ICE TECHNOLOGY NOHAU®

EMUL251™ -PC Windows

User Guide

EMUL251™-PC Windows

User Guide

Edition 3

Warranty Information

The EMUL251™-PC Windows 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 the discretion of ICE Technology.

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 our engineer 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 EMUL251™-PC Windows 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 only.

Table of Contents

Introducing EMUL251™-PC Windows

The EMUL251™-PC Windows is a personal computer-based, emulator for Intel's MCS 251 microcontrollers. The EMUL251™-PC Windows consists of emulator "plug-in" board, a five-foot-long (1.5 m) twisted-pair ribbon cable, a pod board and an optional Trace plug-in board.The EMUL251™-PC Windows design supports the Intel 8XC251SB microcontroller.

The EMUL251™-PC Windows software is a *Microsoft Windows 3.1, 3.11, Windows for Workgroups,* and *Windows '95* application. It follows the *MS Windows* Multiple Document Interface Standard. That means that it has the same look and feel as applications produced by Microsoft and others for *MS Windows*.

The EMUL251™-PC Windows Windows user interface is 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 EMUL251™-PC Windows 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 EMUL251™-PC Windows 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 tracing, triggering and filtering of executed instructions, as well as data transfers.

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*. Many of the EMUL251™-PC Windows features are designed around the features of the supported chip architecture. 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 you do not have., such as the trace options if you do not have one.

If you have used emulators with other microprocessors,

and understand breakpoints, 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 your podthe applicable pod board 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, then begin using EMUL251™-PC Windows, referring to on-line help as 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, but without the quotation marks, 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 and hold down the `<Alt>` key, then press the `F` key. Window names and labels that appear on the screen are printed using the **screen** font to set them apart from the rest of the text.

Notes and hints are printed in italics, and

Warning: Warnings are boxed to set them apart from the rest of the manual text. Pay careful attention to them.

Address Notation

24-bit addresses are shown in this manual with the memory segment before a colon, e.g., FF:0100. However, they must be entered into the emulator fields without the colon, e.g., FF0100. Hexadecimal notation is assumed in emulator address fields, so you need only enter the significant hexadecimal digits. No other notation, such as an "H" suffix for hexadecimal nor a "0x" or "$" prefix, is necessary. Leading zeroes are not required, so 00:047A may be entered as just "47A" and 00:0000 may be entered simply as "0".

Quick Installation Instructions

System Requirements

The EMUL251™-PC Windows requires a personal computer with at least one free ISA (or EISA) -bus slot. The Trace board, if present, will require one additional full-length, 16- bit slot near the emulator board. The PC must also have at least 2 megabytes of RAM (8 megabytes for Windows '95), a CPU that is either80386, 80486, or Pentiu- compatible, a hard disk with at least 3 megabytes of unused space, and *Microsoft Windows* 3.1, 3.11 or *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 more detailed steps for installing and configuring each board and the software as outlined in their respective chapters.

Warning: Always remove the Stand Alone jumper on the pod before connecting the pod to a target system.Warning:Always turn on the PC before applying power to the target. Always turn off the target power before turning off the PC power.

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. Unless you will be installing a trace board, you may now close the cover, and attach the ribbon cable to the emulator and the pod.

Installing the Pod

With the PC power off, line up the ribbon-cable connector key with the keyed 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, open the jaws on the pod connector, line up the key with the keyed slot on the pod board, and insert the ribbon cable connector into the slot firmly, pressing until the jaws on the pod close (or nearly close). Remove any antistatic foam from the pins on the bottom of the pod. 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 below in “Quick Start Instructions.”

I/O Port Locations

The installation software will ask for an I/O port address. I/O addresses are in hexadecimal, with no special notation. The default emulator address is 200. If you need to change the emulator board address, see the Emulator Board chapter for instructions on changing the address jumpers before you can run Setup. Otherwise, use the default address for the emulator board.

If you accept the default address, but later need to change it, you can change the jumpers and the setting in the software configuration as needed.

The default trace board address is 208. As with the emulator board, you may change the jumpers according to detailed instructions included in the Trace Board chapter. Otherwise, accept the default address for the Trace Board. As with the emulator, you may change the address later if necessary.

If you do not have a Trace Board option, enter "0" as the Trace Port address.

Installing the Software

To install this software under *MS Windows 3.x*, 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. If using *Windows* ‘95, open the **My Computer** facility, select the floppy drive containing the installation diskette, then double-click on the **Setup** icon.

After SETUP.EXE is started, a dialog box will ask for a directory for the EMUL251™-PC Windows software. Either accept the suggested directory or type a different one. SETUP will create the directories as needed, decompress 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 program group containing the EMUL251-PCicon. Double-clicking on this icon will start the EMUL251™-PC Windows Windows application.

Installing the Trace Board (If used)

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 trace board into the slot near the emulator board. Once you are sure the trace board is fully inserted, use a screw to secure it.

5. Connect the two ribbon cables attached to the Trace Board to the emulator board, making sure that the pins are fully inserted into the connectors, there are no exposed pins, no connector is offset relative to the pins either vertically or horizontally, there are no twists in either cable, and the cables do not cross. The most common error is to insert only one row of pins into the connector. This could damage either of the boards.

6. Once the ribbon cables are attached, close the PC cover, and install the pod.

Quick Start Instructions

This section describes how to quickly start using **EMUL251-PC** Windows to debug an existing program or target board once the **EMUL251-PC** hardware is installed.

Starting the EMUL251-PC Software

Double-click on the **EMUL251** icon to start the application.

To load and execute a program:

1. Select **Load code ..** from the **File** menu and find a file to load by using the dialog box. TIMES.OMF in one of the subdirectories under the EXAMPLES subdirectory is a typical file.

2. Select **Reset** from the **Run** menu.*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 FF0000
   type <Ctrl>-N to force the program counter to address FF0000
   hit <Enter>
   type: NOP <Enter>
   NOP <Enter>
   LJMP 0000 <Enter>
   then continue with item 3 below.

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

To set a breakpoint either:

1. Click twice on the desired instruction bytes or mnemonic 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 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 **Breakpoints 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.

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 an OnChip RAM window until it shows the address to be changed or hit **`<Ctrl>A`** and type in the desired address.

2. Highlight the data 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>`.

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 EMUL251™-PC Windows 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 type **`A:SETUP`** as the file to run.

If your installation disk is not in drive A, replace the letter “A” with the letter corresponding to the appropriate drive.

From *Windows ‘95*, start the My Computer facility by double-clicking on the **My Computer** desktop icon, and selecting the disk drive that contains the installation disk: double-click on the **drive** icon, then double-click on the **Setup** icon. As an alternate, you may also select **Run** from the **Start** menu, then **Run** from the list, then type **`A:SETUP.`** Windows NT users must be logged in as administrator before they can install the software.

A dialog box will ask for a directory for the EMUL251™-PC Windows software. Either accept the suggested directory (c:\win68), or type a different one. **Setup** will copy files from the floppy to the hard disk directory specified and modify the configuration information stored in the ".ini" files as needed.

For *Windows ‘95* users, there will be a **EMUL251** program group with an **EMUL251** icon. Double-clicking on the emulator icon will start the EMUL251™-PC Windows 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 EMUL251™-PC Windows products. The I/O addresses in the software configuration must agree with the jumper settings on the boards you are using. If not, you may see an error message.

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 any of: the Emulator board, the Trace 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 as needed.

