask "Are you sure?"
<b>Open a new shadow ram window</b>		Open a new <b>ShadowRam</b> window. (Note: ShadowRam is not supported as of this printing. Please check your current software version.)
<b>Open a new Special Registers window</b>	<Alt>WNE	Open a new <b>Special Registers</b> window.
<b>Open a new trace window</b>	<Alt>WNT	Open a new <b>Trace</b> window.
<b>Open a new Watch window</b>	<Alt>WNW	Open a new <b>Watch</b> window.
<b>Toggle help line</b>	<Alt>WH	Turn on or off the text at the bottom of the EMUL296™-PC window.
<b>Refresh</b>	<Ctrl>R	Repaints the screen.
<b>Tile windows</b>	<Alt>WT	Resize and arrange the windows within the EMUL296™-PC application.
<b>Cascade windows</b>	<Alt>WC	Resize and overlap the windows within the EMUL296™-PC application.
<b>Arrange Icons</b>	<Alt>WA	Line up any closed EMUL296™-PC icons at the bottom of the main window.
<b>Zoom</b>	F5	Expand the selected window to fill the EMUL296™-PC window.
<b>Next window</b>	<Ctrl>F6	Change the currently selected (highlighted) window.
<b>Close</b>	<Alt>F4	Close the currently selected window.

Below the **Close** menu item, there is one menu item for each open window, and the active window will be checked. Selecting one of these items will open the window if it is closed down to icon size, and activate it.

### Help Menu

Selecting the **Info ..** menu item will open a box that displays the application version number and date. Please have this information handy when calling for support.

## **Dialog Boxes**

Many menu selections open dialog boxes that allow you to input more specific information. Some of these dialog boxes are described above next to their menu items. The rest are described in this section.

### Child Windows

There are nine primary child windows created by EMUL296™-PC: **Program** windows, **Data** or **Memory** windows, **Inspect** windows, **Source** windows, a **Registers** window, a **SpecialRegs** window, **Call Stack** window, **Watch** windows, and **Trace** windows (even if you have no Trace board). All of these windows are opened by selecting the corresponding item in the **Window** menu.

Any number of child windows may be open at the same time. Any number of child windows can overlap but only one child window is active (has the focus) at a time. Some may be scrolled and resized to view any address desired. Their locations and sizes are saved to the current project file when EMUL296™-PC exits, and will be restored when the software restarts.

Each child window has a corresponding menu that appears between the **Config** menu and the **Window** menu. The menu contains items that only make sense within the context of that window. This window-specific menu will also appear at the cursor when you click with the Right mouse button in the body of the active window.

### Register Windows

The **Registers** window displays the CPU registers. All registers are displayed in hexadecimal notation. Clicking anywhere in the **Registers** window will select that window (make it the active window) and right-clicking brings up the **Registers** menu. The operation supported in the **Registers** window is editing register contents.



The menu also lets you turn on or off, a display mode that, whenever the window is updated, compares the current values with the last values displayed and highlights (displays in a different color) the registers that have changed.

### Data and Shadow RAM Windows

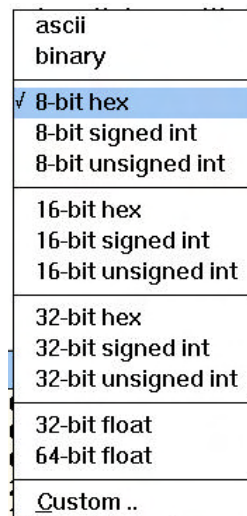
---

*Note: ShadowRAM is not supported as of this printing. Please check your current version of software.*

---

Use **Data** windows to examine or modify emulation or target memory directly. EMUL296™-PC uses the controller to read and write RAM, so the **Data** window cannot be updated while the emulation is running. Instead, asterisks will be displayed until the next time the controller starts executing monitor code.

Data can be displayed or modified in various formats as shown in Figure 32:



**Figure 32. Data Display Formats**

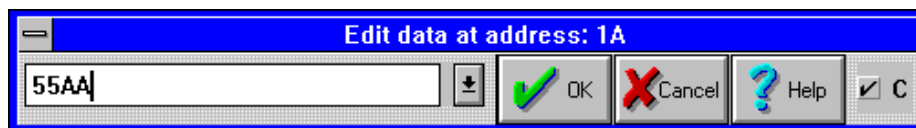
---

*Note: 32 and 64 bit IEEE\_754 floating point numbers must be word aligned. Some compilers support packed structures that can have floating point fields that start on an odd address. These fields will not be displayed properly in a **Data** window.*

---

Selecting any **Data** window displays the **Data** menu which supports filling memory, jumping the selected window to a specific address, setting an address space, and setting the display mode (hex, ASCII, etc.) options .

Changing a value at any memory location is as easy as selecting the byte, word, or long word to change and then typing the new value. The first character you type will open a small data entry window, shown in Figure 33.



**Figure 33. Editing Memory with a Data Window**

Always enter the new data in the same format that the data is displayed. If the **Data** window is displaying ASCII characters, type the new character (not a string) in ASCII. If the **Data** window is displaying signed integers, enter the new value as a decimal number. Symbols are supported but their type is ignored.

If you display the data in bytes, only a byte will be written to memory for each update. In other words, updating one byte uses a single bus cycle that is one byte wide. On the far right side of the **Edit data** dialog box is a small check box labeled with the letter C. This check box impacts how the emulator interprets the data you enter. If you have a symbol named "abcdef" and you are displaying in 16 bit hex, it is not clear whether to interpret "abcdef" as a symbol name or as a hex number. With the box checked, the emulator uses C syntax first, so it will be treated as a symbol name. Without the C box checked, assembler rules apply first, and it will be interpreted as a hex number (see far right edge of Figure 33).

### Custom Display Format

Selecting the custom format option opens a dialog box that lets you input a C printf format string. All standard C formats are allowed, including the newline character. If you are trying to display odd address integers or floating point numbers, you must use the custom display format.

### Program Windows

A **Program** window disassembles and displays code memory. One line in the **Program** window is always highlighted. This is the cursor. The color of the highlighting and the window depend upon how you have configured your color settings. (See page 27 for information about how to change the color settings.) Use the cursor to set and disable breakpoints, set the program counter, and invoke the in-line assembler.

The first column is the hexadecimal address. If the address is highlighted, there is a breakpoint at that address. You may set or inactivate a breakpoint by clicking on the address. The second column is the hexadecimal value at that address. Between the address and the hexadecimal data may be an arrow pointing to the right, indicating the current program counter. The third column contains the disassembled instructions and operands.

A comment will sometimes appear to the right of the highlighted instruction. The comment displayed is a function of the kind of instruction and is a hint about what will happen when

the instruction is executed. For example, if the highlighted instruction will change the contents of memory, the hint will contain the value about to be overwritten.

**Program** windows can control the emulation. To set a breakpoint, click once on the address portion of the instruction where you want the break. Or, you may click once on the desired instruction (to highlight that instruction) and then click on it again to highlight the address. A breakpoint is indicated by displaying the address with white letters on a black or dark background. This second mouse click (not a double click) creates the breakpoint. To deactivate (not delete) that breakpoint, click again on the same instruction. The address will no longer be highlighted and the breakpoint will be inactive. To delete the breakpoint, use the **Setup ..** dialog box from the **Breakpoints** menu. Any highlighted instruction can be a temporary breakpoint. The Run menu item Go until cursor will use the cursor as a temporary breakpoint.

### In-line Assembler

The in-line assembler is easy to use; simply highlight the instruction or address you wish to change in the **Program** window and type. The first character typed will open an edit dialog box to display the characters you type and allow you to edit your assembler source line. Once the source line is as you want it, press <Enter>.

The in-line assembler will translate the input line according to the syntax described in the 80C296 data books and replace the former opcode(s) and data with the new opcode(s) and data. Note that the assembler will write as many bytes as required for the new instruction. This may overwrite part or all of subsequent instructions. Be sure to examine the subsequent instructions as well as the new instructions for correctness.

### Source Windows

The **Source** window displays the C source (or assembler source if the assembler supports source line debugging) of the module containing the Program Counter. Like a **Program** window, a **Source** window displays the source text, line numbers, a cursor (the blinking underline), and a small arrow between the line numbers and the source text to indicate the current Program Counter value.

After each single step, and during each animation pause, the **Source** window scrolls to show the source line that generated the instruction pointed to by the new Program Counter, if it was generated by a source line.

Displaying and toggling breakpoints in **Source** windows is different than in **Program** windows. In **Source** windows, breakpoints are displayed by inverting (or highlighting) the entire source line. In **Program** windows, only the address is highlighted. In **Source** windows, a single click on any line number (or address in the **Program** window) will toggle the breakpoint. In both kinds of windows, pressing **F2** will toggle a breakpoint on the highlighted instruction.

When a **Source** window appears blank with the window title "Source", it usually means that the program counter is pointing to instructions derived from a module with no debugging information. As soon as the PC points to an instruction from a C module or assembly

module with line number symbols, the **Source** window will show that text, and the title on the window will change from "Source" to the name of the source file being displayed.

The simplest way to find the first line of source is to reset the controller, click on the **Source** window title bar to select it, and then execute a single step by pressing the **F7** key (or by clicking on the **Step** button on the speed bar).

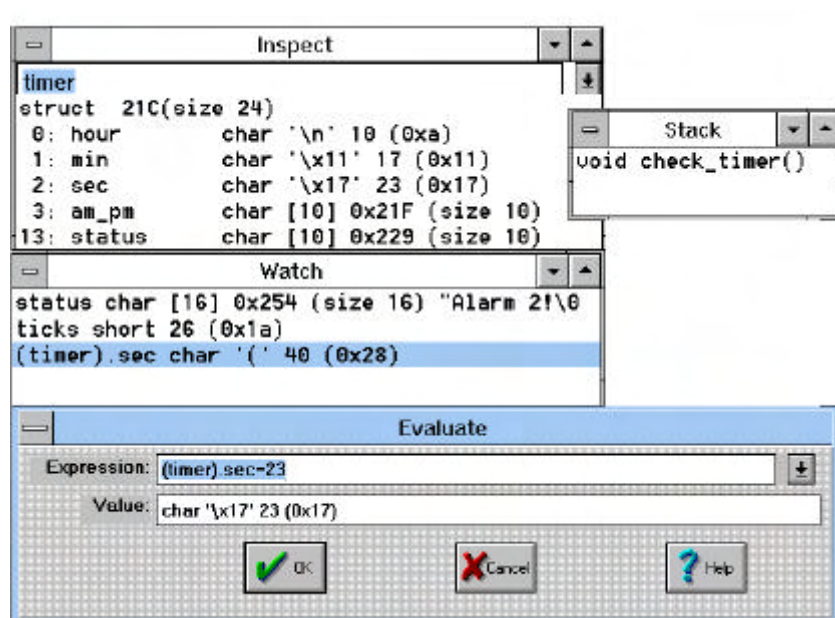
When the **Program** window is selected, a single step means a single opcode. The same is true for animated execution: a pause occurs after every opcode is executed. When the **Source** window is selected, a single step means a single source line. Animation will execute faster when the **Source** window is selected than when the **Program** window is selected because most source lines compile into more than one machine instruction. If the animation is running faster or slower than you expect, or if single stepping executes more or fewer instructions than you expect, visually confirm that the selected window is the one you want to be selected. If in doubt about which window is selected, click on the title bar of the window you wish to be selected.

### Trace Window

For information about the **Trace** window, please refer to the Trace Board chapter.

### Other Windows

Three more child windows used for high level debugging in C are available: the **Evaluate** window, the **Inspect** window, and the **Watch** window. These windows are opened by selecting their respective items in the **View/Edit** menu. Like the other child windows, selecting one of these open windows will bring a corresponding menu up between the **Config** and **Window** menus.



### Figure 34. C call stack, Evaluate, Inspect & Watch Windows

#### Inspect Window

The **Inspect** window displays a single variable, or possibly modifies that variable. To open an Inspect window, either select the Inspect .. menu item in the **View/Edit** menu or double click in the **Source** window on the variable you would like to inspect. Double-click in an open Inspect window on a structure member or array element to open an **Inspect** window detailing that field.

The **Inspect** window can stay open just like a **Data** or **Watch** window, and it will be updated whenever the application stops. The variable being displayed may be part of an equation written following the rules of C that produces a single scalar answer.

---

*Note: If you have an open **Inspect** window with an assignment statement, every time the emulator stops executing, the expression will be evaluated and the variable will be updated. The variable will appear as though your application is not changing it while the emulator is running.*

---

#### Watch Window

The **Watch** window displays multiple variables being watched, one variable per line. Any local variable in the **Watch** window that is not in scope will be displayed with three question marks instead of its value.

Place the cursor on the variable of interest and use <CTRL>W to add it to the **Watch** window.

#### Evaluate Window

The **Evaluate** window is opened by selecting a variable in the **Source** window with the cursor and using <CTRL>E. This allows editing of the current variable by using the C assignment operator = to the right of the variable. In fact, any C expression may be performed in this edit window.

---

*Hint: This window can also serve as a hexadecimal calculator, using the C syntax 0x\_\_\_\_\_ for hex numbers.*

---

#### Stack Window

The **Stack** window displays the "call stack," or the list of functions called to reach the current point in the application, and the current value of parameters passed to them (only supported if the compiler provides the stack information).

Addresses are displayed and entered using hexadecimal notation or global symbol names. In all windows (excluding Inspect windows,) values may be edited by selecting that value (with the mouse or cursor keys) and then typing.

---

*Note: Symbol names are case sensitive. If a symbol cannot be found, try the same name with a different case. Also note that some assemblers shift all symbols to uppercase.*

---

### RTXC Window

---

*Note: RTX is not supported as of this printing. Please check your current version of software.*

---

EMUL296™-PC has built-in support for the RTX multitasking kernel from Embedded Systems Products.

Once you have opened an RTX window, you have an interface that is nearly identical to the debugging interface of RTX itself. Just as with the kernel debugging command interface, you must type an exclamation mark (!) to halt normal kernel and task execution. At that point you can type commands that will display information about the kernel and any of the tasks. This information includes task priorities, message queues, stack usage, etc. The "H" command will show you a summary of commands.

For detailed information about using the RTX debugging features, please refer to manual that comes with RTX.

### **Tool Bar**

Just below the menu bar is the "Tool Bar" containing icons or buttons that, like Hot Keys, execute frequently needed menu options when clicked. The Help button opens the *MS Windows* Help application to the page that describes the current context. The Reset button resets the controller. The **Step** button emulates one source line or opcode depending upon which window was last active. The **Go** button starts full speed emulation that will continue until a break occurs. While emulating, the **Go** button changes to **Break**, and halts emulation when clicked. The **Trace Beg** button resets the Trace board and starts bus cycle recording according to the conditions set in the **Trace Setup** dialog box.



**Figure 35. The Tool Bar**

## Help Line

At the bottom of the EMUL296™-PC window is a line of text that, depending upon the context, explains what the selected item is or what it does. This kind of context-sensitive help is turned on and off with the **Toggle help line** item in the **Windows** menu.

## Dynamic Data Exchange

---

*Note: DDE is not supported as of this printing. Please check your current version of software.*

---

Dynamic Data Exchange (DDE), allows one *MS Windows* application to send data to another. EMUL296™-PC uses DDE protocols and export values from ShadowRam to other applications such as *Word for Windows* or *Excel*. You cannot import values to EMUL296™-PC using DDE. The following example will help you set up the DDE link between EMUL296™-PC and *Excel* (or *Excel* inside *Word for Windows*).

---

*Note: ShadowRAM is not supported as of this printing. Please check your current version of software.*

---

To establish a link from EMUL296™-PC to *Excel*, select the destination cell and enter the following formula:

```
=emul296|shadow! '< Shadow RAM address to monitor in hex notation >'
```

where <> is an address ("5000", for example).

All the pods come with an example file named "TIME.OMF". The TIME program writes the value of a timer at the location 5010H through 5030H (look at the Shadow RAM in Figure 36). In the **ShadowRAM** Window you should choose **Display as... ASCII**. You will see the seconds and tenths of seconds being updated in your *Excel* cell, if you do the following:

Select the destination cell in *Excel* and enter the following formula (see Figure 36, cell A1):

```
=emul296|shadow! '5016'
```

The cell will be updated and displayed as words in decimal representation. If you would like to have it displayed as ASCII, you must write your own function as a Macro under *Excel*. In Figure 36 we make a call for function *AsciConv* in cell A2. The function might look like:

```
Function AsciConv(word)
```

```
AsciConv = Chr(word And 255) + Chr((word / 256) And 255)
```

```
End Function
```

The value in the *Excel* cell will be updated as often as indicated in the **Config Miscellaneous ..** dialog box labeled DDE sampling interval. This may be as little as 100 milliseconds or as long as 32K milliseconds (32s).

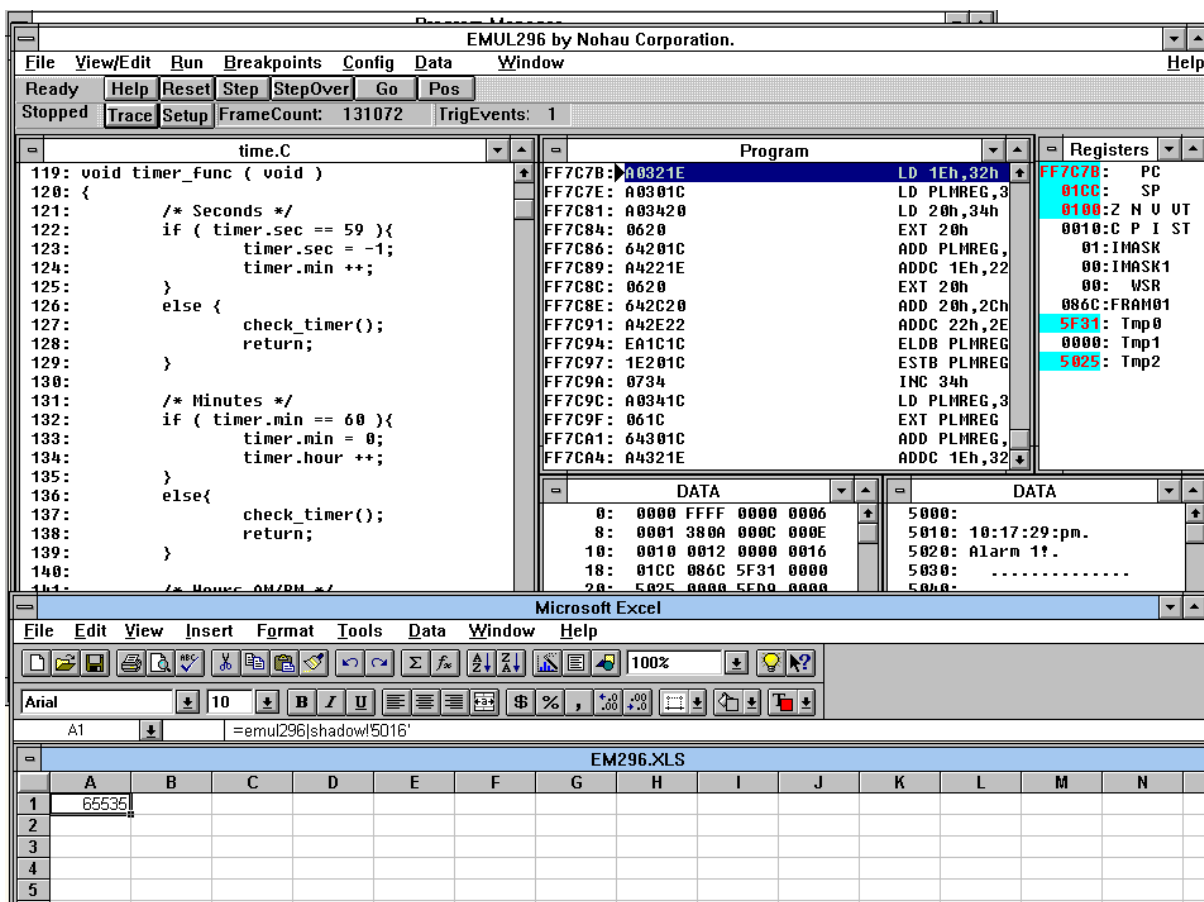


Figure 36. EMUL296™-PC with DDE to Excel

*Note: Editing the memory locations that represent the Stack Pointer, Imask, and WSR in a **Data** window will not update the **Register** window display or the target CPU registers. Always use the **Register** window to edit these registers. In the future, the software will be revised to handle the **Register** and **Data** windows transparently.*





## Chapter 2: Emulator Macro User Guide

### ***Introduction***

The Emulator Macro System consists of two files together with this guide. The System requires the 16 bit version of *Microsoft Visual Basic 4* as its foundation. Macro creation uses many of the features of *Visual Basic*, such as debugging. The macro writer uses a low level DLL and a library of useful functions. Additional functionality is easily added by the macro writer. Run macros from *Visual Basic* for testing and debugging or as stand alone executables.

### General description of the emulator macro setup

There are two files provided in the Emulator directory:

emul296.bas	the <i>Visual Basic</i> API
emul296m.ico	the icon to use for executable macros

*Note: Install Visual Basic before using these files.*

The following is a guide to installing the macro system for writing macros and a reference to the subroutines provided.

### Visual Basic Supplemental User Guide

It's easy to write macros:

1. Execute Visual Basic

Set up your icon

1. Make the form invisible
2. Add the emul296.bas file to your project
3. Name your project
4. Write your macro code
5. Create your executable macro
6. Exit Visual Basic
7. Test it!

For detailed instructions, please refer to the section entitled "Procedure for Writing a Macro".

### General information

Begin by installing *Visual Basic* Standard or Professional edition according to the instructions from *Microsoft*. Use the Standard package for writing macros, and the Professional package for more sophisticated or database work.

**Warning:** *DO NOT change emul296.bas. Use another module if additions are needed.*

We recommend the following *Visual Basic* options to be found under the **Options | Environment** Main Menu selection.:

Require Variable Declaration = Yes

Syntax Checking = Yes

Default Save As Format = Text

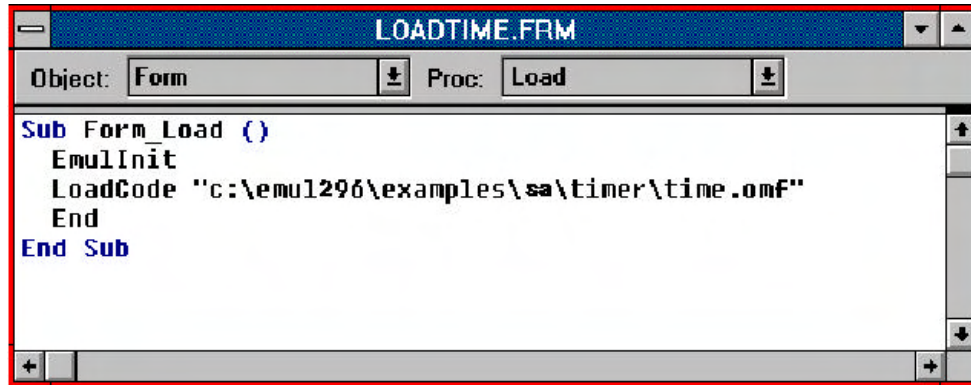
Save Project Before Run = Yes

*Note: Refer to the Visual Basic manual for more details about the program, as this document contains only the information required for writing macros.*

### Procedure for writing a macro

1. Execute *Visual Basic*, or if it is running, create a new project from the *Microsoft Visual Basic* main menu with **File | New Project**
2. Set up an icon in the main form **Properties** window (use the icon "emul296m.ico" for this purpose).
  - a. In the **Properties** window, click on the **ICON** selection
  - b. At the top of the **Properties** window, click on the "..." button
  - c. In the **Load Icon** dialog box, select the directory where the icon is located (i.e., c:\emul296)
  - d. Select the emul296m.ico icon under **File Name**; Click **OK**
3. Set visible to "false" in the property window to make the main form invisible.
  - a. In the main form **Properties** window, select the "visible" entry
  - b. Click drop down arrow in the **Change** Window
  - c. Click on **FALSE**
4. In the *Microsoft Visual Basic* Main Menu, select **File | Add File...** to add the emul296.bas file to the project
  - a. In the **Add File** dialog box, select the c:\emul296 directory
  - b. Select emul296.bas; click **OK**
5. Name the form in the **Project**, and Save
  - a. In the **Properties** box, click on **NAME**
  - b. Type in the new name (e.g., "LoadTime") instead of the default "form 1"
  - c. In the *Microsoft Visual Basic* main menu, select **File | Save Project As**
  - d. Save changes to "LoadTime.frm"? Choose **Yes**
  - e. The **Save File As** dialog box should show "LoadTime.vbp" in c:\emul296 directory; click **OK**
  - f. In the **Save Project As** dialog box, change the file name of the project file

- to "LoadTime.mak"; click **OK**
- g. The result is a project window with the name "LOADTIME.VBP"
6. Write code in the Form\_Load subroutine
- In the **Project** window, select the form ("LoadTime.frm")
  - Click the **View Code** button
  - In the **Forms Code** window, select "FORM" from the object drop down box
  - Type in the line (indented): "EmulInit"; press Enter
  - Type in the body of the macro code (see example below)



**Figure 37: Macro Code Example**

- f. Type "END", allowing the macro to exit.
7. When the program is finished, make an executable program from the file menu using the **File | Make .EXE** command
- Select **File | Make.Exe**
  - The MAKE.EXE dialog box appears; check to ensure it shows the correct file name and directory

**NOTE:** *Change the Application Title at this time if desired.*

- c. Click **OK**
8. **File | Exit**; For any "Save Changes?" Choose **Yes**

***Example of a Macro:***

The following is in the general / declarations section of the main form:

Subroutine	Description
Option Explicit	Force explicit declaration of variables

<code>Sub Form_Load ( )</code>	This subroutine executes when the macro loads and appears in the <b>Form, Load</b> section of the main form
<code>Dim byt As Integer</code>	Declare byt as an integer variable
<code>Dim ret As Integer</code>	Declare ret as an integer variable
<code>Dim dwd As Long</code>	Declare dwd as a long variable
<code>Const ShadowRam = 1</code>	Const ProgramWin = 2
<code>Const RegisterWin = 3</code>	Const DataWin = 4
<code>Const SourceWin = 5</code>	Constants for activating the windows of project: proj1 <sup>1</sup> (see footnote below)
<code>EmulInit</code>	Initialize the macro library and the Emulator (if not running)
<code>LoadProject "proj1"</code>	Load project set up
<code>LoadCode "c:\emul296\examples\kr\timer\t ime.omf"</code>	Load the code and symbol file
<code>WindowSelect SourceWin</code>	Select the <b>Source</b> window
<code>StepOver</code>	Step a couple of times
<code>StepOver</code>	
<code>WindowSelect DataWin</code>	Select the <b>Data</b> window
<code>SetAddress &amp;H5000</code>	Set the start address to 0x5000
<code>DisplayAs DT_8BITHEX</code>	Set the format to 8 bit hex; the DT_xxx constants are found in emul296.bas
<code>WindowSelect ShadowRam</code>	Select the <b>ShadowRam</b> window
<code>SetAddress &amp;H5000</code>	Set the address to 0x5000
<code>DisplayAs DT_ASCII</code>	Set the format to ASCII
<code>WindowSelect ProgramWin</code>	Select the <b>Program</b> window
<code>SetAddress &amp;H3000</code>	Set the address to 0x3000

<sup>1</sup> For the current version of the macro language, we suggest that you use "project" to specify the MDI child window locations and their order (see "Projects" on page 18.)

SendKeys "nop{ENTER}nop{ENTER}nop {ENTER}ljump 3000{ENTER}", True	Assemble some code into the <b>Program</b> window. The last parameter (True) is used to wait for SendKeys to finish before going on the next statement
PutPc &H3000	Set the PC to the assembled code
byt = 0	Clear the byt variable
ret = GetShadowByte(&H5010, byt)	Get the byte at 0x5010
MsgBox "Shadow byte at 0x5010 = " + Hex\$(byt) + "h.", MB_OK, "LoadForm"	Display the value (MB_OK is found in emul296.bas) in a message box
Go	From 0x3000
For dwd = 0 To 100	A wait loop
DoEvents	Let someone else do something
Next	
Break	Stop executing
dwd = GetPc()	Get the current PC
MsgBox "The PC is: 0x" + Hex(dwd) + ".", MB_OK, "Test Macro"	Display it
EmulReset	Prepare to run the previously loaded code module
SetBpAtLine 138	Set a breakpoint
GoToBP	Go until a breakpoint is encountered
End	Exit from macro
End Sub	

9. Enter **File Manager** and select emul296 directory
  - a. Drag the executable file (in this case "LOADTIME.EXE") from the **File Manager** into a file folder in the **Program Manager**
  - b. Execute the program by double clicking on the icon
  - c. Drag the project file ("LOADTIME.MAK") from the **File Manager** into a file folder in the **Program Manager**
  - d. To edit your macro, execute *Visual Basic* by double clicking on the icon produced from step c

Subroutine Reference**Constants**

Use constants for the *Visual Basic* message box subroutine and the **DisplayAs** values for the different data formats. Refer to *emul296.bas* for the names of these constants. Any other constants needed are documented in the *Visual Basic* or *Visual Basic Professional* Help files.

**Global Variables**

Use the following four global variables (DO NOT modify them in any way):

<b>EmulHandle</b>	This is the <i>Windows</i> handle for the emulator program.
<b>EmulName</b>	This is the full emulator program name as appears in the title bar.
<b>EmulIniName</b>	This is the name of the current emulator ".INI" file.
<b>EmulWorkDir</b>	This is the current working directory (usually the directory that the emulator program resides in).
<b>Windows API</b>	If needed, the <i>Windows</i> API is documented in the <i>Visual Basic Professional</i> Help File system.

Windows API

If needed, the *Windows* API is documented in the *Visual Basic Professional* Help File system.

**Warning:** Use either fixed strings (*Dim str As String \* 100*) or variable strings (*Dim str As String*) that have been pre-initialized, for example using *Space\$ ()* to make the string as long as needed for the API call.

Subroutines

Subroutine	Description
<b>Sub Break ()</b>	Stops emulator execution.
<b>Sub DisplayAs (ntype As Integer)</b>	Changes the current display format in the active window. (Note that this can only be used in windows that have a display format.) The window must be made active with <i>Window Select</i> .
<b>Sub EmulInit ()</b>	Sets up the macro for execution and starts the emulator program if it is not running.

Sub EmulReset ( )	Performs a normal emulator reset.
Function FindEmulWindow ( ) As Integer	Used internally to set up the global variables, and is not normally used in macros.
Function GetByte (ByVal address As Long) As Integer	Gets a byte from data memory.
Function GetDWord (ByVal address As Long) As Long	Gets a long or double word from data memory.
Function GetPc ( ) As Long	Gets the current value of the program counter.
Function GetPsw ( ) As Long	Gets the current PSW.
Function GetShadowByte (ByVal address As Long) As Integer	Gets a byte from ShadowRam.
Function GetShadowWord (ByVal address As Long) As Integer	Gets a word from ShadowRam.
Function GetSp ( ) As Long	Gets the current stack pointer value.
Function GetWord (ByVal address As Long) As Integer	Gets a word from data memory.



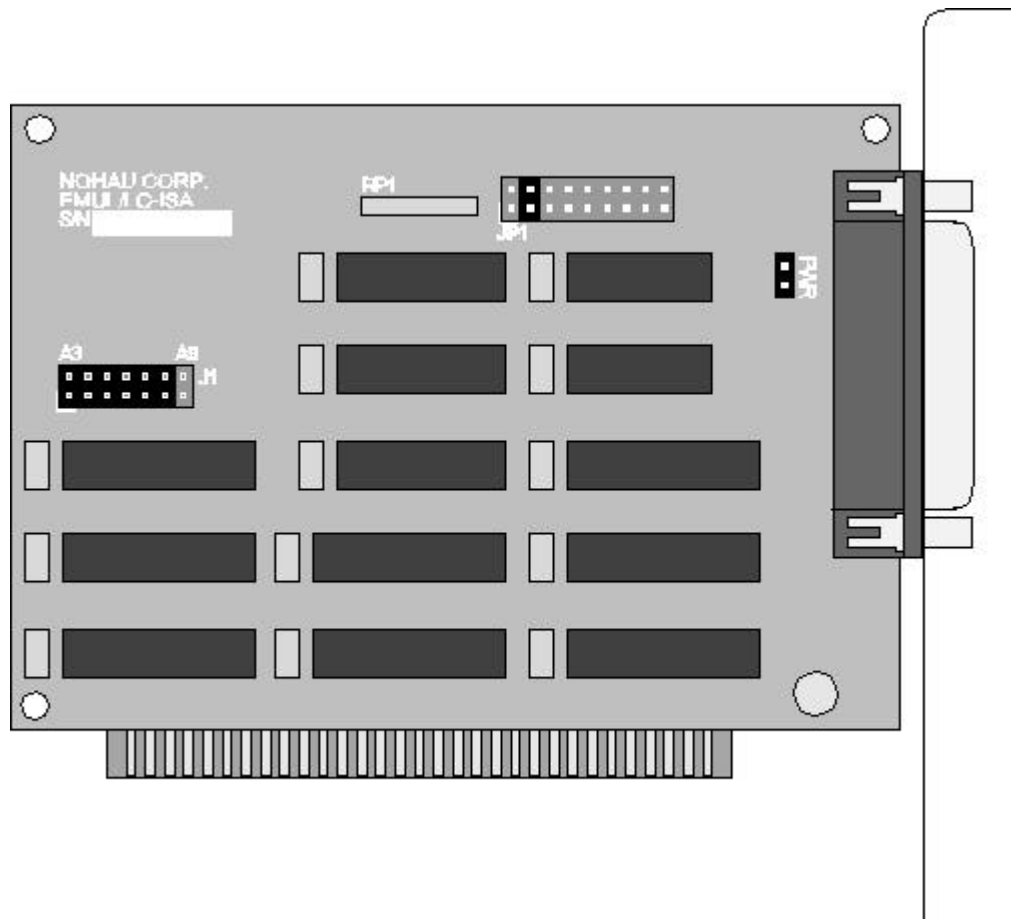
Sub Go_()	Starts executing the emulator at the current address.
Sub GoToBP ()	Starts executing the emulator from the current address and waits until a breakpoint occurs.
Sub LoadCode (codefilename As String)	Loads a program for emulation.
Sub LoadProject (pname As String)	Loads an emulator project setup file.
Sub ModuleSelect (module As String)	Selects a new module in the current program.
Sub PutByte (ByVal address As Long, byt As Integer)	Writes a byte into the data memory.
Sub PutDWord (ByVal address As Long, ByVal dwrd As Long)	Writes a long or double word into data memory.
Sub PutPc (ByVal address As Long)	Sets a new value in the program counter.
Sub PutPsw (ByVal pswreg As Long)	Sets a new value in the PSW.
Sub PutSp (ByVal address As Long)	Sets a new value in the stack pointer.
Sub PutWord (ByVal address As Long, ByVal wrd As Integer)	Writes a word into data memory.
Sub RePaint ()	Redisplays everything in the emulator main window.
Sub SaveTraceText (filename As String, startframe As Integer, stopframe As Integer)	Save the current trace from the "startframe" number to the "stopframe" number into the file name specified as a text file.
Sub SetAddress (address As Long)	Sets an address in all windows that can have an address setting. The correct window must be made active with <b>Window Select</b> .
Sub SetBpAtAdr (ByVal address As Long)	Sets a breakpoint at the requested address.
Sub SetBpAtLine (ByVal lineno As Integer)	Sets a breakpoint at the requested line number.
Sub SetTrigger (ByVal trignum As Integer, ByVal active As Integer)	Make the trigger "1, 2, 3, or filter" active or inactive.
Sub StepInto ()	Executes one instruction, including a jump instruction.
Sub StepOver ()	Executes one instruction or all the instructions in a subroutine.
Function Sym (Symname As String)	Gets numeric value of a symbol.