Projects

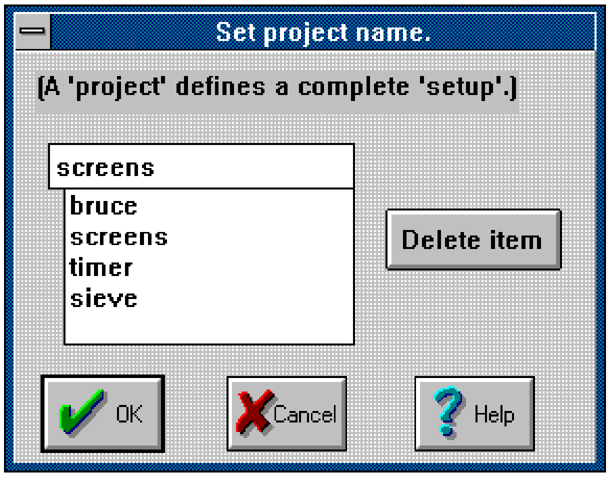
A project is a collection of software configuration settings that are all associated with a specific person, target, or software development project. The menu item opens a dialog box that allows you to set up named configurations or projects. This is first in the menu and described first because all 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 called 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 say, 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 1: Set Project Name Dialog Box

To add a project, open the **Set Project Name** dialog box and 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 2: EMUL251™-PC Windows Title Bar

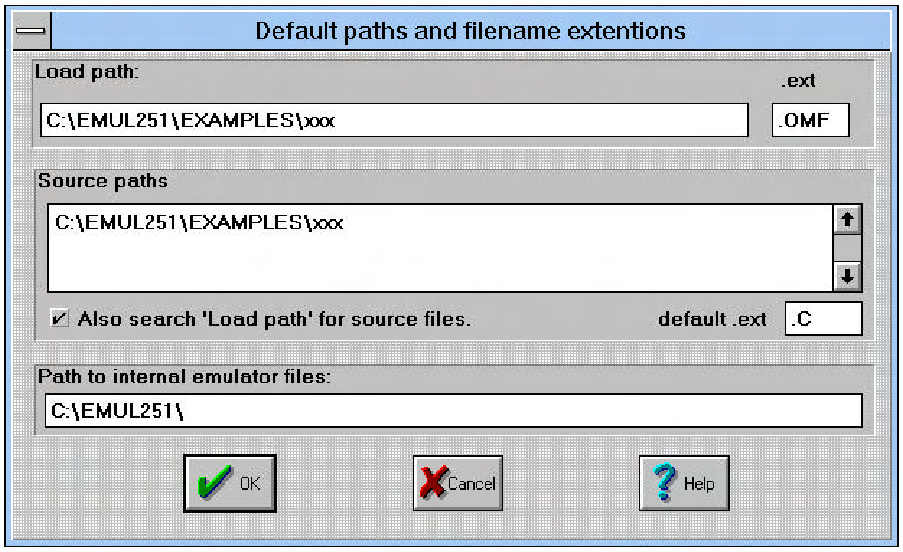
The name of the current project appears in the title bar, which appears at the top of the background window. Figure 2 is an example of the **EMUL251** title bar for the project named **SCREENS** (used to create the screen shots for this manual).

Setting the Paths ..

The next item in the **Config** menu is **Paths ..** , which opens the dialog box shown on page17. 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 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 the default directory with this extension will be listed in the **Load code..** dialog box.


Default paths and filename extentions

Load path: C:\EMUL251\EXAMPLES\xxx .ext .OMF

Source paths: C:\EMUL251\EXAMPLES\xxx

✓ Also search 'Load path' for source files. default .ext .C

Path to internal emulator files: C:\EMUL251\

Figure 3: Paths Dialog Box

With many compilers, the full path name of each 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 EMUL251™-PC Windows is loading, the debugger will first look 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 you, the user. Multiple directories may be entered by using a semicolon between them. Once entered, directories will stay here until you remove it. The small check box, when checked, will tell EMUL251™-PC Windows 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.

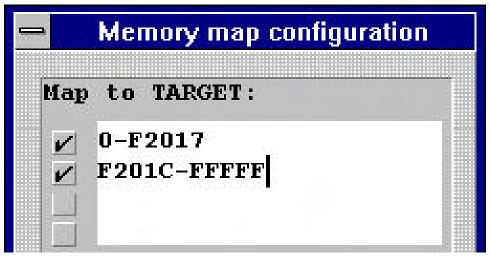
Note: *The ".ext" field specifies the source file extension. If your C modules have the extension ".c", enter that. To see assembler source (.asm) in the Source window, enter ".asm" . The assembly source feature is only available if the assembler and linker provide assembly source information to the emulator.*

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 EMUL251™-PC Windows 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 EMUL251™-PC Windows Windows software 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, or emulation memory. The entire address range for both ROM and RAM can be mapped to either the target orthe emulator memory in blocks as small as 2 bytes in non-page mode. In page mode, blocks may be as small as 256 bytes. The **Memory map ..** menu item under the **Config** menu opens the dialog box that controls those address ranges.

When an address is mapped to emulation RAM on the pod, all READ, WRITE, and instruction fetch cycles at 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.



0-F2017
F201C-FFFFF

Figure 4: Mapping Memory Addresses to the Target

There are 16 address ranges in the dialog box. Each address range can be as large as all of memory or as small as a single word. Each range may start on any even address and may end on any odd address. To define an address range, place a check mark in any check box, place an insertion point to the right of that box, and type the logical address range you want. For convenience, there is a button that will map all memory to the emulator.

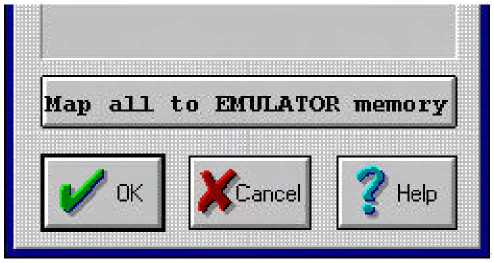
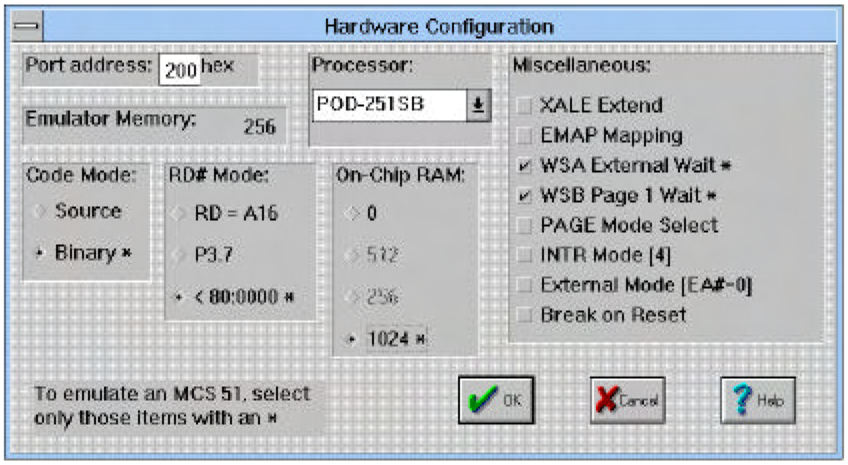


Figure 5: Exiting the Memory Map Dialog Box

When all the memory map ranges are set the way you want them, click on the **OK** button. If you click on the Cancel button, the most recent changes will be discarded before the dialog box is closed.

Emulator Hardware Configuration

The **Emulator Hardware ..** menu item configures the software to correctly communicate with the hardware. The **Emulator port** field contains the hexadecimal address of the emulator in the PC's I/O space. The value entered in the Emulator Port field must agree with the jumper settings on the emulator board header J2 (see "I/O address" on page69).If they do not agree, the EMUL251™-PC Windows software will not be able to communicate with the hardware, and the dialog box will automatically be displayed as a reminder that communication has failed and some change is needed.



Hardware Configuration

Port address: 200 hex

Emulator Memory: 256

Processor: POD-251SB

Code Mode: Source [✓ Binary *]

RD# Mode: RD = A16 P3.7 [✓ < 80:0000 *]

On-Chip RAM: 0 512 256 [✓ 1024 *]

Miscellaneous:
- XALE Extend
- EMAP Mapping
- ✓ WSA External Wait *
- ✓ WSB Page 1 Wait *
- PAGE Mode Select
- INTR Mode [4]
- External Mode [EA#=0]
- Break on Reset

To emulate an MCS 51, select only those items with an *

Figure 6: Hardware Configuration Dialog Box

Warning: The settings in this dialog box must agree with the emulator jumper settings, the pod processor type, and how the application start-up code sets up certain control registers. If this is not the case, EMUL251™-PC Windows will not work properly.

Most of the items in the **Hardware Configuration** dialog box affect settings found in the 8XC251SB's CONFIG registers. Settings in this screen allow you to configure operation through the emulator interface, without having to program the CONFIG registers of an actual 8XC251SB microcontroller. For details about the 8XC251SB address spaces, external memory interface, instruction types, modes, and configuration, see the Intel 8XC251SB embedded Microcontroller User's Manual.

The **Code Mode** selection of **Source** or **Binary** determines how the instruction bytes will be executed and interpreted. This selection corresponds to setting or clearing the SRC bit in the CONFIG0 register. This mode must match the assembly, compile, and linking selection of your assembler, compiler, and linker. Otherwise, instructions will not be executed properly.

The **RD# Mode** determines the number of external address lines and the address ranges for strobing the PSEN# and RD# signals. This selection corresponds to the valid combinations of the RD1 and RD0 bits in CONFIG0.

The **On-Chip RAM** of the 8XC251SB is 1024 bytes. Select 1024 for normal operation. Selecting 0 directs on-chip memory accesses to emulator memory instead of on-chip memory. This allows the optional trace board to record the accesses. At the time of this writing, trigger or filtering the trace on these accesses is not available. Cycle lengths are longer with 0 selected.