As Long	
Sub WaitUntilReady ( )	Wait for the emulator to become READY.
Sub WindowSelect (number As Integer)	Activate one of the current child windows, e.g., <b>Data</b> window, by the number on the Window pop-up menu.



## Chapter 3: Emulator Board

## ***EMUL/LC-ISA Emulator Board***



**Figure 38: EMUL/LC-ISA Emulator Board**

The EMUL/LC-ISA board is an 8 bit P.C. card that fits into any slot. The jumpers on the emulator board control three things: 1. the address used to communicate with the Host PC, 2. the maximum PC clock communication rate to the target, and 3. whether or not power is provided to the target through the LC connector. These are all described in more detail below.

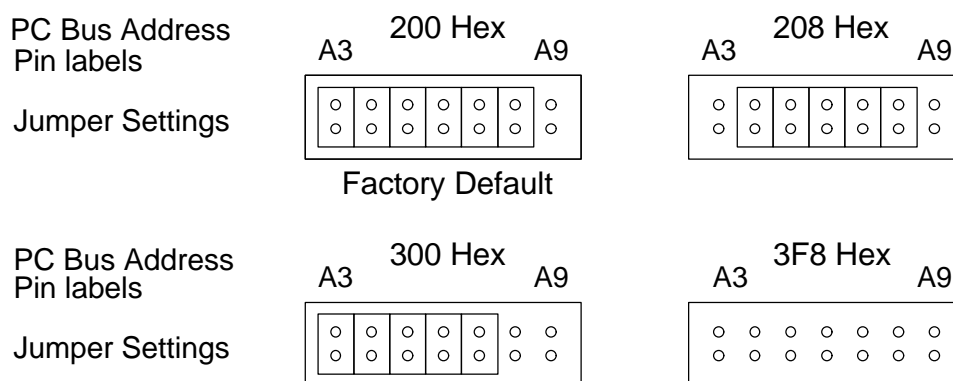
Note: *The power jumper J2 should be left in for EMUL296-PC users.*

## Detailed Installation Instructions

### Setting the I/O address jumpers -- J1

*Note: The factory default is 0x200 for the software and hardware.*

The EMUL/LC-ISA requires 8 consecutive I/O address from the PC's I/O address space (0 Hex -- 3FF Hex) that begin on an address that is a multiple of 8. Set the emulator board address using the jumpers in header J1. These addresses must not conflict with any other I/O device. Each pair of pins in J1 represents on bit in the 10 bit address. Address bits 0, 1, and 2 represent addresses within the 8 consecutive addresses and do not have pin pairs to represent them. This leaves 7 address bits (pin pairs) to set with jumpers. Shorting pins represents a 0 in the address. A pair of pins with no jumper represents a 1. Below are 4 examples where the Least Significant Bit (LSB) is on the left, as it is on the board, if you are holding the board so you can read the silk-screened labels, with the 25 pin **D** connector on the right.



**Figure 39: Emulator Header J1**

### Setting the Target Communication Rate -- Header JP1

The PC's system clock is divided by moving the jumper on JP1.

Set the fixed synchronous communication rate by using Figure 40 to look up the clock rate in the lower row and place one jumper on the header JP1 between the pins indicated in the upper row. There must be only ONE jumper on this header.

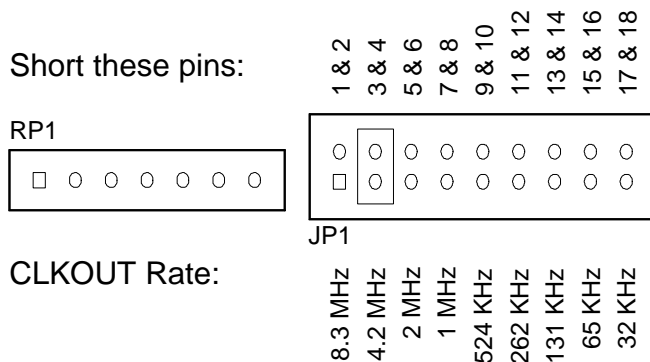


Figure 40: Header JP1

*Note: The pins on JP1 are not numbered on the board. The picture above shows the orientation of both JP1 and RP1 as they appear on the emulator board. Both pin 1 holes are shown as a square, as they are on the emulator board.*

#### Communication Rate Jumper

The communication rate jumper MUST be set in position 3,4.

#### Trace Clock Rate

This clock is used for logic download to the Pod. If you have any problems starting the Emulator software when using an external crystal that is less than 16Mhz; you must modify the .INI file entry for trace clock rate (*see below*) even if you are not using a trace board.

#### **[TRACE]**

**clock rate=** *Enter the POD crystal clock rate here.*

#### The PWR Header -- JP2

*Note: Leave this jumper in place.*

The third header on the low cost emulator board is the PWR header, which is also labeled JP2. With the jumper in place, +5 volts is supplied from the PC's power supply through the LC connector up to .5 amps.

#### Power Supply to Pod / Target

The power supply to the pod / target is controlled by jumper(s) on the POD boards. See the POD-296-256-SA-50 section for information on this jumper.



## Chapter 4: Trace Board

### ***Trace Board Introduction***

EMUL296™-PC needs RAM to record a history of the data used and instructions executed. The trace board contains this RAM. The Pod board has the logic and connectors necessary to support a trace board. The card includes 96 bits of RAM for each trace record. Trace boards are available with two sizes of trig memory: 256K or 1M.

### ***Trace Board Detailed Installation Instructions***

There are two configuration settings related to the hardware that must be set correctly before the trace board can be used. These are both found in the upper left corner of the **Config Trace** dialog box.

First, the software must know that there is a trace board in the PC. To indicate that there is a trace board installed, click on the **Yes** button next to **Board installed:**. If the top of the **Trace** window says "Not available" then this option is probably set to disable the trace board. Figure 41 shows the normal default settings for an installed trace board.



**Figure 41. Trace Board I/O Address Setting**

Also critical is **I/O address:**. Be sure to set this field to the same base address as the emulator.

### ***External Inputs and Controls***

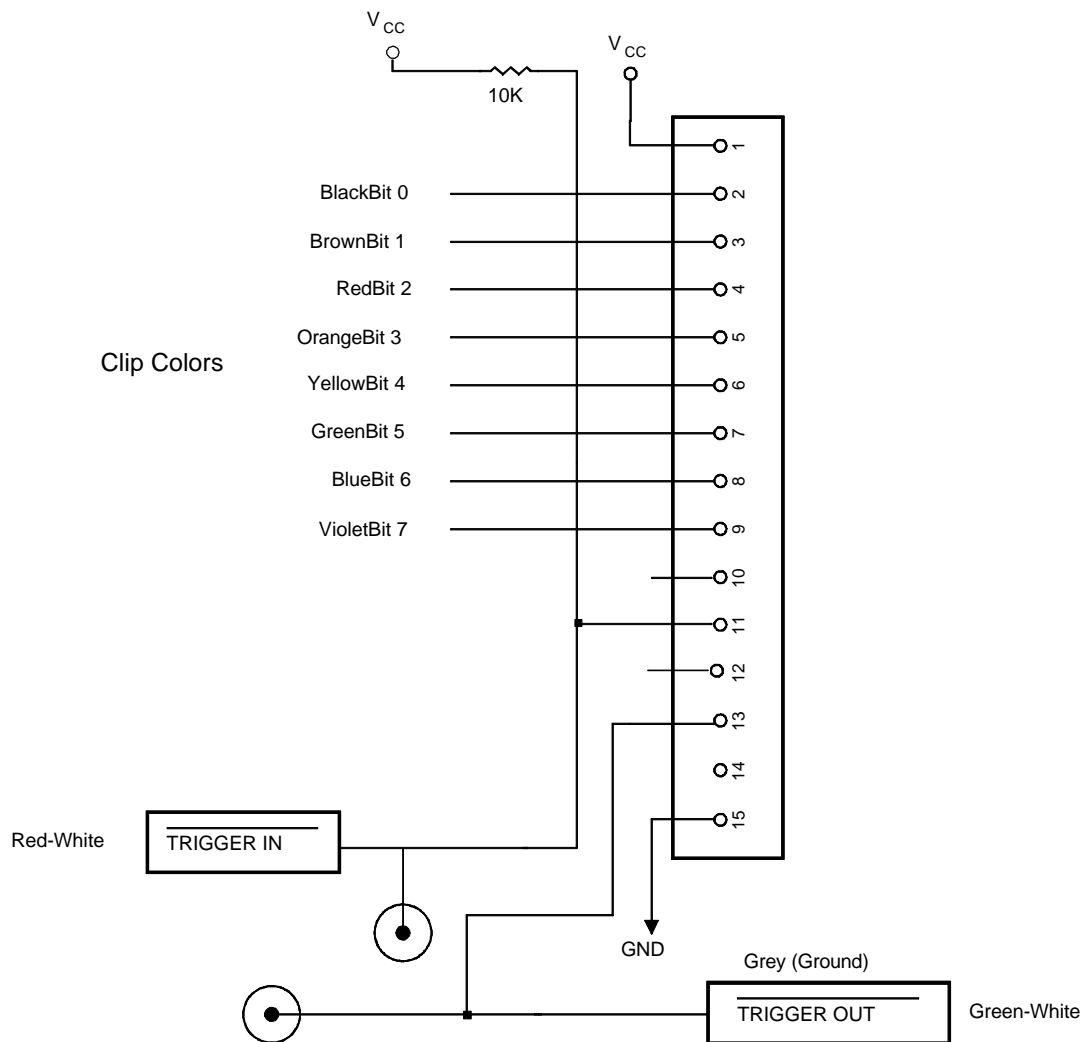
The Trace board records eight external digital inputs with every bus cycle. These signals are input through the 15 pin D connector on the edge of the Trace board.

---

*Note: As external inputs and controls are sampled every frame, you cannot expect higher time resolution than the sample frame rate.*

---





**Figure 42. Trace Board Connectors**

Two of the micro-clips duplicate the trigger controls found in the SMB connectors<sup>2</sup>: TRIGGER IN (J11) and TRIGGER OUT (J10).

Note that the signal voltage levels for TRIGGER IN and TRIGGER OUT are inverted. A transition from 5 Volts to 0 Volts on the TRIGGER OUT micro-clip indicates that a trigger has occurred. The signal is held low until the trace board starts recording again.

The TRIGGER IN micro-clip can prevent triggering when this line is held low. As long as this line is held low, the **Last trig event repeat count** will not count down, the events that satisfy the trigger conditions will not cause a trigger, and trace recording will not stop.

<sup>2</sup> Some trace boards do not have SMB connectors. If your board does not, and you would like them, contact customer support at support@icetech.comm

## ***Introduction to Tracing***

A trace history is a time ordered recording of bus cycles (with some other helpful information). Events that do not affect the CPU external bus, such as testing a CPU internal data register, will not get recorded. Events that do affect the bus will only get recorded if the "recorder" is turned on and set up to record those types of events. All tracing emulators record bus events and not actual instruction execution, so they all must have some way to deal with the effects of the instruction pipeline. The trace board for EMUL296™-PC includes pipeline decoding and marks opcode fetches that are not executed. As a result, the display software can show the trace records as though the pipeline did not exist, but it can also display the uncorrected bus cycles just as they were recorded.

Tracing starts automatically every time emulation starts. Even single stepping will turn on the trace recording during that step. Clicking on the **Trace Beg** button or pressing the **F10** key will also start recording (but until emulation starts, there will be nothing to record). Once trace recording has started, the **Trace Beg** button changes to the **Trace End** button, and will stop recording when clicked. The trace buffer will continue to collect records until recording is stopped, either by a trigger, by stopping emulation, by pressing the **F10** key, or by clicking on the **Trace End** button.

Once emulation has started and bus cycles are "being recorded," every bus cycle is examined to see if it meets the conditions in the **Filter:** field of the **Trace Setup** dialog box. If it does, then it will be recorded. If it does not, that bus cycle will not be recorded in the trace buffer. Bus cycles that are not the correct type (opcode fetch, data read, or data write), or that fall outside the address range(s) specified in the **Filter:** field, will be examined to see if they meet any trigger conditions but will not be added to the buffer.

Every time tracing starts the buffer is cleared. After recording a single step, the trace buffer will only contain the records for that one instruction or source line. Once the buffer is full, the new records will begin to overwrite the oldest records. The trace buffer is a ring buffer that will continue to collect new records and replace old records until recording is stopped. Triggers without an address qualifier will be made inactive.

### **Triggers and Hardware breakpoints**

The trace board can do more than just record what happens on the controller bus. A "trigger" can occur when certain conditions on the bus are met. For example, you can program a trigger to occur when the instruction at 4FE Hex has been fetched for the fourth time. Triggers can start and stop trace buffer recording, and can cause hardware breakpoints. These are useful if you are executing out of ROM or need to break on certain hardware conditions. For information about how to create triggers and hardware breakpoints, see the section titled "Trace Setup Dialog Box" on page 80.

## Trace Window

The contents of the trace buffer are displayed in the **Trace** window. If there is no **Trace** window open, you may open one using the **Window** menu item and selecting **Open a new trace buffer window**. Most of the **Trace** window features are controlled by the trace menu, and are described in the **Trace Menu** section below. Please refer to both this section and the **Trace Menu** section for a complete description of the **Trace** window.

### Pipeline Effects

When a jump occurs and the pipeline is flushed, some instructions are fetched but not executed. These fetched but ignored instructions are captured by the trace board when they are fetched but the display software will not disassemble them.

### Bus Cycle Order

All bus cycles are shown in the order fetched, not in the order executed. The **Trace** Window shows the executed fetches, with the last row possibly showing only fetched and not executed instructions.

### Bus Width

The bus size is dynamic and can be either 8 bits or 16 bits wide, but the trace buffer always records 16 bits of data for the each bus cycle. Whenever the bus is executing an 8 bit bus cycle, the trace board will still record all 16 bits of data on the bus even though the other 8 bits are ignored by the CPU. Normally, when showing corrected bus cycles, the **Trace** window will not display these bytes.

## Trace Menu

Trace	Window
Go to Frame number ..	Ctrl-F
Search Address ..	Ctrl-A
Search Next address	Ctrl-N
Search Previous address	Ctrl-P
Find Trig point	Ctrl-Q
Save trace as text ..	
Save trace image to file	
Show trace image from file	
Compress	
Show rgbic data	
Show TRB, 7 bits	
Show timestamp	
Relative timestamp	
Convert cycles to time	
Synchronize program window	
Trace Setup ..	

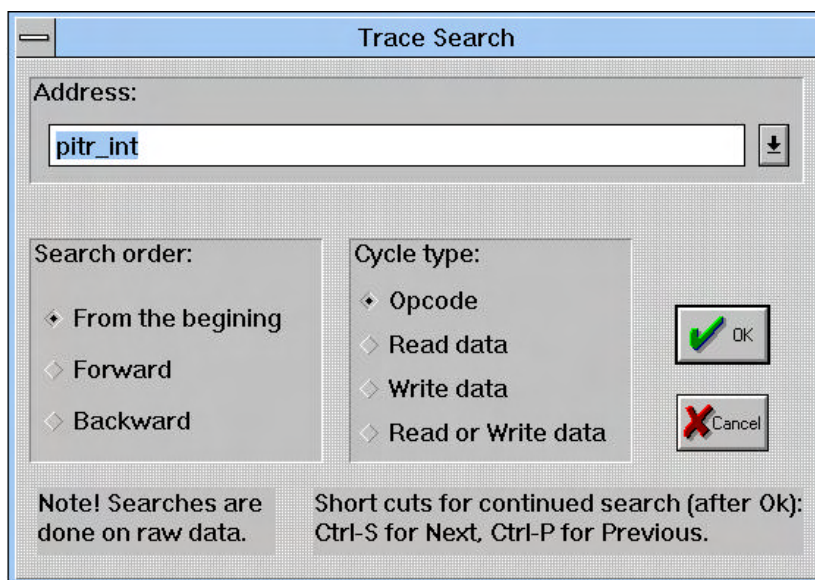
Like the other window-specific menus, the **Trace** menu only appears when the **Trace** window is selected. The **Trace** menu contains items that control how the trace is displayed.

### Find Frame number ..

When this menu item is selected, it opens a dialog box to get the desired frame number. Once the trace buffer has records, this menu item scrolls the trace window to the record entered in the dialog box.

### Search Address ..

This menu item opens a dialog box, shown in Figure 43, to get the desired address, then searches from the beginning in the frame buffer for the first record that contains the specified address.



**Figure 43. Trace Search Dialog Box**

By default, the search includes only opcodes and starts at the first (oldest) frame in the buffer (not necessarily frame 0). By selecting options in the dialog box, you can choose the search direction and limit the search to only certain kinds of bus cycles.

#### Search Next Address

From the current frame, this menu item searches forward for the next occurrence of the last address searched. If a search has not yet been specified, no frame will be found.