The **XALE Extend** box corresponds to the XALE bit in CONFIG0. Checking this box is the same as activating (clearing) the XALE bit.

Checking the **EMAP Mapping** box corresponds to activating (clearing) the EMAP bit in CONFIG1.

WSA External Wait and **WSB Page 1 Wait** correspond to the WSA bit in CONFIG0 and the WSB bit in CONFIG1. Checking a box is the same as activating (clearing) a bit in a register.

INTR Mode [4] corresponds to the INTR bit in CONFIG1. Checking this box is the same as activating (setting) the bit, which causes four bytes to be pushed onto the stack for interrupts and four bytes to be popped on returns from interrupts.

External Mode [EA#=0] specifies external access in the range of FF:0000 to FF:3FFF with the target EA# pin high. Access in this 16-k range from the emulator memory have the same timing as access from on-chip program memory. Also when the target EA# pin is high, all access in the range FF:0000 to FF:3FFF are automatically mapped to emulation memory.

When using external mode, program access within the FF:0000 to FF:FFFF range take a greater number of clock cycles. This condition exists when the EA# pin is asserted (low). The emulator will emulate exact 8XC251SB timing for both external and internal modes.

Selecting external mode in this box allows you to emulate the slower timing of off-chip access without requiring that the EA# pin be low, or actual target memory be connected. This can be useful to use emulator memory to emulate off-chip code access as if they were from external memory.

Break on Reset will cause a break in emulation when the reset line from the target system is asserted or a watchdog timeout occurs. Normally a reset will not cause a break in emulation, and program execution will continue from the reset vector. Selecting this box causes the emulator to break upon target reset rather than continue emulating.

Miscellaneous Configuration

The **Miscellaneous** item in the **Config** menu opens a dialog box that controls special features of EMUL251™-PC Windows Windows:

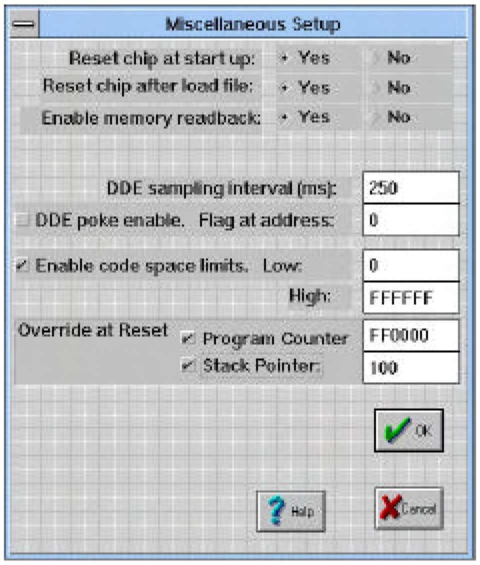
- when and if automatic resets occur

- optional reset vector values

- the source code address range for limiting where breakpoints are set

writing values to memory while the application is running

By default, the emulator resets the controller when the EMUL251™-PC Windows 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.



Reset chip at start up: [✓ Yes] No
Reset chip after load file: [✓ Yes] No
Enable memory readback: [✓ Yes] No
DDE sampling interval (ms): 250
DDE poke enable. Flag at address: 0
✓ Enable code space limits. Low: 0
High: FFFFFF
Override at Reset
✓ Program Counter FF0000
✓ Stack Pointer: 100

Figure 7 : Miscellaneous Setup Dialog Box

Even though the next field appears to work, it is not implemented yet. Writes to memory are always read back. For information about the status of these features, email support at support@icetech.com.

The next field in the **Miscellaneous Setup** 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.

Occasionally, while the target 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. EMUL251™-PC Windows supports "poking" a value into an address. 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 Shadow RAM value to RAM.

The check box to the left turns poking on or off: a check turns it on. When EMUL251™-PC Windows 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 EMUL251™-PC Windows will suspend the emulation for approximately 200-250 microseconds while it updates RAM. Finally, EMUL251™-PC Windows 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.

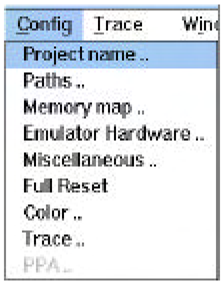
Note: *Displaying Shadow RAM and sending Sending a value to another application does not slow the emulation.*

Enable Code Space Limits

In some symbol 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 EMUL251™-PC Windows software try to set breakpoints in your data 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, EMUL251™-PC Windows will not put breakpoints outside that address range.

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 EMUL251™-PC Windows resets the controller. If you have some test code at an address other than FF:0000 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.

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 and break** from the **Run** menu, or pressing **`<Ctrl>`F2**. With both kinds of reset, the controller is reset by pulling the reset line high. When the emulator software is first started, 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 (which is why 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.

Window Colors

Under the **Config** menu is the **Color ..** menu item. Open this dialog box to set the colors of different kinds of EMUL251™-PC Windows 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 each window very distinctive.

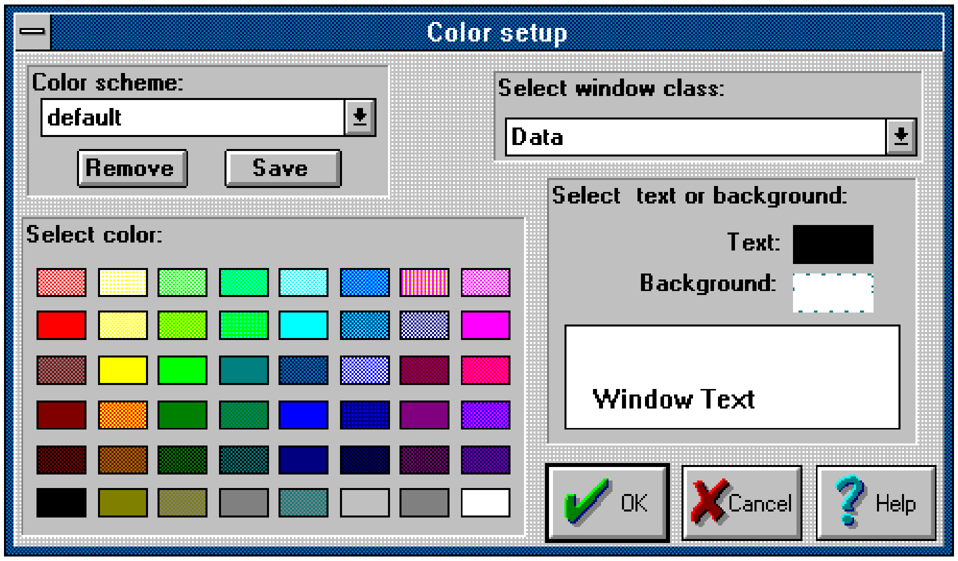

Color setup
Color scheme: default
Select window class: Data

Figure 8: The Color Setup Dialog Box

For each window class that you wish to change, select it 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 your creation ("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 due to constraints imposed by MS Windows and your video configuration. EMUL251-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 in this dialog box will show you the colors that will actually be used by EMUL251-PC.*

Trace Config Menu

For information about the **Trace ..** menu item, please refer to "Chapter 3: Trace Board" on page69 .

Performance Analysis

PPA Analyzer in the **Config** menu is not yet implemented.

: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 first line of the Tool Bar is mainly for emulator functions; the second line is for trace functions. 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 **Pos** button allows pointing the active window to anew position. The **Trace** button changes to stop when the trace board is busy, and stops the trace buffer when the button is clicked. The **Setup** button opens the **Trace Setup** dialog box. The **Trace Beg** button resets the Trace board and starts bus cycle recording according to the conditions set in the **Trace Setup** dialog box.

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 eight primary child windows created by EMUL251™-PC Windows: **Program** windows, **Data** or Memory windows, **Inspect** windows, **Source** windows, a **Registers** window, a **SpecialRegs** window, **Trace** windows (even if you have no Trace board), **Watch** windows, and a **Call Stack** window. 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 EMUL251™-PC Windows exits, and will be restored when the software restarts.

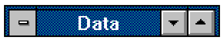
Each child window has a corresponding menu that appears between the **Config** menu and the **Window** menu. The menu contains items that only make sense within the context of that window. This window-specific menu will also appear at the cursor when you click with the Right mouse button in the body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

The menu also includes a display feature 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



Use **Data** windows to examine or modify emulation or target memory directly. EMUL251™-PC Windows uses the controller to read and write RAM, so the **Data** window cannot be updated while the emulation is running. If the application attempts to repaint the window, asterisks will be displayed until the next time the controller breaks emulation.

Data can be displayed or modified in a variety of formats. Keep adding new windows for each display format and starting address.

Selecting any **Data** window displays the **Data** menu which supports filling memory, jumping the selected window to a specific address, moving blocks of data, 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 .

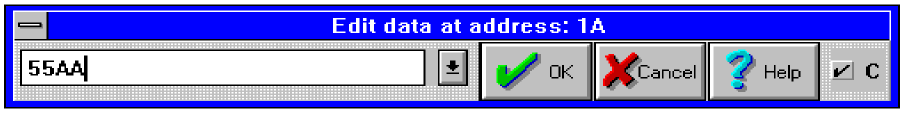


Figure 9: Editing Memory with a Data Window

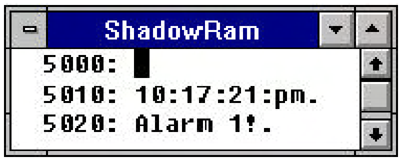
To avoid confusion, always enter the new data in the same format that the data is displayed. If the **Data** window is displaying ASCII characters, type the new character (not a string) in ASCII. If the **Data** window is displaying signed integers, enter the new value as a decimal number. Symbols are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

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 “abcd” 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).Shadow RAM Windows

A **Shadow Ram** window is a type of **Data** window. It displays the contents of Shadow RAM on the emulator board in any of the same formats as any **Data** window. A **Shadow Ram** window is updated even though the application is running at full speed. This is made possible by the logic on the pod that captures the data value and address for every bus WRITE cycle executed from user code. Those two values are sent to the emulator board which then duplicates the WRITE into Shadow RAM. Because Shadow RAM is on the emulator, not on the pod, the Shadow RAM can be read while the application is running at full speed, without slowing the application at all. For more information about Shadow RAM, see the description on page66.

To see a Shadow RAM example for the 8XC251SB, load and run the file titled: “Example\...\timeb.omf”. In the **Config Emuator Hardware** menu, make sure the **Code Mode** is set to **Binary**. Set the Shadow RAM address to 5000, and choose “display shadow as ascii” from the menu. The result will be a clock in Shadow RAM that displays hours, minutes and seconds as shown in Figure 10.



```
5000:
5010: 10:17:21:pm.
5020: Alarm 1!.
```

Figure 10. Shadow RAM Example

Shadow Ram windows display values in all the same formats as any **Data** window. This is controlled with the**Data** menu that is presented when a **Shadow Ram** window is selected.

Usually a **Data** window and a **Shadow Ram** window will display the same value for the same address. This may not be true when the board is first powered, when the emulation is running (because updating the **Data** window is suspended while running at full speed), when displaying addresses that map onto the register addresses, or when a WRITE occurs to a RAM address higher than the top of Shadow RAM. These WRITEs will modify Shadow RAM at a lower (false) address. (See the Chapter 1 section that describes Shadow RAM starting on page66.)

Unlike other types of **Data** windows, **Shadow Ram** windows are updated as the target software writes values into emulation RAM or target RAM. This updating occurs without affecting the emulation. An attempt to scroll a **Data** window while emulating will result in a blank window which will be updated when the emulation stops.

Shadow RAM Size and Addressing

What you see in the Shadow RAM can depend on the size of the Shadow RAM board and the memory segment your program writes to. If you have a 64K Shadow RAM board, only 16 address lines are compared. This means that the segment will be ignored.

For example, if the program writes to 01:5010, the byte will appear at 5010 in the Shadow RAM. If the program then writes to 00:5010, it will overwrite the value previously written to the Shadow RAM (but not necessarily to address areas other than Shadow RAM, depending on memory mapping and target design).

If you have a larger Shadow RAM board, it will record write values for as many address lines as it has available. A one megabyte Shadow RAM board can distinguish 20 address lines, so for the example above, the value written to 01:5010 will not be overwritten by the value written to 00:5010. The Shadow RAM board captures logical addresses, not physical addresses.

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 page26 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.

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[1]. 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 8XC251SB 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

1. The colors used to indicate a breakpoint may be different if you have changed the color scheme as described in the next section.

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.

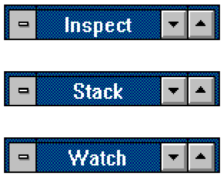
The simplest way to find the first line of source is to reset the controller, click on the **Source** window title bar to select it, and then execute a single step by pressing the F7 key (or by clicking on the **Step** button on the speed bar).

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 "Chapter 3: Trace Board" on page69

Other Windows



Three more child windows used for high level debugging in C are available: the **Stack** window, 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.

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. 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.

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.

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.*

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 first line of the Tool Bar is mainly for emulator functions; the second line is for trace functions. 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 **Pos** button allows pointing the active window to anew position. The **Trace** button changes to stop when the trace board is busy, and stops the trace buffer when the button is clicked. The **Setup** button opens the **Trace Setup** dialog box. The **Trace Beg** button resets the Trace board and starts bus cycle recording according to the conditions set in the **Trace Setup** dialog box.

Help Line

At the bottom of the EMUL251™-PC Windows 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

For information about using the DDE feature, please contact customer support at support@icetech.com.

Menus

The primary means of controlling the debugger, thus the emulation, is through menus. The EMUL251™-PC Windows menus conform to 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 EMUL251™-PC Windows 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

Load code .	`F3`	Load an absolute file.

EMUL251-PC supports many popular compiler object file formats. See the EMUL251-PC price list for more information about supported compilers.

Load default symbols ..

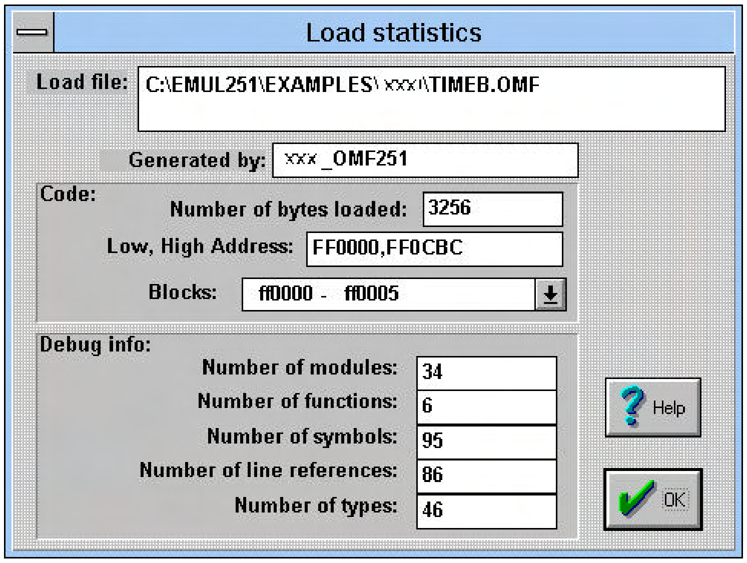
`<Alt>FL`	Load symbols defined by the micronctroller manufacturer.

Selecting this menu item will load the default symbols defined by the microcontroller manufacturer in its manuals. This will enhance the display in **Program** window by converting the addresses of registers into their respective names and bit descriptions.

Note: *Loading default symbols may take as long as 40 seconds on some machines.*

Save code as ..		This feature is not currently implemented.
Remove Symbols	<Alt>FR	Delete all line number and symbolic information.
Show load info ..	<Alt>FS	Display a window describing the object file last loaded including number of variables, address range loaded, etc.

This menu item opens an information box like the one in Figure 11.



Load file: C:\EMUL251\EXAMPLES\ xxx \TIMEB.OMF

Generated by: xxx _OMF251

Code:

Number of bytes loaded: 3256

Low, High Address: FF0000,FF0CBC

Blocks: ff0000 - ff0005

Debug info:

Number of modules: 34

Number of functions: 6

Number of symbols: 95

Number of line references: 86

Number of types: 46

Figure 11: Example Load Statistics

The load information is a summary similar to the one shown when loading completes.

Note: *The compiler information in the* **Generate by** *field may show a different compiler than the one you used. If two compilers use identical file formats, the emulator cannot distinguish one from the other.*

Preferences <ALT>FPControls the way the emulator loads object files.

The preferences dialog box controls the way that object files are loaded.

Exit	<Alt>X	Exiting the EMUL251™-PC Windows 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

Copy to clipboard	`<Ctrl><Ins>`	Copy the text (without formatting or font information) of the entire active window to the clipboard.
User defined symbols	`<Alt>VS`	This item opens a dialog box that lets you select the module from which you can view symbols.
Default CPU symbols	`<Alt>VC`	View and edit memory-mapped registers by name and by the bit.
C call stack ..	`<Ctrl>SS`	Opens a child window that displays the C call stack and passed parameters needed to reach the current Program Counter.
Evaluate ..	`<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".*

Inspect ..	`<Ctrl>I`	Open a dialog box that displays the contents of a single variable, structure, or array in detail.
Add a watch point ..	`<Ctrl>W`	Open a child window that displays groups of variables that is updated every time emulation halts.
Search..	`<Ctrl>S`	Open the Search dialog box.