#### Search Previous Address

From the current frame, this menu item searches backward for the next occurrence of the last address searched. If a search has not yet been specified, no frame will be found.

#### Find Trig point

This menu item will scroll the **Trace buffer** window to show frame 0, which is the trigger point.

#### Save trace as text ..

With this menu item, you can save any portion of the trace buffer to a text file suitable for inclusion in documents or processing with text manipulation or word processing tools. Selecting this menu item will open a dialog box that lets you set a range of frames and the name of the file where the text goes.

The file text will be formatted in the same manner and with the same options as the text in the **Trace buffer** window. If you want the text file to include timestamps, arrange for the **Trace buffer** window to show them as well.

### Show misc. data

Selecting this menu item will display another 24 bits (shown as 3 bytes in hexadecimal notation) for each record. The trace board records 8 external input bits from the DB15 connector, 8 processor status signals, and 8 signals on the TRACE header on the pod with each trace record.

Of the three bytes, the left most shows the bits input from the DB15 connector. Figure 42 on page 74 shows how the pins of the DB15 are used and which color micro-clip is assigned to which bit in the display.

The other two bytes are sometimes useful to the Customer Support staff but generally not useful to our customers. You may ignore these bits until a Customer Support person asks you to list them.

### Show timestamp

The timestamp is not always displayed. By default, to reduce the size of the **Trace** window, timestamps are not shown. To see the timestamp, select this menu item.

### Benchmarking Using Timestamp

When you have captured a trace and are looking at the timestamp information, keep in mind that the timestamp reflects fetch activity. Therefore, do not try to look at the timestamp for an individual instruction to determine the execution time. For example, if you want to know how long it will take to execute a multiply instruction, type in ten multiply instructions in the **Program** Window. Then, type in a jump instruction to the start of the multiply sequence. Executing this sequence and looking at the trace result will let you determine the execution time by comparing the time the first byte of a multiply instruction was fetched to the time the first byte of the subsequent multiply instruction was fetched.

### Relative timestamp

The timestamp is an integer which is large enough to uniquely number all CLKOUT cycles in a 26 hour period running at 50 MHz. The default display mode for the timestamp is to show the cumulative time since (or before) the trigger. To see the delay between individual instructions or bus cycles, select this menu item.

This display mode is also useful for timing segments of code. In the example below, only the bus cycles at addresses 400-410 were recorded. You can see that each timestamp is shown relative to the timestamp of the next frame in the buffer. Note that frame -1 is not 743 milliseconds long. Rather, it was 743 milliseconds between the end of the bus cycle that fetched the first word of frame -1 and the end of the bus cycle that fetched the first word of frame 0. It took 743 to finish fetching the operands for frame -1, (execute any other instructions in the loop,) and to fetch the first word (4E56) of frame 0.

Also note that the timestamp for the first frame shown, frame -9, shows the time since the very first frame in the buffer, not the time to the previous frame. This is why the timestamp show 748 milliseconds, not 742 milliseconds, as is shown for the other times through the loop.

T = 0 at Cursor

The software has the ability to force the timestamp value to zero for a certain frame, giving the user an easy way to measure execution time (<CTRL>Z).

frame#	address	timestamp	data
-9,	400:	748.945312 ms	4e56ffe4
-7,	404:	750 ns	48ef0c0c0004
-4,	40A:	2.250 us	45f900000560
-1,	410:	742.983313 ms	47f90000000f
0,	400:	188 ns	4e56ffe4
2,	404:	750 ns	48ef0c0c0004
5,	40A:	2.250 us	45f900000560
8,	410:	742.983313 ms	47f90000000f
9,	400:	188 ns	4e56ffe4

Figure 44. Example of relative timestamp displayed in seconds

Convert cycles to time

The actual timestamp in the trace record is a count of clock cycles. When this menu item is checked, the timestamp is displayed in seconds (or fractions thereof) as shown in Figure 44. It uses the value in the **Clock** field of the **Trace Setup** dialog box to convert the cycle count to time. If the value in the Clock field is incorrect, these timestamps will be incorrect also. When unchecked, the **Trace** window displays the timestamp in clock cycles.

---

*Note: If the application changes the clock frequency dynamically, the timestamp displayed in seconds may or may not be correct. The timestamp displayed in clock cycles will always be correct no matter what the clock speed.*

---

Synchronize program window

When this menu item is checked, as you move the cursor around the **Trace** window from opcode cycle to opcode cycle, the cursors in the **Program** and **Source** windows will also move to point to the instruction fetched and it's context. If the application is running, only the **Source** window will scroll.

Trace setup ..

Selecting this menu item, just like selecting the **Trace ..** menu item in the **Config** menu, opens the **Trace Setup** dialog box. The details of that dialog box are described in the following paragraphs.

### Toggle trace (stop/run)

If the trace board is recording cycles, this menu item will turn off cycle recording. Conversely, selecting this menu item before the trace board has started or after a trigger has occurred will turn on recording, just like clicking on the **Trace Beg** button.

## **Trace Setup Dialog Box**

### Board Installed

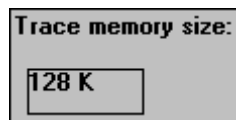
The box next to **Yes** must be marked before the trace board will be used. If this box is marked and the board is not there or the starting I/O address is not set correctly when the application is started, the **Trace Setup** dialog box will be opened automatically. If the board is installed and the **No** box is checked, the application will probably not execute normally. It might only single-step, and not run at full speed.

### Address

The **I/O Address** field is the field that identifies the start of the emulator board I/O addresses. This value must agree with the setting of the trace board header jumpers. For setting the trace board address header jumpers, see the installation section starting on page 73.

### Trace Memory

This box, shown in Figure 45, displays the number of records that can be stored in the trace board. This field is set whenever the emulator and trace board are reset.



**Figure 45. Size of Trace Memory**

### Triggers

A trigger is an event that occurs once for each time the trace recording is started. There are two ways to set up triggers and bus cycle filtering: **Normal** mode and **Window** mode. In **Normal** mode, more control is given to triggers. A trigger in **Normal** mode either stops recording or starts the countdown until recording will be stopped and can cause a hardware break. (Frame 0 is always the frame where the trigger occurred.) In **Window** filtering mode, more control is given to controlling which bus cycles are recorded. Each mode is described in more detail below.

The field labeled **Post trigger samples** contains the number of frames to be recorded after the trigger occurs. Once the trigger occurs, recording continues until the number of samples recorded is equal to the number in this field. If it is set to 0, no cycles will be recorded after the trigger occurs. If it is set to 10, then 10 cycles will be recorded after the

trigger occurs. If it is set to the total buffer size, then frame 0 will always be the first frame in the buffer.

---

*Note: If the trace board is configured to break execution when the trigger occurs, the **Post trigger samples** field is not used because recording will stop when execution stops. If the **on trace stop** box is checked, then this field controls when the break will occur.*

---

The **Last trigger repeat count** field contains the number of times the highest numbered trigger conditions must be met before the trigger itself occurs. If a trigger condition is set for an opcode fetch at address 400 and the **Last trigger repeat count** is set to 10, the first 9 fetches from address 400 will be counted and the trigger will occur when that opcode is fetched for the 10th time.

This count may be as high as 256, and applies to the last (highest numbered) trigger event that has conditions in **Normal** filter mode. If **Trig event3** has no conditions, it will be ignored and the conditions in **Trig event2** will be counted. If neither event 2 nor event 3 have any conditions, the **Trig event1** events will be counted.



Filter Mode: Normal

Using the **Normal** filter mode, you can set up 3 trigger conditions. Each trigger condition is a series of statements that are OR'ed together logically. Each statement contains one address range and one type of bus cycle. Approximately 2000 statements may be used in each trigger condition. The three triggers conditions are tested sequentially: once the conditions for trigger 1 are met, the conditions for trigger 2 (if present) are tested. If there are no trigger 2 conditions, then all the conditions are satisfied and the trigger occurs. and likewise for trigger 3. See the section starting on page 95 for information about setting and changing the trigger conditions.

Figure 46 shows that trigger 1 has one condition and it will be met by "any" kind of bus cycle that includes any address from 0 to 7FFFF.

Qualifiers: Click under 'Addresses' or 'Data' below to review a qualifier list:

	Active?	Addresses	Addr mask	Data	Data mask
Trig event 1	<input checked="" type="radio"/> Yes <input type="radio"/> No	<input checked="" type="radio"/>	*****	<input type="radio"/>	FFFF
Trig event 2	<input type="radio"/> Yes <input checked="" type="radio"/> No	<input type="radio"/>	*****	<input type="radio"/>	FFFF
Trig event 3	<input type="radio"/> Yes <input checked="" type="radio"/> No	<input type="radio"/>	*****	<input type="radio"/>	FFFF
Filter:	<input type="radio"/> Yes <input checked="" type="radio"/> No	<input type="radio"/>	*****	<input type="radio"/>	FFFF

"Address" qualifier list for "Trig event 1". ☐ Extend recording

include all: 0 7FFFF (0-7FFFF)	<input type="button" value="New"/>
	<input type="button" value="Edit"/>
	<input type="button" value="Delete"/>

**Figure 46. Trigger on anything**

With **Filter mode: Normal**, bus cycles are recorded until a trigger stops the recording. Recording bus cycles is a separate activity from deciding to trigger or not, and so it has a separate set of conditions. The **Filter:** field can contain up to 2000 statements that are logically OR'ed to decide whether to record the bus cycle or not. In Figure 47, the trace board will record everything from the start of main to `MAIN + 100H`, the opcode fetches between line 93 and 108 from the module `TIME`, and only the data bus cycles (read and write cycles) to and from address `timer.sec` (1338H).

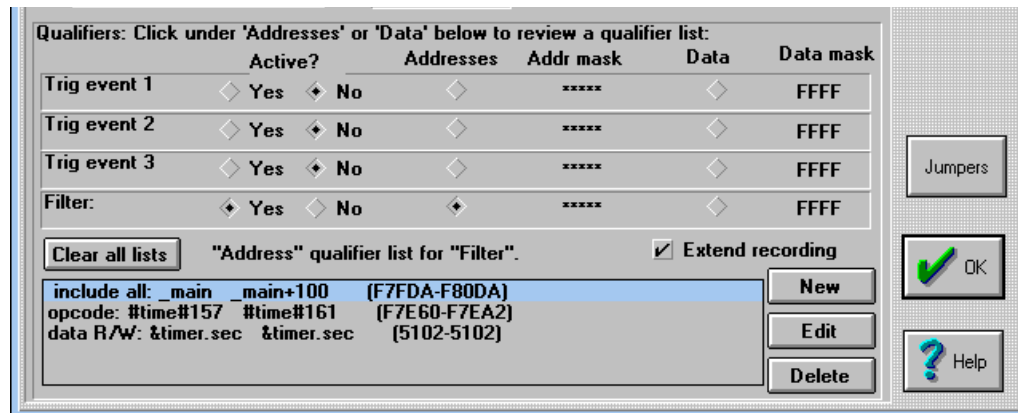


Figure 47. Selective Recording

---

*Note: Trigger events must be sequential. Also, the Address and Data mask are ANDed with the entered hex address or data value.*

---

### Extend Recording

Also shown in Figure 47 is a button with the label **Extend recording**. **Extended recording** means continuing to record 3, 4, or 5 records after the filter condition has turned off recording. Every time the filter allows a record to be captured, a counter is set. If the next instruction does not pass through the filter, the record is captured anyway and the counter decrements. The same is true for the next few instructions, until the counter reaches 0. Then, if a record is filtered out, it is not captured. This behavior will be easier to understand when you see where it is useful.

To see how **Extend recording** may be useful, let us say that some part of the application is calling the function that ends with line 161 in the module TIME. Line 161 is the closing brace in the C source file from the example in Figure 47. How do you find out what part of the code (possibly in assembly language) is calling that function?

One way is to record everything and then break on line 161 with a small post trigger sample count. Line 161 will be executed, the trigger will occur, then a few frames will be captured from after line 161, showing where the program counter went when returning from the function call. Out of 128K records, only one frame will contain the information you are looking for. If that is not the call you are interested in, you can set the Last trigger repeat count field but you won't know whether the function call you want to capture will be next or the ten thousandth call.

It will be far more efficient to use the emulators resources to selectively capture only those records that are likely to have the information you want. With **Extend recording**, and with the filter set as shown in Figure 47, every time the trace board captures an opcode fetch from line 161, it will also capture the next 3, 4, or 5 bus cycles, and those bus cycles will tell you where the program counter went after line 161. When you use **Extend recording**, the trace board may contain up to 32K instances of where line 161 was executed and where the program counter went afterwards.

### Filter Mode: Window

Using the **Window** filtering mode gives a different kind of control over what cycles are recorded, and can selectively record program threads in a way that record filtering cannot. Like trigger conditions, the **Enable recording** and **Disable recording** conditions are set using the editor described below, and each condition is logically OR'ed with all other conditions to find a match. Bus cycle recording will start when the **Enable recording** conditions are met, and will stop when the **Disable recording** conditions are met. Once recording has started, the conditions in the **Filter** field decide whether to record a bus cycle or not.

**Figure 48: Filter Mode Window**

### Editing the Trigger Conditions

To set up the trigger conditions, click on one of the triggers (or recording controls), click on the statement you wish to change, then click on the **Edit** button. This will open an **Address qualifier** window like the one in Figure 49. The **Start:** field will be selected. Type the address range start (either a hexadecimal number or a symbol name), hit <TAB> to get to the **End:** field (or click on it), type the upper limit of the address range, and then click on one of the types of cycles. Figure 49 shows how to monitor a single address: put that address in both the **Start:** and **End:** range fields.

**Figure 49: Data mode = Opcode**

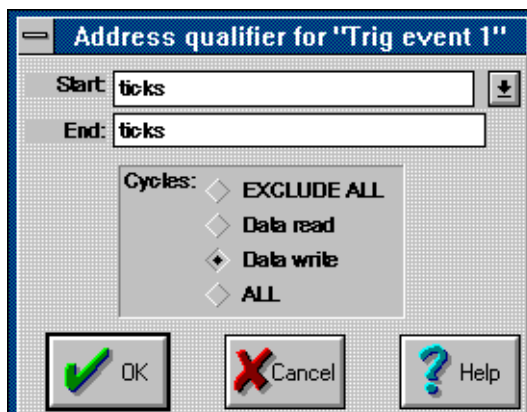
By default the **Data** mode field has the **Opcode** box marked. Marking the **Data** box gives you the options shown in Figure 50.

When addressing 16 bit wide memory, triggering on a single byte WRITE to an odd address occurs on the high byte on the data bus. To set up a trigger for this condition, set the mask to FF00 and data qualifier in high byte.

---

*Note: Future versions of the software will make this feature more intuitive.*

---



**Figure 50: Data mode = Data**

---

*Note: Changing the **Data** mode for one trigger also changes the **Data** mode for all other triggers. After changing the **Data** mode, review all the conditions for all triggers to make sure they are still correct.*

---

When this dialog box contains the desired address range and the **Cycles:** field has the correct button selected, either press the <Enter> key or click on the **OK** button. Each trigger event can have approximately 2000 conditions.

#### Break Emulation? Box

A trigger can cause a hardware break either when the trigger occurs or when recording stops, after the **Post trigger samples** frames have been recorded. Mark either box (or both). This feature means you can set breakpoints "on-the-fly".



**Figure 51: Turning on Hardware Breakpoints**

---

*Note: There may be a delay of up to 3 instructions between the trigger that causes break and when the break actually occurs.*

---



## Chapter 5: Pod Boards

### **Features Common to All Pods**

Every pod is a fully functional, stand-alone 8xC296 board, with a processor, RAM, a crystal, PROM, and logic to glue all those pieces together.

#### How It Works

Clicking on the **Reset** button tells the emulator to pull the RST line low, resetting the controller. When the RST line is released, the controller begins by executing instructions that allow the emulator board to communicate with the pod. These instructions are the monitor code. The controller will continue to execute monitor code until you click on the **Step** button, the **GO** button, or select **Reset and go** from the **Run** menu.

When you click on the **Break** button, a specific kind of non-maskable interrupt occurs, the return address is pushed on the stack, the program counter is loaded with the corresponding vector, and it continues to run at the new address. The new address is mapped to the pod. That memory contains the monitor code.

When sections of memory are displayed on your screen, it is the controller that actually reads the memory locations and sends the values back to the emulator board in your PC. If the emulator cannot read the contents of memory, then your application will not be able to either.

*Note: If you are running user code, target power can be turned off/on to emulate power on if reset is held low during power off. Some emulation chips will require a special power board.*

---

#### Stack Pointer

Because the emulator pushes the return address on the stack, the Stack Pointer must point to valid memory and there must be room on the stack for 2 bytes (or 4 bytes for users of chips with larger addressable ranges) to hold the address.

In addition, there is a lower limit to the Stack Pointer. If the Stack Pointer has a value of 0x50 or less, it interferes with emulation. This is not usually a problem but we want you to know about it.

---

#### Indicator Lights

The pod boards contain 4 lights. They are labeled HALT, RESET, RUN, and USER. The HALT light is lit whenever the P2.5 (HOLD#) goes low. The RESET light will only be lit when the emulator resets the controller. The RUN light will be lit whenever the controller is executing user code (as opposed to monitor code). You may use the USER light to indicate

the state of any signal on the pod or target by connecting a wire from the desired signal to the test point labeled TP1. It will be lit when the test point is brought low.

*Note: If using the HLD pin as low speed I/O, disregard the light.*

#### How to Break Two Emulators Simultaneously

At the edge of the pod board there are two test points called BRK\_IN and BRK\_OUT. The BRK\_OUT test point will show logic low when the user code stops. The BRK\_IN test point, if forced to logic low, will make the user code stop. With two emulator systems, you can connect BRK\_OUT from one pod to BRK\_IN on the other pod to make the two emulator systems stop user code execution simultaneously.

*Note: We can provide a DOS program to start two emulators simultaneously;  $t$  = start of first emulator to start of second emulator;  $t < 50$  ns.*

#### Trace Input Pins

Next to the indicator lights and the test point is an array of 8 pins, labeled TRACE. These pins may be connected to any logic signal and will record the state of that signal with every trace record. Pins 0 through 3 are sampled on the falling edge of ALE, with the address. Pins 4 through 7 are sampled on the rising edge of the RD/WR strobes, with the data. For more information about displaying these bits and Trig-In/Trig-Out, please see the Trace Board chapter.

#### Duplicate Resources

The pod board has many resources and your target may also have the same resources. If the same resource appears on both the target and the pod board, you will have to choose to remove or disable either the target or the pod resource for all the resources that appear on both (see below).

When the pod is connected to a target that has no power supply the pod can supply +5 Volts to the target limited by your PC supply capacity and target's sensitivity to under-voltage. If the target has its own power supply, remove the jumper on the PWR header. If you do not, it is possible to damage either the target power supply or the power supply in your PC.

If your target has a crystal operating at a speed different than the frequency on your pod, you may wish to use it instead of the crystal on the pod. To use the target crystal, find the two headers labeled TARGET/POD near the pod crystal and place the two jumpers so that they are on the Target side, not the Pod side. This will disconnect the pod crystal from the controller on the pod and allow the pod controller to use the crystal on the target.

EMUL296-PC uses a special emulation controller to emulate the 80C296. This special chip has extra pins that give the emulator extra features. The special emulation controller can map memory, halt execution, set breakpoints, etc. This is why your program must execute in the controller on the pod and not in the controller on your target board.

Most adapters fit between the pod and the target board, replacing the target controller. In summary:

RESOURCE:	WHAT TO DO WHEN THE TARGET HAS IT:
RAM	Map the RAM to the Target by using CS0..5
Crystal	Move JP7 and JP10 to "TARGET" side of header
Serial Port	Do not use J1, remove RXD jumper.
Power Supply	Remove the jumper from the PWR header

The black wire with the micro-clip is a ground wire, which is helpful for ensuring that the pod and target grounds are at the same potential. We recommend you attach this clip to a grounded point on your target before attaching the pod to the target.

### Configuration Requirements

The emulator is designed to be as transparent as possible to the target and the target application. There is a short list of things that the emulator requires of the target. Those are described here.

The Intel manuals say that address 18H is reserved for the Stack Pointer. However, when fetching instructions, a fetch from that address will get the instruction from an external memory device. On the pod, that address contains the value 0. If you map address 18H to the target, your target ROM/RAM must also contain a 0.

The emulator requires enough memory to push a return address onto the stack. If the stack is pointing to memory that doesn't exist or is pointing to address 0, the emulator will be unable to reach it's monitor code and communications with the emulator will fail.

### ***Do you have enough emulator memory?***

A POD-296-256-xx only has 256K of breakpoint memory in parallel with 256K of emulation memory. That means that you only have four pages to use. If you have pages that overlap because of this, you should order a 1M pod. For instance, if you access to a physical memory at address 5000H, it will also show on three other pages: 45000H, 85000H and C5000H. The emulator reads them from page zero.

### Internal Addressing or Single Chip Mode

*Note: This section pertains only to pods that emulate controllers that support single-chip operation, unlike the current POD-296SA.*

Target designs that use only internal RAM and ROM may use the address and data bus pins for low speed I/O. This is called either single-chip mode or Internal Addressing mode. Pulling the EA pin high during RESET will configure the 8xC296 for internal addressing, freeing the address and data bus pins for general purpose I/O. Even when in single-chip mode, the pod still uses emulation RAM as a substitute for internal RAM and ROM in the target controller, which requires the same pins being used for I/O on the target. In fact, unlike a normal 8xC296, the address, data, and bus control pins on the special emulation controller cannot be used for low speed I/O. This creates a conflict between single-chip



target applications and emulation. The solution to this conflict is a Port Replacement Unit that reconstructs the low speed I/O ports for the target. (External only controllers do not need a port replacement unit.)

## Chapter 6: POD-296-256-SA-50

### Introduction

**Warning:** Do not install more than one jumper on EA16 (JP6). If you do, you are likely to damage the target, the pod, or both.

This pod board contains an Intel 80C296 special emulation microcontroller chip suitable for emulating the Intel 8xC296SA, an oscillator operating at either 16, 20, or 25 (50) MHz, 256 kilobytes of emulation RAM for instructions and/or data, circuits for driving the bus, two PROMs, and two large FPGA chips.

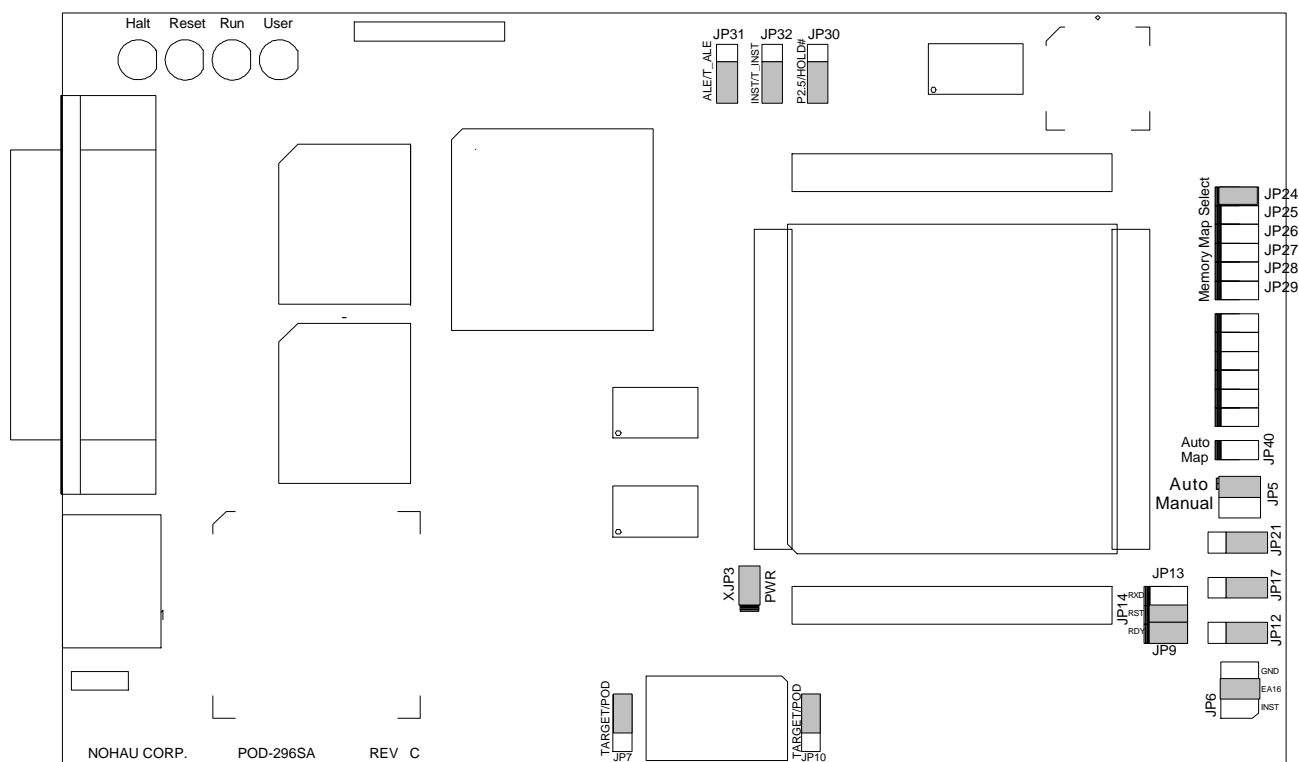
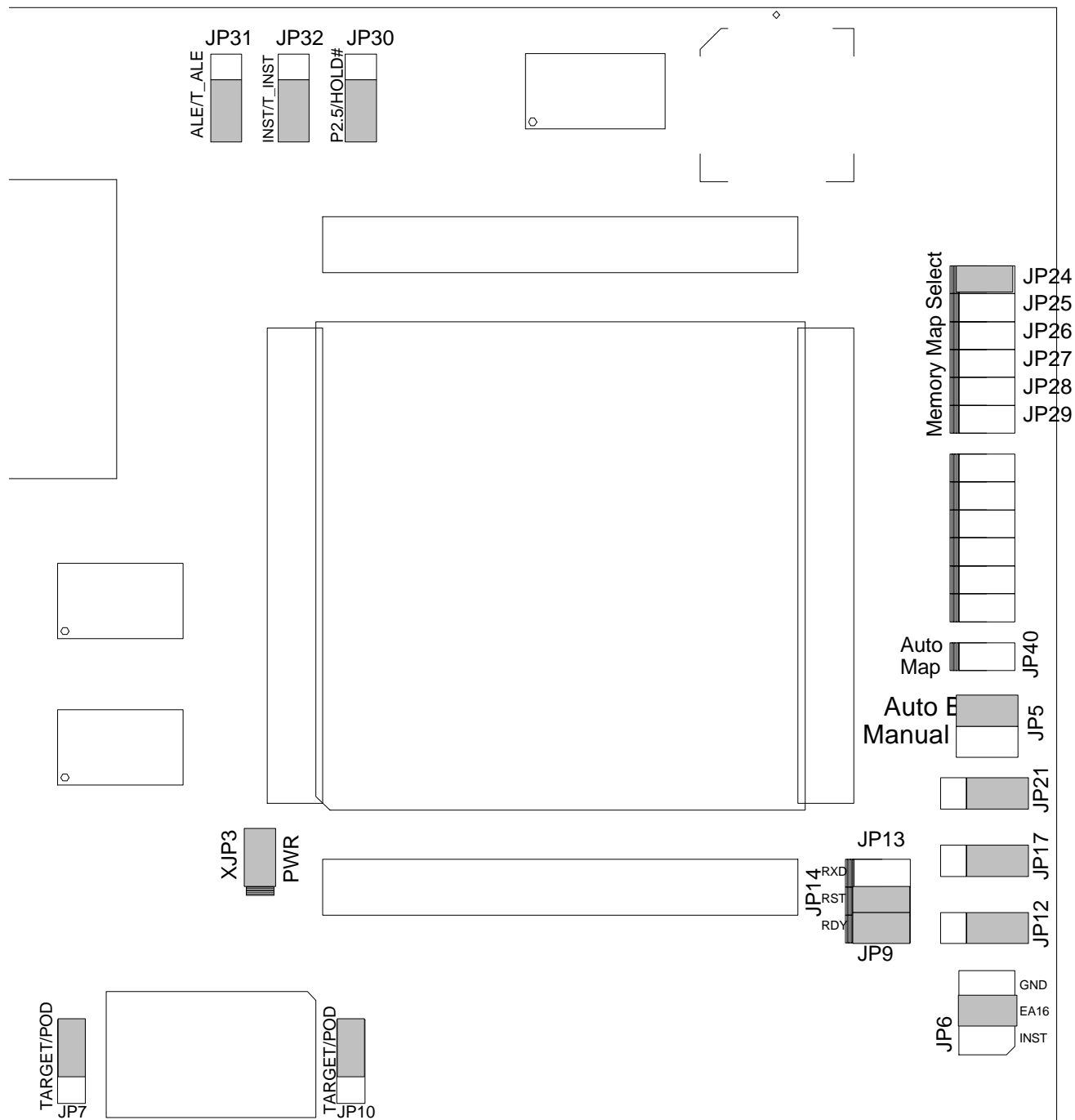


Figure 52: POD-296-256-SA

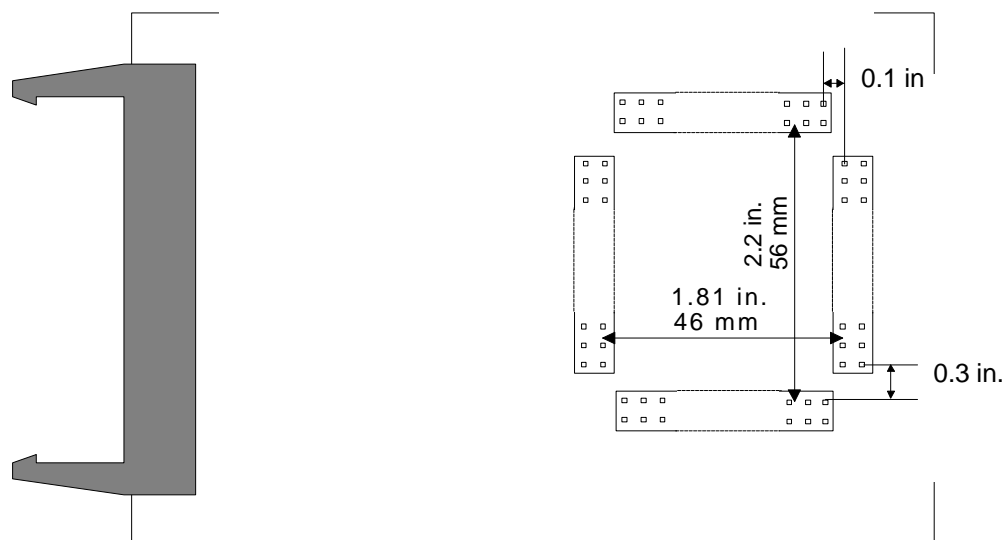


**Figure 53. Enlargement of POD-296-256-SA Jumper Configuration**

### Dimensions

The pod board itself is 6.5 inches by 4 inches (16.6 cm. by 10.3 cm). The pod requires between one and two inches (2.5 cm to 5 cm) of space above the target, depending upon which adapter is being used to connect the pod to the target.

POD-296-256-SA The footprint dimensions for the pin pattern are shown in Figure 54. This information enables the design of a board with a matching pin pattern, eliminating the need for an adapter.



**Figure 54. Footprint Dimensions**

#### POD-296-256-SA Emulation Memory

This pod comes with 256K bytes of high speed static RAM for emulating ROM or target RAM. Controllers like the 8xC296SA, with 20 address bits, can address 1 megabyte. Support has information about ordering a 1 megabyte pod. (See “Do You Have Enough Emulator Memory?” on page 89).

#### Wait States

The emulator uses the number of wait states specified in the **Emulator Hardware Config** dialog box. (or found in the CCBs. In addition, you may use the READY pin to increase the number of wait states to any number. If the target board continuously holds the READY pin low, the application will stop executing and the emulator may display one of several error messages. (An oscilloscope trace of the READ or WRITE strobe will show the strobe signal stuck low.) If the emulator hangs in this way, remove the READY jumper to isolate the target READY signal from the emulator READY pin until your target works.

*Note: Every time you have the emulator reset the controller, the emulator software writes \$0000 to addresses \$1F40 and \$1F42. This feature uses chip select 0 to activate emulation RAM throughout the entire address range and allows you to load code. Typically your start-up code will reprogram the chip select registers and your application will then run normally.*

### Breakpoints

All breakpoints (hardware and software) should be enabled and disabled by clicking on the line number in either the source window or the program window. If you use the hardware breakpoint dialog box, make sure you put the breakpoint at the first byte of instruction. Hardware breakpoints on protected instructions<sup>3</sup> will lock up the 80C296SA emulation chip..

### POD-296-256-SA Headers

In Figure 52, all the headers are shown with their jumpers in the factory default positions. Figure 55 shows some of these headers in detail.

When shipped from the factory, all headers have jumpers located for stand-alone operation (without a target). When you do connect this pod to a target, be sure to examine all jumpers and make sure that they are all correctly placed. Use the descriptions below as a guide to jumper placement.

#### ***POD-296-256-SA Clock Headers: JP7 & JP10***

These jumpers are located on either side of the oscillator on the pod. They are used to select whether the 80C296SA special emulation chip will use the clock from the target or from the pod. Both jumpers should be moved to the same position (TARGET or POD). If the target has an oscillator at a different frequency than the one on the pod, it is recommended that these jumpers be in the TARGET locations.

#### ***POD-296-256-SA PWR Header: XJP3***

This jumper should be removed when connected to the target. It is intended to supply power to the special emulation chip when running the pod without a target. The pod can supply up to 100 mA to the target but it is recommended that this jumper be removed when the target has its own power supply or when the power requirements of the target exceed 0.1 amps.

#### ***POD-296-256-SA RXD Header: JP13***

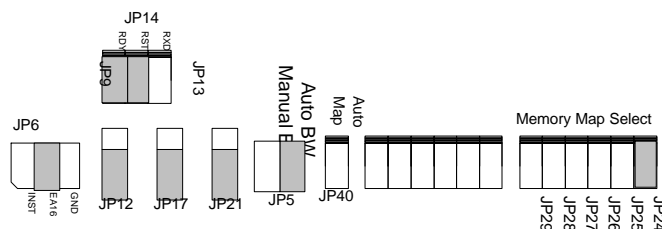
If your target outputs debugging information on the serial port, you may wish to connect an RS232 device like a terminal or a PC to header J1. This pod includes a MAX232 chip to convert the signal levels from RS232 to TTL levels. Whether or not you connect the RxD pin on J1 to an RS232 device, the MAX232 chip will drive the serial port input pin on the controller. To keep the MAX232 chip from driving the serial input pin on the controller, remove the jumper on the RXD header. To allow the MAX232 chip to drive the serial port input pin, place a jumper on the header.

---

<sup>3</sup> The following are considered to be protected instructions: PUSHF, POPF, PUSHA, POPA, DI, DI, SIGN (0xFE), and the first instruction of an interrupt service routine.