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 **Source** window is active, you can search for text strings within that file. If the Trace window is active, you can search for any trace record. (See page76 in the Trace chapter for more details.) In all other windows that support searching, the search is for a hex pattern.

Search next	`<Ctrl>X`	The last search defined will be performed again, from the cursor forward.
Search previous	`<Ctrl>P`	The last search defined will be performed again from the cursor backwards.

Run Menu

Step into	`F7`	Execute one instruction, including a jump instruction. If a **Source** window is selected, execute all the instructions for one line of source.

Step over	`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.
Animate ..	`<Ctrl>F7`	Execute instructions continuously and slowly, highlighting each instruction or each line as it is executed.
Go	`F9`	Begin executing instructions from the current PC at full speed until the next breakpoint.
Go to cursor	`F4`	Execute the instructions from the PC to the current cursor position.
Go to ..	`<Ctrl>F9`	Execute the instructions from the PC to the specified address.
Go to return address	`<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.
Reset and Go	`<Alt>R`	Reset CPU and begin execution from reset vector.
Go FOREVER	`<Alt>RF`	Execute instructions from the current PC after disabling all breakpoints.

Break Emulation	`F9`	Suspend execution as if a breakpoint was encountered.
Reset Chip!	`<Ctrl>F2`	Reset CPU without executing any instructions.

Breakpoints Menu

Toggle	`F2`	Disable or enable existing breakpoints.
At ..	`<Alt>F2`	Set a breakpoint by address, line, or line in module.
Setup ..	`<Alt>BS`	Open a breakpoint editing dialog box.
Disable all	`<Alt>BI`	Disable all breakpoints from being active while remaining in the list.
Delete All	`<Alt>BD`	Clear all existing breakpoints.
Break now!	`<Ctrl>C`	Immediately halt the emulation.

Config Menu

Project name ..		Choose a configuration or project from a list of existing projects, or create a new one.
Paths ..	`<Alt>CP`	Sets the default directories for finding load files, source files, and emulator files.
Memory map ..	`<Alt>CM`	Assign memory to either emulation RAM or the target.

Emulator Hardware ..	`<Alt>CE`	Sets the emulator board address, controller type, and configurations.
Miscellaneous ..	`<Alt>CM`	Sets automatic PC & SP reset value, DDE sampling interval, and memory scroll range values.
Full Reset		Reloads on-pod logic & performs reset.
Color ..	`<Alt>CC`	Assign colors to windows.
Trace ..	`<Alt>CT`	Please refer to page80in the Trace Chapter for information about the Trace Config dialog box.
PPA Analyzer	This feature is not yet implemented.	

These next nine 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 do this, either use the **Window** menu, or just click the mouse on any part of the desired window.

Program Menu

Address..	`<Ctrl>A`	Scroll the selected **Program** window to the specified address.
Origin (at program counter)	`<Ctrl>O`	Scroll the **Program** window to display the PC address.
Set new PC value at cursor	`<Ctrl>N`	Set the Program Counter to the address at the cursor.

Module	`<Ctrl>F3`	Open a dialog box that allows quickly scrolling the **Program** window to the start of any module.
Function	`<Ctrl>F`	Open a window listing all the functions in all modules loaded. Selecting one will scroll the **Program** window to the start of that function.
View source window	`<Ctrl>V`	Scroll (or open) a **Source** window to show the source at the current **Program** window cursor.
Toggle breakpoint	`F2`	Enable or disable a breakpoint at the cursor.

Source Menu

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

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

Data Menu

Address..	`<Ctrl>A`	Scroll the selected **Data** window to the specified address.
Edit ..	`<Enter>`	Alter the contents of the highlighted location.
Block move..	`<Ctrl>B`	Move a segment of RAM to another location (in RAM).
Fill..	`<Ctrl>F`	Fill RAM with the specified value or pattern.
Display as..	`<Ctrl>D`	Set the data display mode (ASCII, hexadecimal bytes, long integers, etc. See page29 for the complete list of formats).

Address space ..	`<Alt>DA`	Set the address space for the selected **Data** window.

Register Menu

The only choice in the **Register** menu is **Edit.** 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 3 for all information regarding the Trace board and user interface.

Stack Menu

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

Add ..	`<Insert>`	Open a dialog box for adding a variable to the **Watch** window.
Edit ..	`<Enter>`	Open a dialog box for editing an existing variable in the **Watch** window.
Remove ..	`<Delete>`	Delete the selected variable from the **Watch** window.

Window Menu

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

Open a new program window	`<Alt>IOA`	Open a new (assembly code) window.
Open a new source code window	`<Alt>IOS<Enter>`	Open a **Source** window.
Open a new data window	`<Ctrl>IOM`	Open a **Data** (memory dump) window.
Open a new registers window	`<Alt>IOR`	Open a new Registers window.
Open a new special registers window	`<Alt>IOSS<Enter>`	Open a new **Special Registers** window.
Open a new trace window	`<Alt>IO`	Open a new **Trace** window.
Open a new watch window	`<Alt>`	Open a new **Watchpoints** window.
Toggle help line	`<Alt>IT<Enter>`	Turn on or off text at the bottom of the EMUL251™-PC Windows window.
Repaint	`<Ctrl>R`	Repaints the screen.
Tile windows	`<Alt>ITT<Enter>`	

		Resize and arrange the windows within the EMUL251™-PC Windows application.
Cascade windows	`<Alt>IC<Enter>`	
		Resize and overlap the windows within the EMUL251™-PC Windows application.
Arrange Icons	`<Alt>IA`	Line up any closed EMUL251™-PC Windows icons at the bottom of the main window.
Zoom	`F5`	Expand selected window to fill the EMUL251™-PC Windows window.
Next window	`<Ctrl>F6`	Change the currently selected (highlighted) window.
Close	`<Ctrl>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. The box also displays other version numbers, such as Flex, PRU and C PAL. 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 eight primary child windows created by EMUL251™-PC Windows: **Program** windows, **Data** or Memory windows, **Inspect** windows, **Source** windows, a **Registers** window, a **SpecialRegs** window, **Trace** windows (even if you have no Trace board), **Watch** windows, and a **Call Stack** window. 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 EMUL251™-PC Windows 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

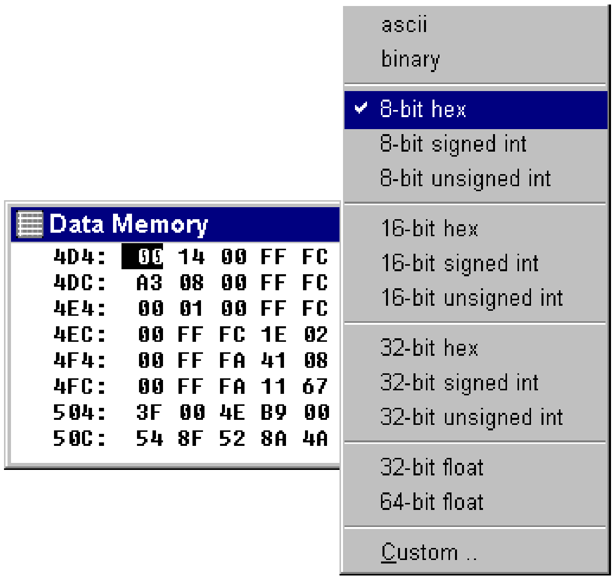
The **Registers** window displays the CPU registers. All registers are displayed in hexadecimal notation. Clicking anywhere in the **Registers** window will select that window (make it the active window) and right-clicking brings up the **Registers** menu. The only operation supported in the **Registers**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

The menu also includes a display feature 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

Use **Data** windows to examine or modify emulation or target memory directly. EMUL251™-PC Windows uses the controller to read and write RAM, so the **Data** window cannot be updated while the emulation is running. If the application attempts to repaint the window, asterisks will be displayed until the next time the controller breaks emulation.

Data can be displayed or modified in a variety of formats as shown in Figure 12. Keep adding new windows for each display format and starting address.



```
4D4:  00 14 00 FF FC
4DC:  A3 08 00 FF FC
4E4:  00 01 00 FF FC
4EC:  00 FF FC 1E 02
4F4:  00 FF FA 41 08
4FC:  00 FF FA 11 67
504:  3F 00 4E B9 00
50C:  54 8F 52 8A 4A
```

Figure 12: Data Display Format

Selecting any **Data** window displays the **Data** menu which supports filling memory, jumping the selected window to a specific address, moving blocks of data, 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 .

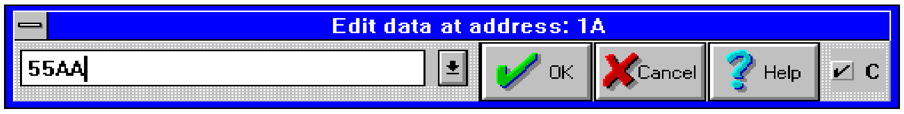


Figure 13: Editing Memory with a Data Window

To avoid confusion, 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 and their type is irrelevant.

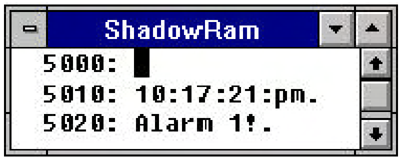
If you display the data in bytes, only a byte will be written to memory for each update. In other words, updating one byte uses a single bus cycle that is one byte wide.

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 “abcd” 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).

Shadow RAM Windows

A **Shadow Ram** window is a type of **Data** window. It displays the contents of Shadow RAM on the emulator board in any of the same formats as any **Data** window. A **Shadow Ram** window is updated even though the application is running at full speed. This is made possible by the logic on the pod that captures the data value and address for every bus WRITE cycle executed from user code. Those two values are sent to the emulator board which then duplicates the WRITE into Shadow RAM. Because Shadow RAM is on the emulator, not on the pod, the Shadow RAM can be read while the application is running at full speed, without slowing the application at all. For more information about Shadow RAM, see the description on page66.

To see a Shadow RAM example for the 8XC251SB, load and run the file titled: "Example\...\timeb.omf". In the **Config Emuator Hardware** menu, make sure the **Code Mode** is set to **Binary**. Set the Shadow RAM address to 5000, and choose "display shadow as ascii" from the menu. The result will be a clock in Shadow RAM that displays hours, minutes and seconds as shown in Figure 10.



```
5000: 
5010: 10:17:21:pm.
5020: Alarm 1!.
```

Figure 14. Shadow RAM Example

Shadow Ram windows display values in all the same formats as any **Data** window. This is controlled with the**Data** menu that is presented when a **Shadow Ram** window is selected.

Usually a **Data** window and a **Shadow Ram** window will display the same value for the same address. This may not be true when the board is first powered, when the emulation is running (because updating the **Data** window is suspended while running at full speed), when displaying addresses that map onto the register addresses, or when a WRITE occurs to a RAM address higher than the top of Shadow RAM. These WRITEs will modify Shadow RAM at a lower (false) address. (See the Chapter 1 section that describes Shadow RAM starting on page66.)

Unlike other types of **Data** windows, **Shadow Ram** windows are updated as the target software writes values into emulation RAM or target RAM. This updating occurs without affecting the emulation. An attempt to scroll a **Data** window while emulating will result in a blank window which will be updated when the emulation stops.

Shadow RAM Size and Addressing

What you see in the Shadow RAM can depend on the size of the Shadow RAM board and the memory segment your program writes to. If you have a 64K Shadow RAM board, only 16 address lines are compared. This means that the segment will be ignored.

For example, if the program writes to 01:5010, the byte will appear at 5010 in the Shadow RAM. If the program then writes to 00:5010, it will overwrite the value previously written to the Shadow RAM (but not necessarily to address areas other than Shadow RAM, depending on memory mapping and target design).

If you have a larger Shadow RAM board, it will record write values for as many address lines as it has available. A one megabyte Shadow RAM board can distinguish 20 address lines, so for the example above, the value written to 01:5010 will not be overwritten by the value written to 00:5010. The Shadow RAM board captures logical addresses, not physical addresses.

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 page26 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.

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[1]. 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 8XC251SB 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

1. The colors used to indicate a breakpoint may be different if you have changed the color scheme as described in the next section.

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).

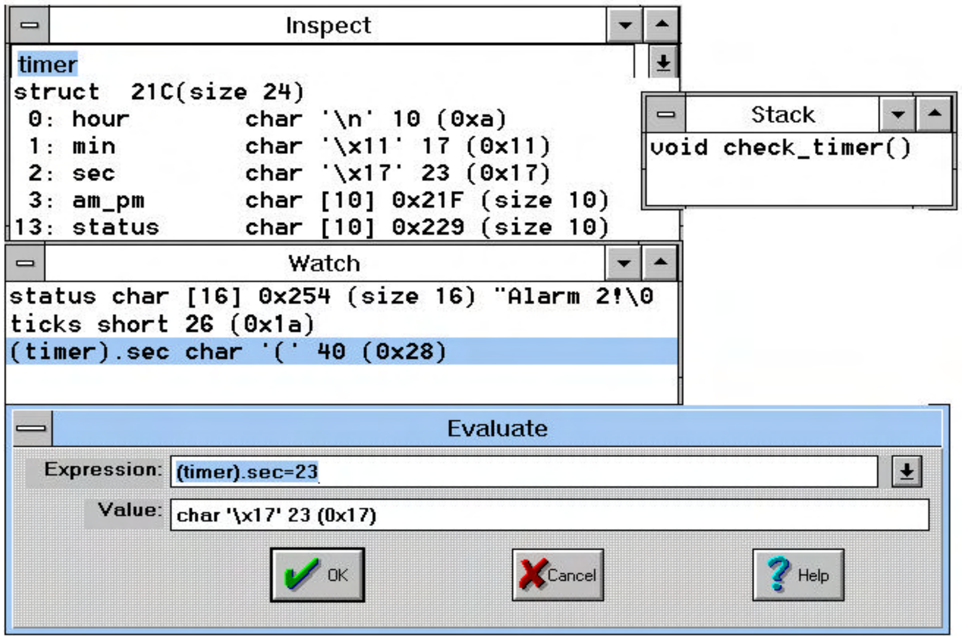
When the **Program** window is selected, a single step means a single opcode. The same is true for animated execution: a pause occurs after every opcode is executed. When the **Source** window is selected, a single step means a single source line. Animation will execute faster when the **Source** window 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 "Chapter 3: Trace Board" on page69

Other Windows

Three more child windows used for high level debugging in C are available: the **C call stack** window, 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.



```
timer
struct  21C(size 24)
 0: hour       char '\n' 10 (0xa)
 1: min        char '\x11' 17 (0x11)
 2: sec        char '\x17' 23 (0x17)
 3: am_pm      char [10] 0x21F (size 10)
13: status     char [10] 0x229 (size 10)
```

```
void check_timer()
```

```
status char [16] 0x254 (size 16) "Alarm 2!\0
ticks short 26 (0x1a)
(timer).sec char '(' 40 (0x28)
```

Expression: (timer).sec=23
Value: char '\x17' 23 (0x17)

Figure 15: 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. 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.

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.

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.*

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 first line of the Tool Bar is mainly for emulator functions; the second line is for trace functions. 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 **Pos** button allows pointing the active window to anew position. The **Trace** button changes to stop when the trace board is busy, and stops the trace buffer when the button is clicked. The **Setup** button opens the **Trace Setup** dialog box. The **Trace Beg** button resets the Trace board and starts bus cycle recording according to the conditions set in the **Trace Setup** dialog box.

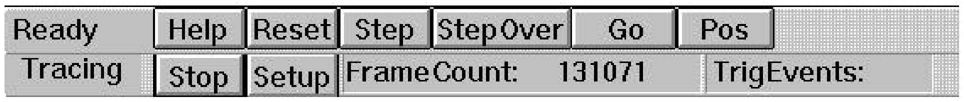


Figure 16: The Tool Bar

Help Line

At the bottom of the EMUL251™-PC Windows 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

For information about using the DDE feature, please contact customer support at support@icetech.com.

Chapter 2: Emulator Board

General Description

The EMUL251™-PC emulator board is designed to support boards for different members of Intel's MCS 251 family of microcontrollers. The available pod board is the POD-251SB As Intel introduces other members of the N87C251SB family of microcontrollers, corresponding pod boards may be introduced and maybe supported by EMUL251™-PC. Email sales@icetech.com for the current list of available pod boards and supported controllers.