***POD-296-256-SA RST Header: JP14***

Occasionally, a target may contain an external device designed to reset the controller by pulling the RST pin low. During debugging, that may be inconvenient. The signal from the target RST pin passes through the RST header. Removing the RST jumper will prevent the external device from resetting the pod controller.



**Figure 55: Pod Configuration Headers**

***POD-296-256-SA Auto/Manual BW Header - JP5***

This jumper should remain in the default (AUTO-BW) position. (This jumper was previously used for the 80C296 NU emulation chip.)

***POD-296-256-SA Auto Map Header: JP40***

This jumper, when in place, currently maps all memory to the pod. This is sometimes convenient when running without a target. Normally, this jumper should be removed so that all memory mapping is done using the chip selects. On previous pods (196NP/NU) this jumper would allow mapping to be done with software. Because of the larger addressability of the 296SA and the high speeds, software mapping is not an option. All mapping should be done with this jumper removed and corresponding jumpers should be placed on JP24-JP29. Installing JP40 will map all memory to the pod.

***POD-296-256-SA JP6, JP12, JP17, & JP21 (EA16-EA19) Headers***

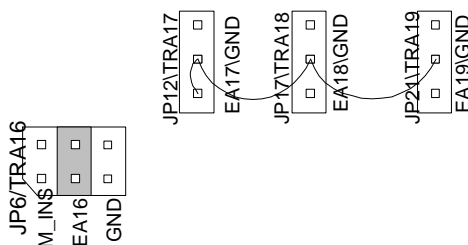
Because the upper address lines (A16-A19) can be used either as I/O pins or as address lines, we need to determine whether or not these pins should be directed to pod memory and whether or not they should be sampled in the trace. If any of these pins are not configured as extended address lines, the corresponding jumper should be placed in the GND position. All four jumpers should reflect the configuration of the EPORT registers. It is also important that the **Hardware Config** Menu match these jumper settings or the software could hang. JP6 also has a third position labeled INST. This is not supported and there should never be a jumper in this position.

***Symbols in the Trace Window***

Right out of reset, the 80C296SA looks for the start-up code and CCB values starting at FF 2018. Many applications will compile and link code (and all code symbols) to page FF 0000 and up. If that application also maps global variables to address 0 and then uses some of

the higher address pins for low speed I/O, the trace disassembly will be unable to associate the trace buffer addresses to the correct code symbols. (Some of the EA1x jumpers will need to be in the GND position.) If this is true for your application, there is a work-around you may want to consider.

Under these circumstances, to correctly associate addresses with symbols, the trace board needs to receive an address that is different than the one appearing on the address pins. If you run a wire from the EA1x side of the highest TRA1x header not carrying an I/O signal to the center pins of the higher address headers, the trace board will get correct address for code space and will likely still get correct addresses for data space bus cycles. A picture and an example will make it much more clear.



**Figure 56: Wiring for 256k by 8 RAM chips**

The application in Figure 56 uses the two highest address pins for low speed I/O. The 256k by 8 RAM chips for holding data need 18 address bits: bit 0 through bit 17. Again, the instructions are mapped to the top of the address range: from FF 0000 to FF FFFF hex. This wiring ensures that when address pin 17 is high, the trace board will receive high signals for TRA17, TRA18, and TRA19. If this example application has global data symbols between 20000 hex to 40000 hex, they will not be identified correctly in the Trace window. This wiring will have no effect on how the trace displays global symbols below 20000 hex or local variables found on the stack.

#### ***POD-296-256-SA JP30-JP32 (P2.5, ALE, INST) Header***

These jumpers should all be moved together depending upon how P2.5 is configured. If P2.5 is configured as standard I/O, these jumpers should be in default position 1&2 (P2.5,Inst,Ale). However, if P2.5 is configured as HOLD#, these jumpers should be in position 2&3 (HOLD,T\_Inst,T\_Ale) -- (used to tristate these pins during hold).

#### ***POD-296-256-SA RDY Header: JP9***

Similar to the RST jumper, the RDY jumper passes the READY signal from the target to the controller. Since the controller is also responsible for communicating information to the Windows software, if Ready suspends the operation of the controller, it will also suspend the communication between the pod and the Windows software. To get past this problem, the RDY jumper may be removed to continue operation of the controller if a problem should occur.

#### ***POD-296-256-SA Memory Mapping Headers: JP24 - JP29***

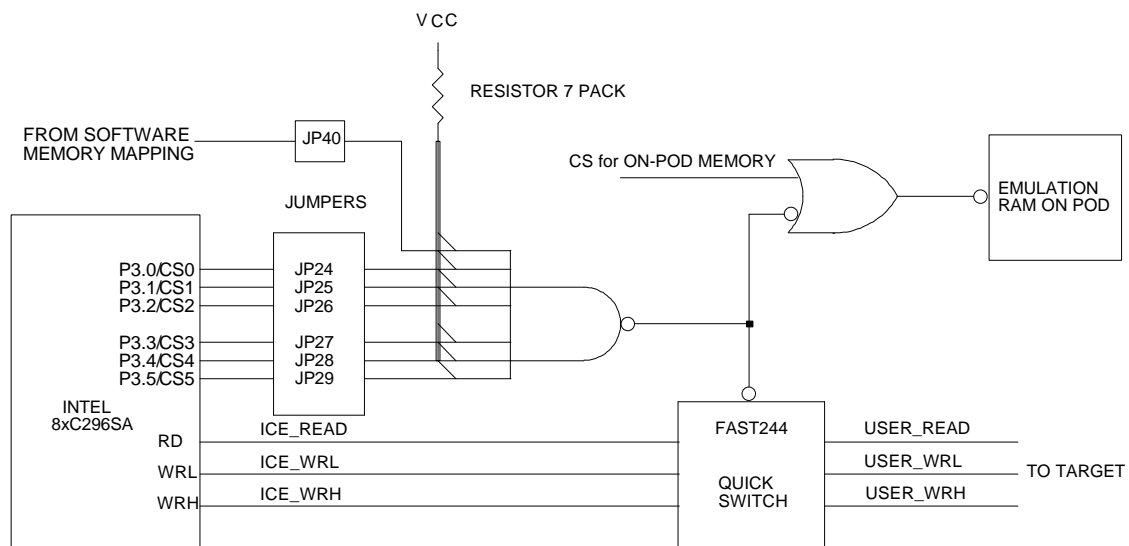
A jumper block on one or more of these headers will pass the corresponding chip select signal to a PAL to create our mapping signal. One side of the header is connected to a

chip select signal. The other side has a pull-up resistor and is fed to the PAL. The PAL ANDs all of these signals together such that a logic zero on any input will map that cycle to the pod. Thus, for each chip select that has a jumper block installed, the address range of that chip select will be mapped to the pod. With no jumper blocks installed, all memory access will be directed to the target. A typical configuration would be to have a jumper on CS0 (JP24) only so that the code is mapped to the pod memory and all other memory access will be directed to the target. This should be in addition to JP40 being removed.

*Note: Software Memory mapping is NOT supported by the 296SA. The memory mapping dialog box should not be used.*

### Mapping Memory Using Chip Selects

While debugging your hardware and software, you typically want to use the RAM on your target for data and replace your EPROM with emulation RAM so you can reload and run your application quickly.



**Figure 57: Schematic for Memory Mapping**

The essence to using chip selects to map memory is to map all addresses to the target and then use a chip select signal (or your target PAL output) to override the software mapping and re-map an address range back to the emulation memory on the pod. This signal can be either a chip select signal from the 8xC296SA controller, or it can be the output from some address decoding logic.

To use a chip select signal, place a jumper on the corresponding header. JP24 through JP29 pass CS0 through CS5 respectively. When any jumpered chip select signal is active (low) bus cycles will be directed to the pod.

To use the output from a PAL on your target, run a wire from the PAL to the JP24 header, to the pin closest to the edge of the pod. When that pin is pulled low by the PAL, bus cycles will be directed to the pod.



*Note: The read-strobe and write-strobe signals are gated so that there can never be a bus collision between emulation RAM and target memory devices.*

*Note: Inserting jumper JP40 will map all memory to the pod for running the emulator without a target.*

**Warning:**    *Mapping all RAM addresses to a fully functioning target will almost never cause any new problems. But, the emulator cannot function normally when stack RAM addresses are mapped to non-functioning RAM.*

*Note: When using an EMUL296 trace board, the trace port address should match the emulator port address in the EMUL296.INI file.*

*Note: Because of a limitation in the debug capabilities of the current 80C296SA emulation chip, we will not support non-prioritized mode (PRIOR\_EN = 0). Normal operation requires that the user's code set the PRIOR\_EN bit in the NMI\_PEND register and that no interrupts are allowed prior to this occurring. If this is not done, the emulator will not be able to run monitor code without being interrupted, which will result in communication errors.*

## Chapter 7: Accessories

### Overview

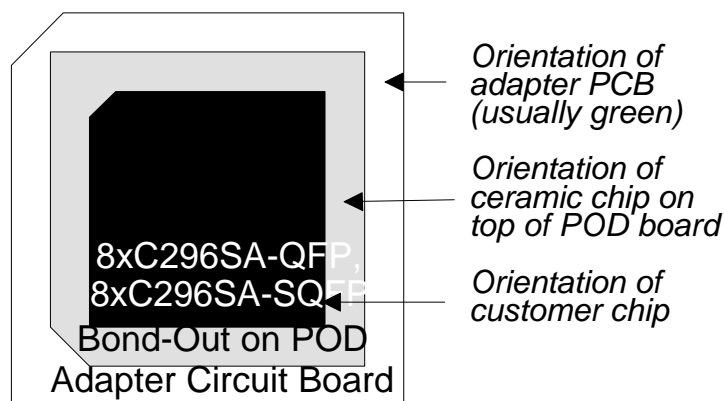
To support EMUL296™-PC, we provide hardware accessories and supports several compilers. Most of the hardware accessories are adapters for connecting the various pods to a wide variety of targets. The compilers described at the end of this chapter are for the microcontrollers with both 16 bit and 20 bit addressing. This list of accessories was complete as of the publication date, but new accessories and support for new compilers are added routinely. Contact us for current prices and a complete list of adapters and compilers for EMUL296™-PC.

This chapter contains descriptions of adapters for PLCC parts, pin grid array parts, and clips for surface mount applications. After that the Port Replacement Units are described. Following those are the miscellaneous hardware accessories that make the adapters even more adaptable to unusual target configurations. Finally, the compilers are described in general terms; outlining issues pertinent to the EMUL296™-PC debugging environment.

The most effective way to connect the pod to the target is to design headers on your target that match the pins on the underside of the pod. Doing this will avoid the cost, capacitance, and extra contacts associated with each kind of adapter. If you have the space on your target board, we highly recommend adding these sockets to your target. If your target does not have room, all of the following adapters are equally effective and easy to use.

### Surface-mount QFP adapters - SA Family

The following diagrams show the proper adapter orientation for the microcontrollers listed.



**Figure 58: QFP/SQFP Adapter Orientation**

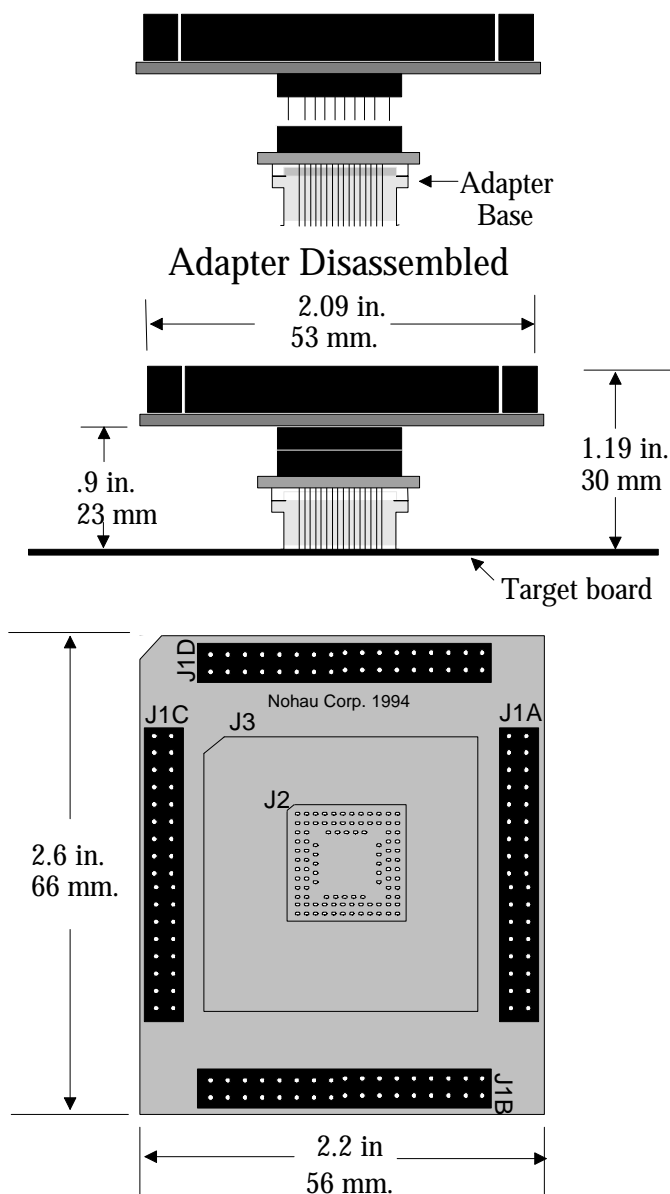
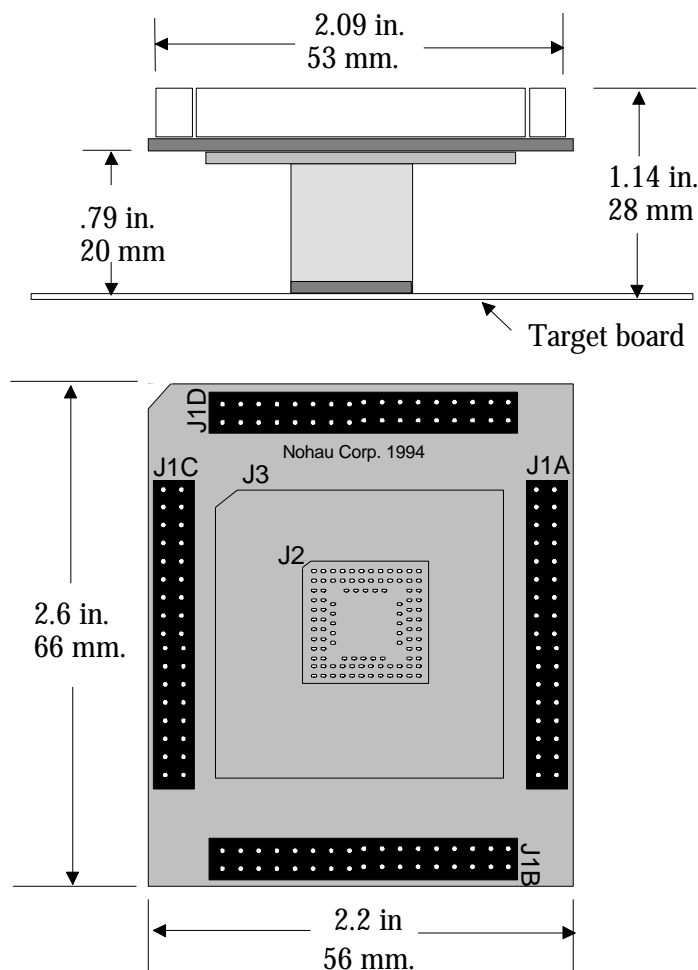


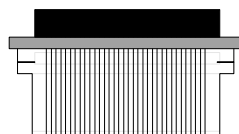
Figure 59: Dimensions for EMUL296-PC/SA-ADP100-QFP/ET



**Figure 60: Dimensions for EMUL296-PC/SA-ADP100-QFP/EDI**

The 8xC296SAA comes in two Quad Flat Pack (or QFP) packages. Both include 100 pins. One is square and one is rectangular. Figure 60 shows the dimensions for the NP-ADP100-QFP/EDI part. Dimensions for the other 8xC296SA adapters: NP-ADP-QFP/ET, NP-ADP-SQFP/EDI, and NP-ADP100-SQFP/ET are very similar in appearance and size.

The most significant difference between the EDI adapters and the ET adapters is the appearance of the adapter base that solders onto the target board. The EDI solder-in adapter base is shown in . The base is short and has short pins. A typical solder-in QFP adapter base for the ET parts is shown in Figure 109. Most of the height in the ET adapter is in the adapter base. Assembled, the two kinds of adapters are about the same height.



**Figure 61: Typical Emulation Technology QFP adapter Base**

With all the solder-in adapters, the base plugs into the rest of the adapter both to facilitate soldering, and to let you separate the pod from the target without separating the pod from the adapter. Once the target, adapter, and pod are assembled, you may find it easier to separate the two halves of the adapter than to separate the pod from the adapter. In either case, use care when disconnecting the pod from the target to avoid damaging any of the components.