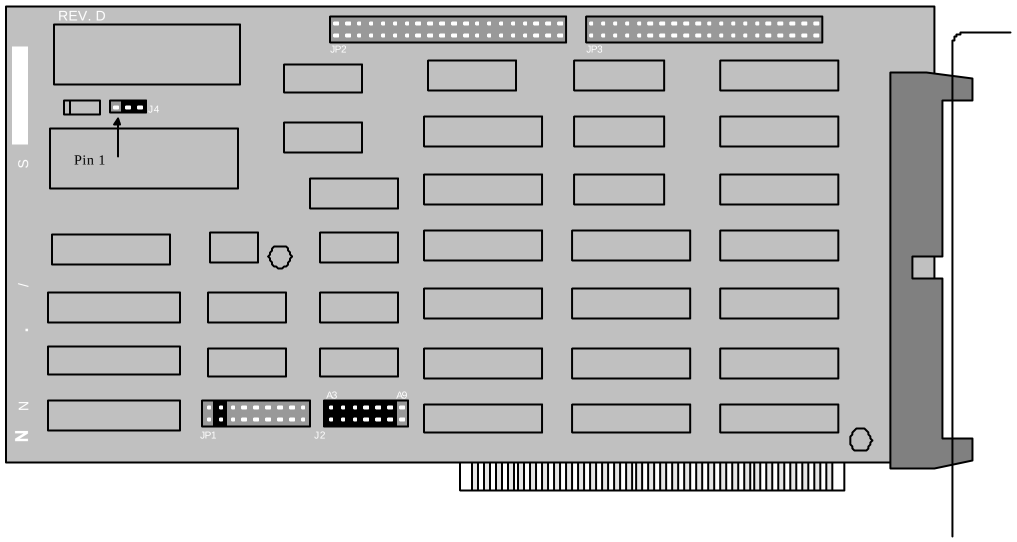


Figure 17: Rev. D Emulator board

The EMUL251™-PC board is an 8-bit PC Card that fits into any 3/4 length slot. It contains 64 kilobytes, 256 kilobytes, or 1 megabyte of Shadow RAM, bus interface logic, Trace board support logic, and the logic needed to communicate with the pod. The jumpers on the emulator board control the address used to communicate with the Host PC. This is described in more detail below.

Detailed Installation Instructions

Setting the I/O address jumpers -- J2

The EMUL251™-PC requires 8 consecutive I/O addresses from the PC 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 J2 . These addresses must not conflict with any other I/O device.

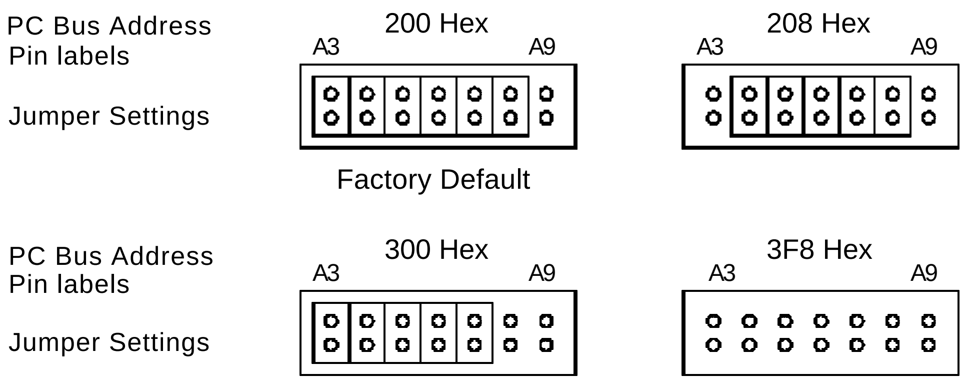


Figure 18: Emulator Header J2

Each pair of pins in J2 represents one 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 two pins represents a 0 in the address. A pair of pins with no jumper represents a 1. Above are four examples where the Least

Significant Bit 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 ribbon connector on the right.

Header JP1

This header controls features not implemented on EMUL251™-PC. Leave the jumper in the default position, between pins 3 and 4.

Header J4

The next paragraph only applies to emulator boards with 1 Megabyte of Shadow RAM. Emulators with less than 1 Megabyte of Shadow RAM must leave the jumper between pins 2 and 3.

If your emulator board has 1 Megabyte of Shadow RAM, short pins 1 and 2. This is the default position for 1 Megabyte Shadow RAM emulator boards.

Do not change the jumper for header J1.

After these jumpers are set as described above, with the PC power off, remove the PC cover, insert the emulator board into any free slot, close the PC cover, connect the ribbon cable to the emulator board, connect the pod to the ribbon cable, connect the pod to the target (if present), turn on the PC, apply power to the target if present, start Windows, and start the EMUL251™-PC application.

Warning: Always turn off the PC before connecting or disconnecting the ribbon cable to the emulator or pod board. ot doing so could damage either the CPU, the emulator.

Initial Software Configuration

The Windows software is used for all EMUL251™-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 anerror message. After you have set up the initial processor type and I/O addresses, you can start the emulator application. You are done with installation.

Shadow RAM

The EMUL251-PC emulator board contains either 64 kilobytes, 256 kilobytes, or 1 megabyte of static RAM used to "shadow" or duplicate the contents of the target RAM. Every time the CPU generates a WRITE bus cycle while running the target application, the pod captures the address/data pair and the emulator board writes that data to the same address in Shadow RAM. The EMUL251-PC application can simultaneously read Shadow RAM. This allows the software to display values written by the application without interrupting emulation.

Note: *Shadow RAM will capture external data writes while you are running your code. Shadow RAM will not capture the bus activity when you are interactive with the EMUL251-PC user interface. Loading code, filling memory, and editing registers will not update Shadow RAM.*

Please note that if the emulator board has 64 kilobytes of Shadow RAM and the application data area RAM is larger, the references to addresses above 64 kilobytes (00:FFFF Hex) will overlay the addresses in the lowest 64K addresses. The Shadow RAM address logic strips off the bits above bit 15. That is, the Shadow RAM address 100 Hex will be modified by WRITEs to application RAM addresses 100 Hex, 01:0100 Hex, FF:0100 Hex, etc. This is true for emulation RAM, or RAM on the target, or even memory-mapped I/O devices. If this happens, the EMUL251-PC application may display incorrect information and send incorrect data to other applications through the DDE interface. Ordering an emulator board with 256 kilobytes of Shadow RAM will eliminate this problem for all 8XC251SB applications.

Chapter 3: Trace Board

Introduction

EMUL251™-PC Windows needs RAM to record a history of the data used and instructions executed. The trace board contains this RAM. The emulator board has the logic and connectors necessary to support a trace board. The trace board is a full-length ISA-style bus card. It may occupy any 16-bit or 8-bit slot as long as the two ribbon cables can reach from the emulator card to the trace card. When inserted into a 16 bit slot, it connects with the additional power and ground lines in the other connector. The card includes 104 bits of RAM for each trace record. Trace boards are available with three sizes of trace memory: 32K, 128K, or 512K records.

Detailed Installation Instructions

The trace board includes three connectors on the back for inputting and outputting signals. Figure 21 on page 72 shows the DB25 and the two BNC connectors and how they are wired.

I/O address

Like the emulator, the trace board uses 8 consecutive I/O addresses for communicating with the PC. At the factory, the jumpers on the card are set so that the trace board will use the I/O addresses that start at 208 Hex. Either confirm that these addresses are available on your PC, or find 8 consecutive free addresses and set the address jumpers on header J1 accordingly. (See the examples in Figure 19 on page70.)

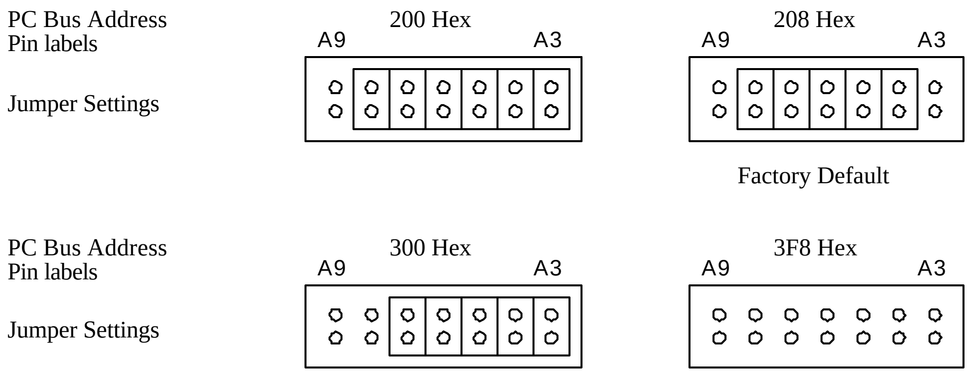


Figure 19: Trace Board I/O Address Header J1

Note: *On the trace board, A3 is on the right; on the emulator board, A3 is on the left.*

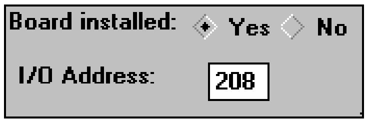
Once the trace board address jumpers are set[1], turn off the PC power, remove the cover, and slide the board into the chosen slot. Make sure that the board is fully inserted. There are two identical ribbon cables and although the connectors are not keyed, due to the length and shape of the cables, it is almost impossible to deliberately attach both cables to the incorrect connector. Just make sure that the pins are fully inserted into the connectors, there are no exposed pins, there are no twists in either cable, and the cables do not cross. Be absolutely sure that the connectors are not offset vertically or horizontally. The most common error is to insert only one row of pins into the connector which could damage either of the boards. Double check all four connectors for any exposed pins before continuing.