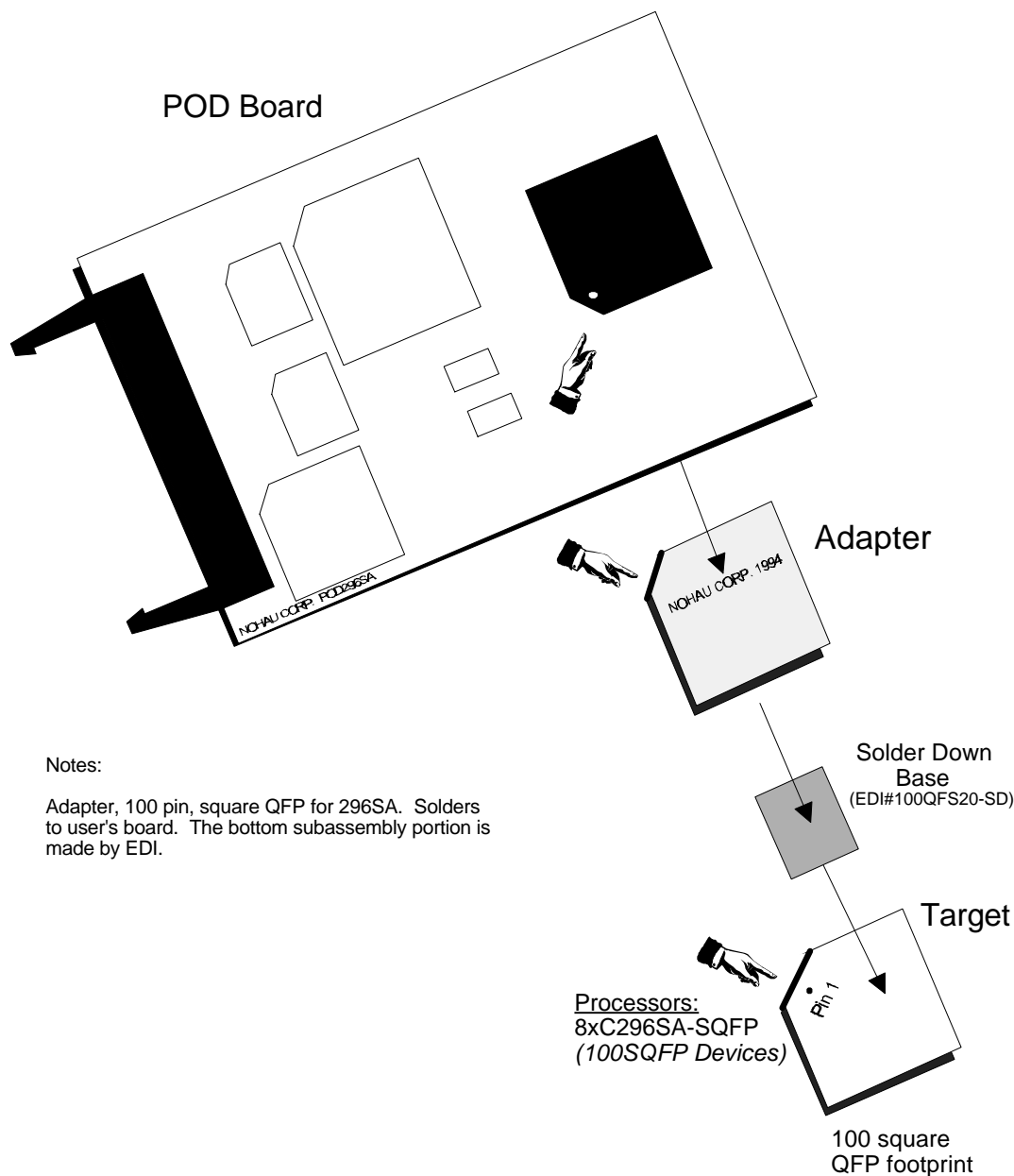


Figure 62: EMUL-PC/NP/Sx-ADP100-QFP/EDI Adapter Orientation in Detail

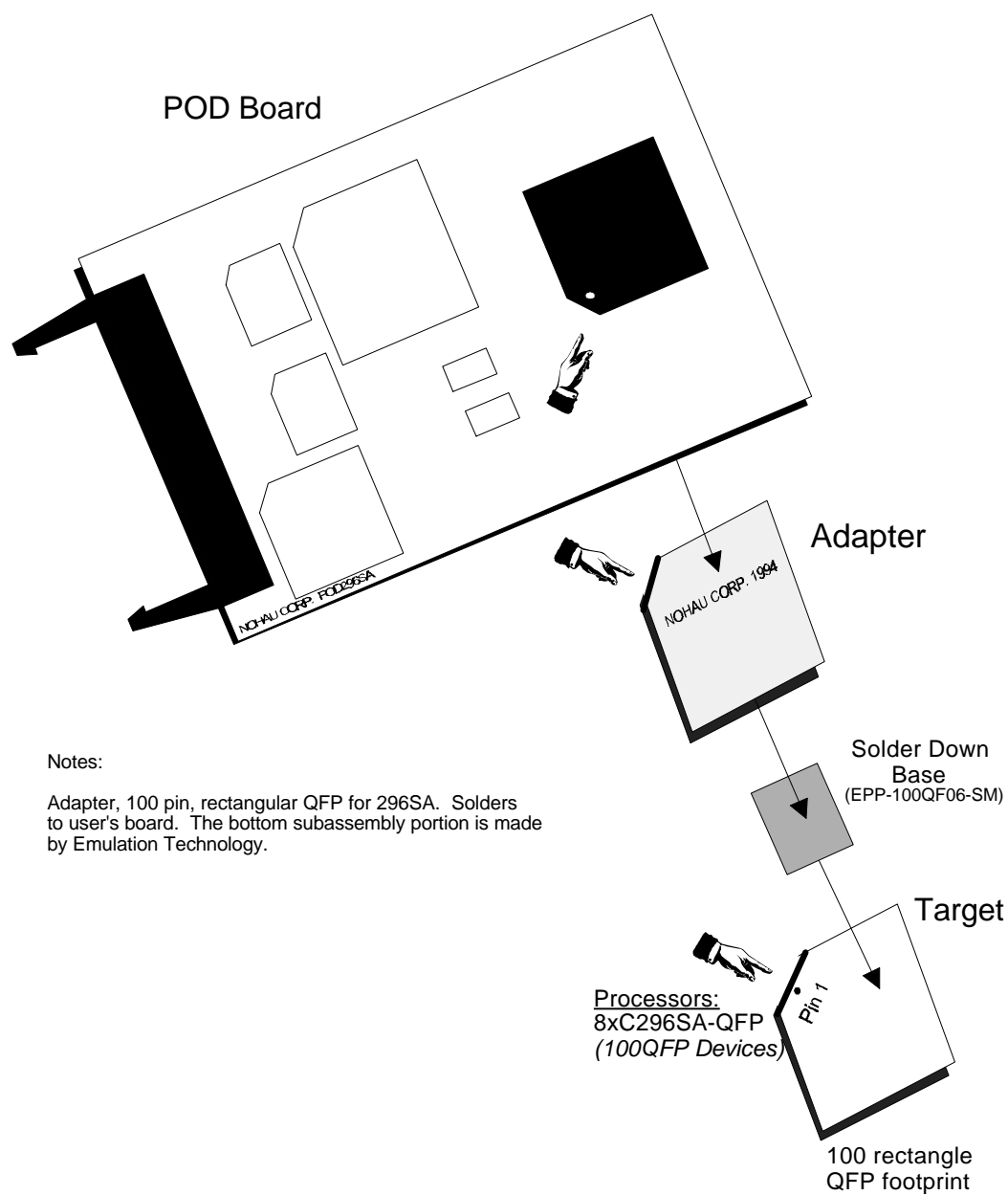
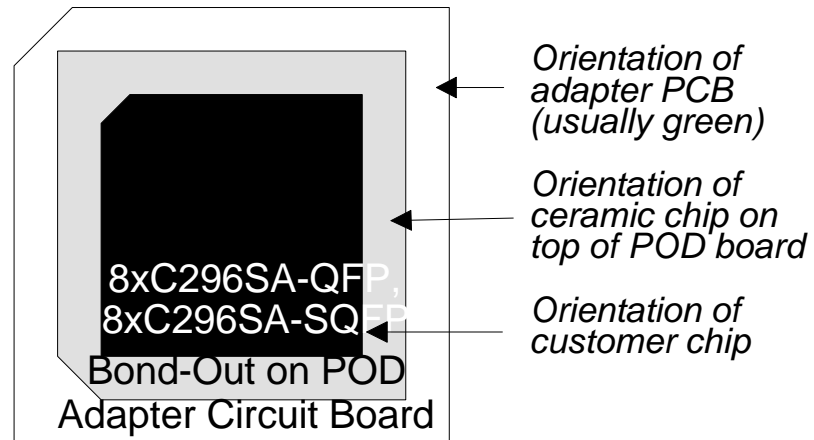


Figure 63: EMUL-PC/NP/Sx-ADP100-QFP/ET Adapter Orientation in Detail

### Surface-mount SQFP Adapters

The following diagrams show the proper adapter orientation for the microcontrollers listed.



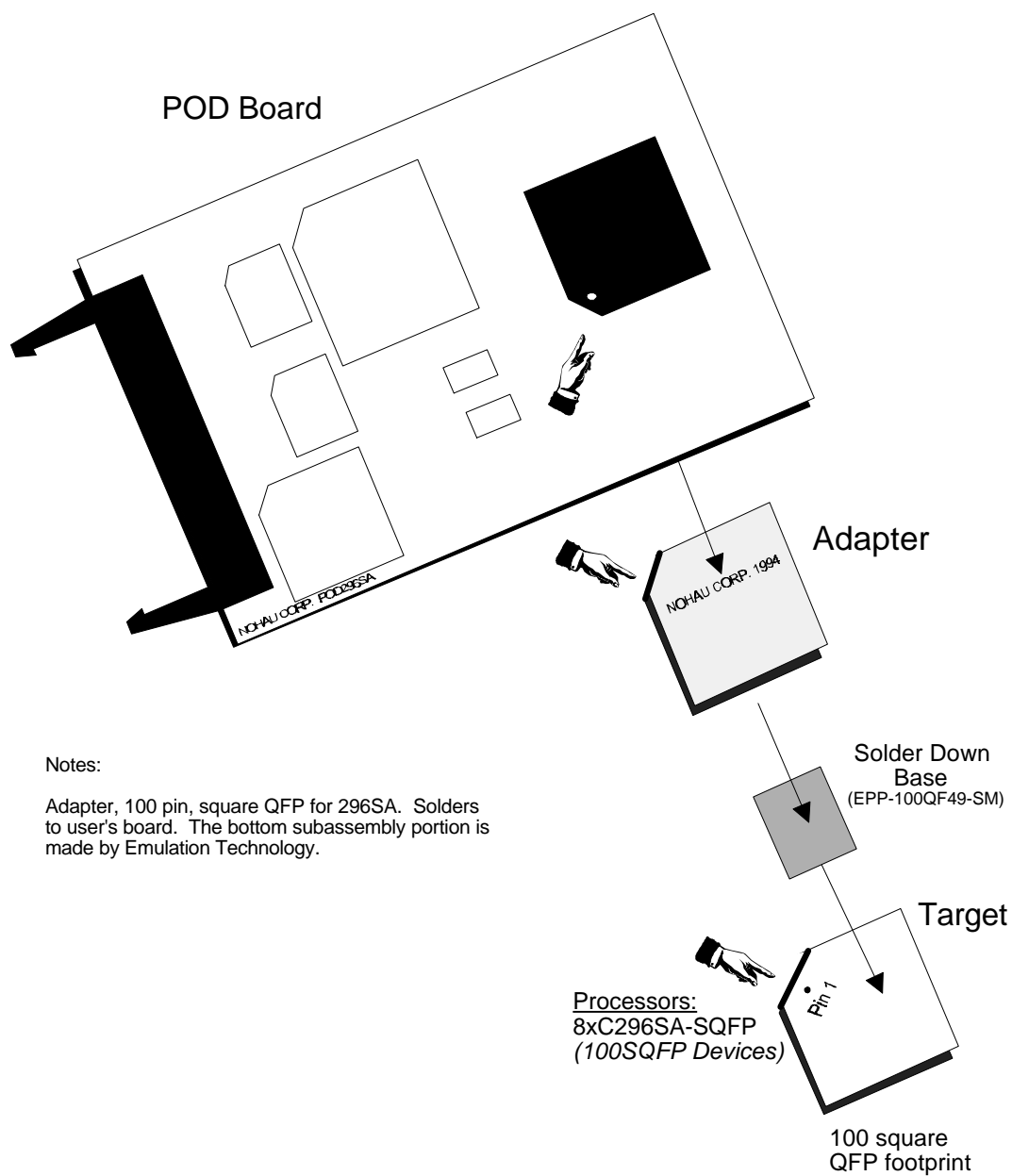
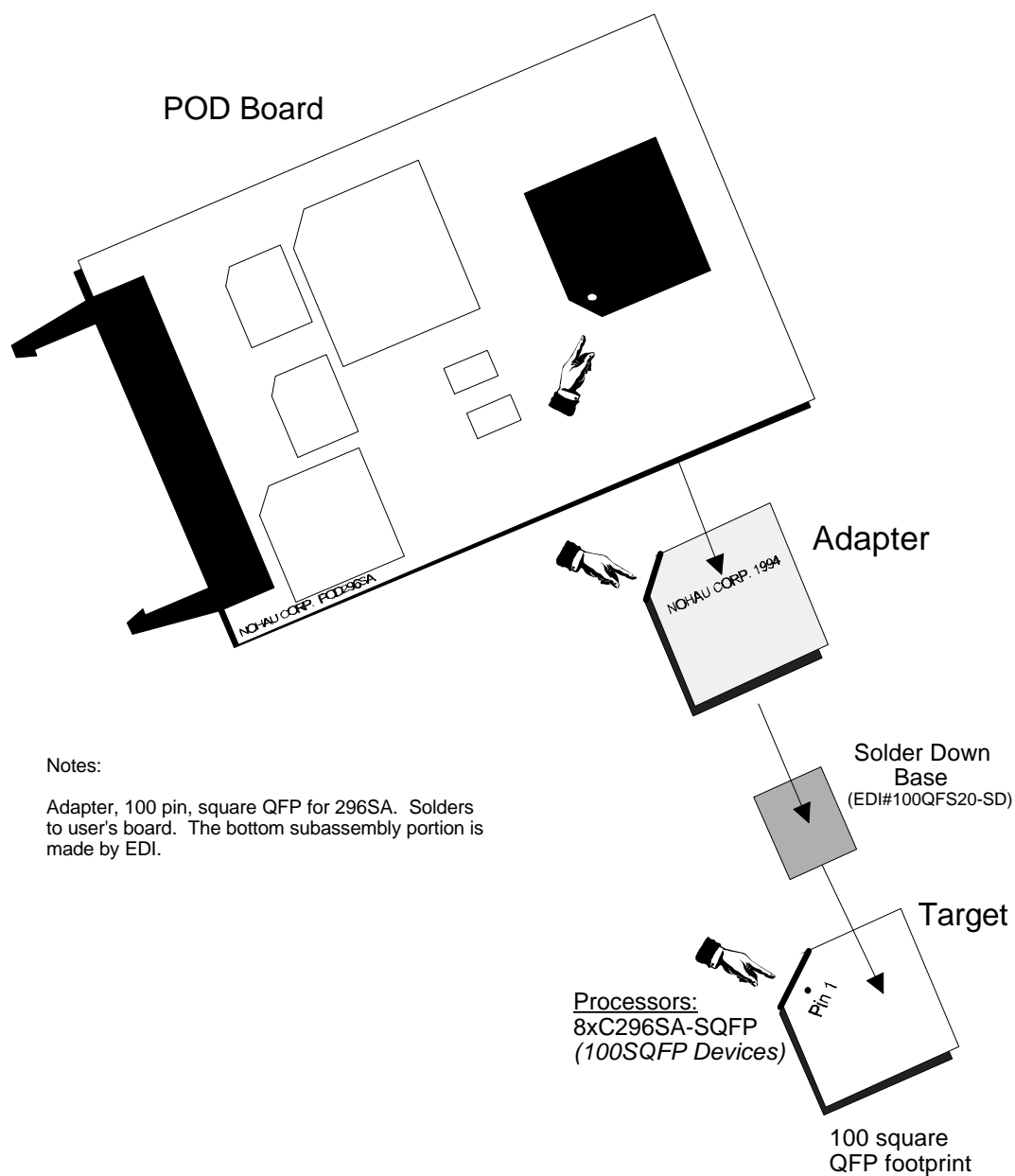


Figure 64: EMUL-PC/NP/Sx-ADP100-SQFP/ET Orientation





**Figure 65: EMUL-PC/NP/Sx-ADP100-SQFP/EDI Adapter Orientation in Detail**

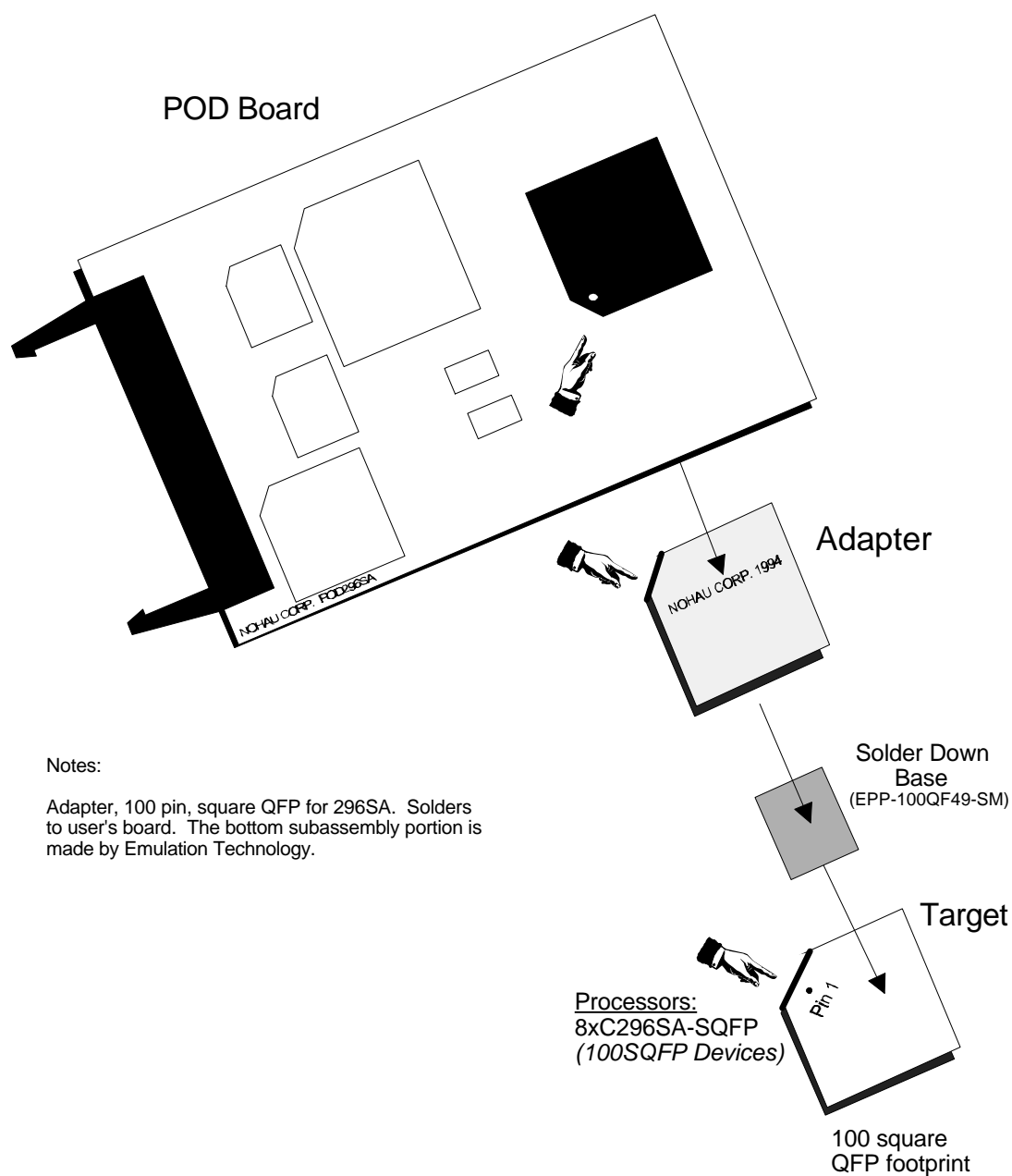


Figure 66: EMUL-PC/NP/Sx-ADP100-SQFP/ET Adapter Orientation in Detail

## Compilers

### BSO/Tasking

#### *Assembler notes*

To do Source Level Debugging, add two switches when assembling your code: debug and source.

*Note: This applies only if you have version 4.0, revision 3 or later of the BSO/Tasking assembler. Previous versions did not support this feature.*

A typical command line follows:

```
asm296 cstart.asm md(sa) farcode debug source
```

Set all other switches to match your target. For more information about other assembler settings, refer to the BSO/Tasking manual.

The example files on the release disk include a file called CSTART.ASM. For simplicity, please use that file instead of any of the start-up example files shipped with the BSO compiler when compiling the examples.

*Note: To get line number/source information from BSO/Tasking version 4.0, use the SOURCE switch.*

#### *Compiler Notes*

Like the assembler, the debug switch produces all the symbols needed by the debugger and puts them in the unlinked object file. Set all other switches to match your target. For more information about other compiler command line settings, refer to the manual from BSO/Tasking.

### IAR Systems Software, Inc.

#### *Compiler Notes*

Like the assembler, the debug switch produces all the symbols needed by the debugger and puts them in the unlinked object file. Set all other switches to match your target. For more information about other compiler command line settings, refer to the manual from IAR Systems Software, Inc..

## Chapter 8: Troubleshooting

### ***Troubleshooting Overview***

If you have emulator trouble you may contact customer support at support@icetech.com. If you do, the engineer will likely lead you through the following steps to test for the most common mistakes. To save time, you may also test for the most common reasons that the emulator is not working the way you want.

The items to check for below are in order. Start at number 1 and continue until either the emulator works or you have reached the end of the list. Each item is a short version of a description from earlier in this manual. Each item has at least one page number where more details can be found.

---

*Note: We suggest that you remove the pod from the target when you do the following steps.*

---

#### Step 1: Board I/O Addresses

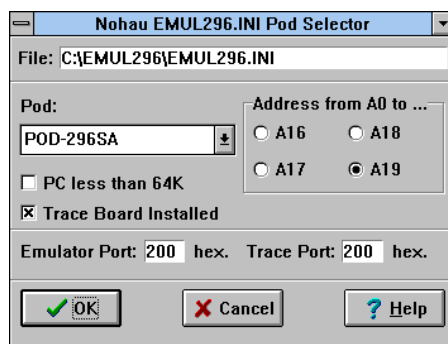
Confirm that the I/O address set in the jumpers on the emulator and trace boards both agree with the software settings found in their respective configuration dialog boxes. (For more information, see page 73.)

#### Step 2: .INI Editor

The *Windows* software is used for all EMUL296™-PC products. The type of target processor in the software configuration must agree with the type of pod you are using. If not, you may see a “Pod Communication Failed” error message. To ensure that you do not get this error, we include a utility that you probably want to run when you first install the emulator, and possibly again when you change your pod type. This utility is called INI296. (You can also run this utility any time you want to check the values in the initialization file.)

To invoke INI296, double click on the icon in the NOHAU program group labeled INI296. If the EMUL296.INI settings are not self-consistent, you will see a warning message, otherwise you will see the window shown in Figure 67.

To correctly configure your software to match your hardware, start by clicking on the button that matches your pod type. If your pod should not address memory above 64K, put a check mark in the PC < 64K box. If you are using memory above 64K, leave the box unchecked and make sure that the jumpers TRA16 through TRA19 match the field labeled Address from A0 to ... For example: if TRA16 is in the EA16 position but the rest are in the GND position, click on the button labeled A16. If all the TRA headers are in the EA position, click on the button labeled A19. (For more information on INI296, see page 18.)



**Figure 67: Choosing a target processor with INI296**

After you have set the **Emulator Port:** and **Trace Port:** fields, click on the **OK** button or press <Enter>. The **Emulator Port:** field must agree with the values you set in the J2 jumper on the emulator board, as described in the section, "Setting the I/O address jumpers -- J2" on page 70. Likewise, the **Trace Port:** field must agree with the way you set the address jumpers on the emulator board.

#### Step 3: PWR and XTAL jumpers

If there is a power supply on the target, remove the PWR jumper from the pod (for more information, see page 71).

If the crystal or oscillator on the target is running at a different frequency than the one on the pod board, move the XTAL jumpers to the target position.

For more information, see the section in the Pod chapter that describes the kind of pod you have.

#### Step 4: I/O On Addresses Pins

Most 8xC296 parts use 16 address bits. In those parts that support more address bits, the target can use from 0 to 4 of the extra address bits for I/O instead. The table below shows, for each combination of address pins used for addressing, how to set the jumpers shown in the table. Make sure the jumpers on your pod match the settings in the row that applies to your target. (For more information, see page 70.)

Bits used for Addressing	TRA16	TRA17	TRA18	TRA19
A0 - A15	GND	GND	GND	GND
A0 - A16	EA16	GND	GND	GND
A0 - A17	EA16	EA17	GND	GND
A0 - A18	EA16	EA17	EA18	GND
A0 - A19	EA16	EA17	EA18	EA19

Step 5: Chip Configuration Bytes (CCB's)

The CCB's that you specify in the hardware configuration menu must match what the microcontroller reads from location 2018 at reset. For example, if you have mapped 2018 to a target with eprom that contains CCB's specifying 8 bit mode while your hardware configuration menu specifies 16 bit mode, you will surely run into trouble. (For more information, see page 25.)

---

*Note: CCB's on 80C296 pods running in "big mode" (i.e., PC>64k) are fetched at FF2018.*

---

Step 6: The Stack Pointer

The Stack Pointer must point to valid even memory location at all times. The emulator needs either 2 bytes or 4 bytes of temporary storage on the stack. (SP should have a value > 0x50). (For more information, see page 87.)

Step 7: Interrupt Vectors

Support for software breakpoints requires specific values for certain interrupt vectors. When troubleshooting target systems that use 16 bits of addressing, confirm that the following addresses have the following values:

Address:	0x0012 & 0x0014	0x2010	0x2012
Value:	0x0000	0x0012	0x0012

When troubleshooting a target design that uses a processor with 20 bits of addressing like the 8xC296SA, add an address offset of 0xFF0000 to each of the above addresses to locate the interrupt vectors:

Address:	0xF0012 & 0x0014	0xF2010	0xF2012
Value:	0x0000	0x0012	0x0012

If you map these addresses to the target ROM, be sure your ROM contains these values at those addresses. If it does not, software breakpoints will not work.

Step 8: Sample User Program

If you telephone the technical support team, you will probably be asked to do the following to enter a sample user program:

1. Click in the Program Window
2. Hit <Ctrl>-A
3. Type in address 2080

4. Hit <Enter>
5. Type: NOP <Enter>  
NOP <Enter>  
LJMP 2080 <Enter>
6. Click on the GO button in the tool bar.
7. Click on BREAK.
8. Make a note of software revision.
9. Make a note of compiler revision.
10. Make a note of serial numbers of boards.

## INDEX

---

.

.INI Editor · 2

---

## 8

80C296 · 2

---

## A

Accessories · 2

Adapters

    Surface-mount QFP, SA family · 2

    Surface-mount SQFP, SA family · 2

**Add ..** · 2

**Add a watch point ..** · 2

**Address Range** · 2

**Address space ..** · 2

**Address..** · 2

**Animate ..** · 2

**Arrange Icons** · 2

**At ..** · 2

---

## B

**Bargraph** · 2

Benchmarking

    Using Timestamp · 2

Bin

    Adding a · 2

**Block move..** · 2

**Break**

    Emulation · 2

**On internal access ..** · 2

Break Emulation? Box · 2

**Break now!** · 2

Break Two Emulators Simultaneously · 2

Breakpoints

    Hardware · 2

BSO/Tasking · 2

Bus Cycle Order · 2

Bus Width · 2

---

## C

**C call stack ..** · 2

**Call stack ..** · 2

**Cascade windows** · 2

Child Windows · 2

Chip Configuration Bytes · 2

**Close** · 2

**Color ..** · 2

Communication Rate Jumper · 2

Compilers · 2

Confidence Test · 2

Convert cycles to time · 2

**Copy to clipboard** · 2

---

## D

**DataDisplay as..** · 2

**Default CPU symbols** · 2

**Delete All** · 2

Dialog Boxes · 2

**Disable all** · 2

Display as.. · 2

Duplicate Resources · 2

Dynamic Data Exchange · 2

---

## E

Edit

    Trigger Conditions · 2

**Edit ..** · 2

EMUL/LC-ISA · 2

EMUL296-PC/SA-ADP100-QFP/EDI

    Dimensions · 2

EMUL296-PC/SA-ADP100-QFP/ET

    Dimensions · 2

Emulator

    Board · 2

    Detailed Installation Instructions · 2

    Hardware Configuration · 2

    Macro User Guide · 2

    Memory · 2

**Emulator Hardware ..** · 2

EMUL-PC/NP/Sx-ADP100-QFP/EDI

    Orientation · 2

EMUL-PC/NP/Sx-ADP100-QFP/ET

    Orientation · 2

EMUL-PC/NP/Sx-ADP100-SQFP/EDI · 2

EMUL-PC/NP/Sx-ADP100-SQFP/ET · 2

    Orientation · 2

Enable Code Space Limits · 2

**Evaluate ..** · 2

**Exit** · 2

Extend Recording · 2

External Inputs and Controls · 2



---

**F**

Fast Break Write · 2  
**Fast\_Br\_W** .. · 2  
**Fill**.. · 2  
 Filter Mode: Normal · 2  
 Filter Mode: Window · 2  
 Find Frame number .. · 2  
 Find Trig point · 2  
**Full Reset** · 2  
**Function** · 2

---

**G**

**Go** · 2  
     **FOREVER** · 2  
     **To** .. · 2  
     **To cursor** · 2  
     **To return address** · 2

---

**H**

**Hardware breakpoints** .. · 2  
**Hardware breaks only** · 2  
 Headers  
     Auto Map · 2  
     Auto/Manual BW · 2  
     Clock · 2  
     JP30-JP32 · 2  
     JP6, JP12, JP17 & JP21 · 2  
     Memory Mapping · 2  
     PWR · 2  
     RDY · 2  
     RST · 2  
     RXD · 2  
 Help Line · 2

---

**I**

I/O Addresses · 2  
 I/O On Addresses Pins · 2  
 IAR Systems Software, Inc. · 2  
 Indicator Lights · 2  
 INI296 · 2  
 In-line assembler · 2  
**Inspect** .. · 2  
 Installation Instructions, Quick · 2  
 Installing  
     Emulator · 2  
     Pod · 2  
     Software · 2  
     Trace Board · 2  
 Internal Addressing or Single Chip Mode · 2  
 Interrupt Vectors · 2

---

**L**

**Last trig event repeat count** · 2  
**Length** field · 2  
**Load**  
     Code · 2  
     Default symbols · 2  
     EEPROM · 2  
 Load and execute a program · 2

---

**M**

Macro  
     Constants · 2  
     Example · 2  
     Global Variables · 2  
     Nohau Subroutines · 2  
     Procedure for writing · 2  
     Setup · 2  
     Subroutine Reference · 2  
 Macro System · 2  
 Manual Conventions · 2  
 Mapping  
     Memory · 2  
     Memory Using Chip Selects · 2  
 Memory  
     Coverage · 2  
     Coverage Report, Detailed · 2  
     Coverage Report, Summary · 2  
     Mapping · 2  
**Memory Coverage** .. · 2  
 Menus · 2  
     Breakpoints · 2  
     Config · 2  
     Data · 2  
     File · 2  
     Help · 2  
     Program · 2  
     Register · 2  
     Run · 2  
     ShadowRam · 2  
     Source · 2  
     Stack · 2  
     Trace · 2  
     View/Edit · 2  
     Window · 2  
*Microsoft Visual Basic* · 2  
**Miscellaneous** .. · 2  
 Miscellaneous bits · 2  
 Miscellaneous Configuration · 2  
**Miscellaneous Setup** · 2  
**Module** · 2  
 Multiple Document Interface Standard · 2

---

**N**

Next window · 2

---

**O****Open**

- A new data window · 2
- A new program window · 2
- A new register window · 2
- A new shadow ram window · 2
- A new source code window · 2
- A new trace window · 2
- A Watch window · 2

Origin (at program counter) · 2

Original Address · 2

---

**P****Parameters in Hex · 2****Paths**

- Setting · 2

**Paths .. · 2**

Performance Analysis · 2

Pipeline Effects · 2

**Pod**

- Breakpoints · 2
- Configuration Requirements · 2
- Emulation Memory · 2
- Headers · 2

Pod Boards · 2

Pod Dimensions · 2

POD-296-256-SA-50 · 2

**Post trigger samples · 2**

Power Supply to Pod / Target · 2

**PP Analyzer · 2****Preferences · 2****Programming**

- Algorithms · 2
- External Flash Memory · 2
- Options · 2

**Project name .. · 2**

Projects · 2

- Creating · 2

PWR and XTAL jumpers · 2

PWR Header -- JP2 · 2

---

**Q****QFP/SQFP**

- Adapter Orientation · 2

Quick Start Instructions · 2

---

---

**R**

RAM value, change · 2

**Refresh · 2**

Relative timestamp · 2

**Remove .. · 2****Remove Symbols · 2****Reset**

- and Go · 2

- Chip · 2

Reset vs. Full Reset · 2

---

**S**

Sample User Program · 2

**Save**

- Trace as text .. · 2

**Save code as .. · 2****Search**

- Address .. · 2
- Next Address · 2
- Previous Address · 2

**Search next · 2****Search previous · 2****Search.. · 2****Set new PC value at cursor · 2****Setting**

- I/O address jumpers -- J1 · 2
- Target Communication Rate -- Header JP1 · 2

**Setup .. · 2**

Setup Instructions, Quick · 2

**Show**

- Misc. data · 2
- Timestamp · 2

**Show function · 2****Show load info .. · 2****Software breakpoint**

- Delete · 2
- Make inactive · 2
- Setting · 2

Software Configuration, Initial · 2

Software Installation Instructions, Detailed · 2

Software, Configuring · 2

Stack Pointer · 2

**Step into · 2****Step over · 2**

Symbols in the Trace Window · 2

Synchronize program window · 2

System Requirements · 2

---

**T**

T = 0 at Cursor · 2

**Tile windows · 2****Toggle · 2****Toggle breakpoint · 2**

**Toggle help line** · 2

Toggle trace (stop/run) · 2

Tool Bar · 2

Trace

Board Installed · 2

Board, Detailed Installation Instructions · 2

Board, Introduction · 2

Clock Rate · 2

Config Menu · 2

**I/O Address** field · 2

Input Pins · 2

Memory · 2

Setup .. · 2

Setup Dialog Box · 2

Triggers · 2

**Trace ..** · 2

Tracing

Introduction to · 2

Triggers · 2

Troubleshooting · 2

---

## U

**User defined symbols** · 2

---

## V

**View assembly code** · 2

**View source window** · 2

---

## W

Wait States · 2

Window

Colors · 2

Memory Coverage · 2

PPA Control · 2

Source · 2

Windows

Data and ShadowRam · 2

Evaluate · 2

Inspect · 2

Other · 2

Program · 2

Register · 2

RTXC · 2

Source · 2

Stack · 2

Trace · 2

Watch · 2

Windows API · 2

---

## Z

**Zoom** · 2