1. These instructions assume that the emulator board is already installed and operational. If this is not hte case, install the emulator and pod boards first before installing the trace board.

Once the ribbon cables are attached, close the PC cover, turn on the PC power, start Windows, then start the EMUL251™-PC Windows program.

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 20 shows the normal default settings for an installed trace board.



Board installed: [✓ Yes] No
I/O Address: 208

Figure 20: Trace Board I/O Address Setting

Also critical is **I/O address:**. Be sure to set this field to the same base address set by the jumpers in header J1. If the trace board is installed and the trace configuration dialog box opens automatically when the emulator application is started, then the I/O address is probably set to a different value than the jumpers on header J1.

External Inputs and Controls

The Trace board records eight external digital inputs with every bus cycle. These signals are input through the 25 pin D connector on the back of the Trace board. To simplify providing these signals to the Trace board, use the color coded set of micro-clips provided with the Trace board. (The 25 conductor ribbon cable included with the Trace Board is wired straight through and may be used to extend the reach of the micro-clips.)

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

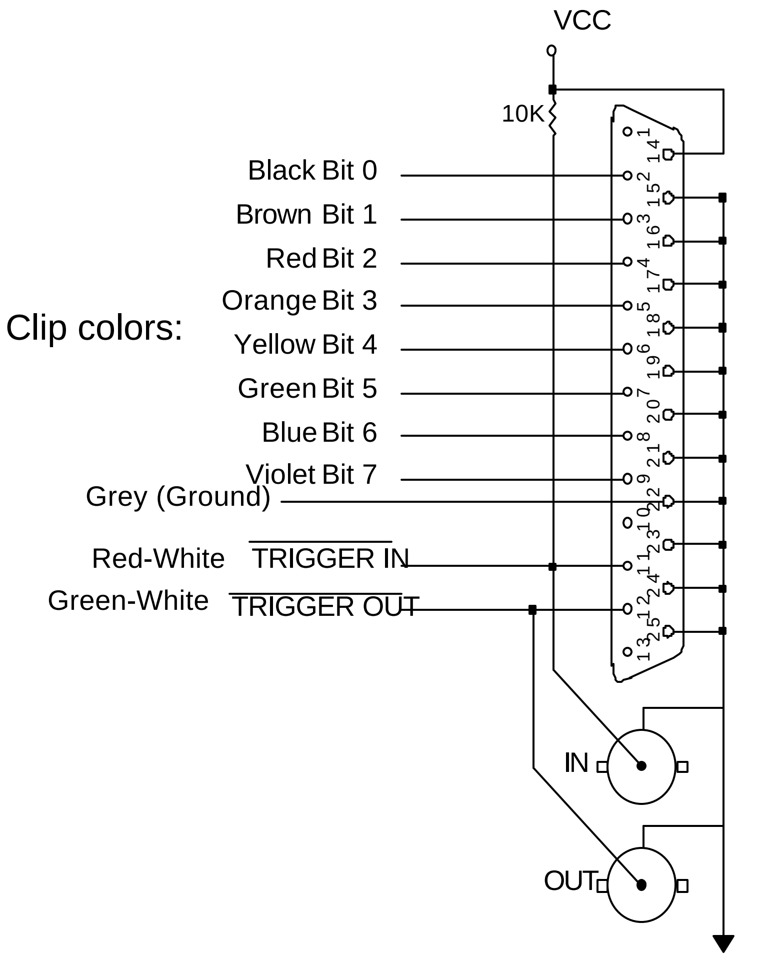


Figure 21: Trace Board Connectors

Two of the micro-clips duplicate the trigger controls found in the BNC connectors[1]: $\overline{\text{TRIGGER IN}}$ and $\overline{\text{TRIGGER OUT}}$.

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

The $\overline{\text{TRIGGER IN}}$ micro-clip currently functions as a trigger inhibit. 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.

It is intended that a future software version will give a choice of Inhibit Trigger or Trigger for this input. When the feature is implemented, you will be able to click on the **TRIG** radio button, and the transition to ground on the $\overline{\text{TRIGGER IN}}$ line will cause a trigger on the trace board and stop trace recording. Like a trigger caused by a bus cycle, this external trigger will cause a hardware break if the **Break Emulation? Yes, on trig** check box is checked.

Introduction to Tracing

A trace history is a time ordered recording of execution (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. An emulator and trace must have some way to deal with the effects of the instruction pipeline. The pod board for EMUL251™-PC Windows includes pipeline decoding and

1. Some trace boards do not have BNC connectors. If your board does not, and you would like them, contact customer support at support@icetech.com.

discriminates opcode fetches that are not executed. As a result, the trace board can collect the trace records as though the pipeline did not exist.

Tracing starts automatically every time emulation starts. Even single-stepping will turn on the trace recording during that step. Clicking on the **Trace** 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** button changes to the **Stop** 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 **Stop** button.

Once emulation has started and bus cycles are "being recorded," every execution cycle and external read or write 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. As long as trace recording continues, records will be added to the buffer. 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 executed for the fourth time. Triggers can start and stop trace buffer recording, and can cause 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 breakpoints, see the section titled “Trace Setup Dialog Box” on page80.

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.

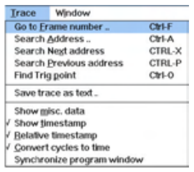
ipeline Effects

When a jump occurs and the pipeline is flushed, some instructions are fetched but not executed. These fetched but ignored instructions are seen by the pod board when they are fetched but the instruction tracker logic only allows the trace to record those instructions which are actually executed.

Bus Cycle Order

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

Trace Menu



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.

Go to 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 22, to get the desired address, then searches from the beginning in the frame buffer for the first record that contains the specified address.

{bmc manual\graphics\tr_srch.bmp}

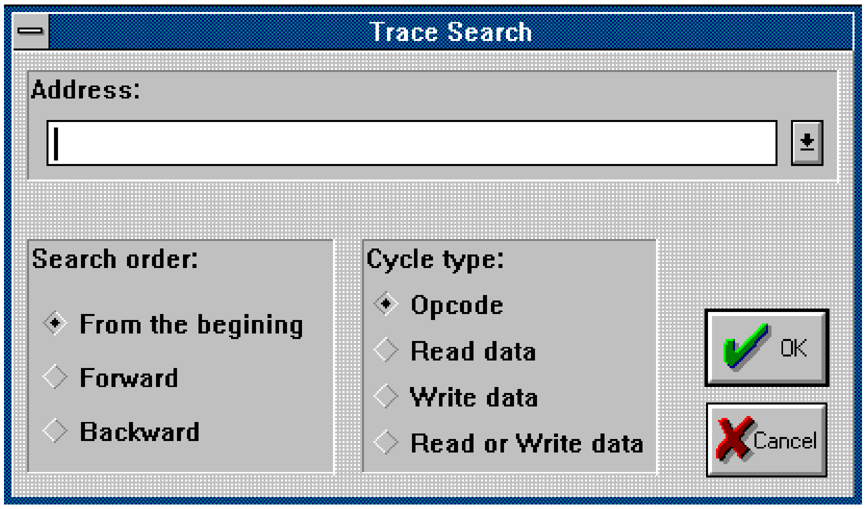


Figure 22: Trace Search Dialog Box

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 DB25 connector, 8 processor status signals, and 8 signals on the TRACE header on the pod with each trace record (recorded on third clock of internal bus CPU cycle).

Of the three bytes, the left most shows the bits input from the DB25 connector. Figure 21 on page page72 shows how the pins of the DB25 are used and which color micro-clip is assigned to which bit in the display.

The other two bytes are sometimes useful to Customer Support staff but generally not 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

Many trace buffers, including many of our trace buffers for other emulator families, capture bus fetch activity rather than processor execution. The trace for EMUL251-PC, however, shows execution timing.

The timestamp shown in the trace is that of the execution of an instruction, regardless of the timing of the fetches. For example, two consecutive no-operation (NOP) instructions could be fetched at the same time because the MCS 251 can fetch two bytes at a time from internal memory. The trace will show the individual execution timestamp of each of the NOP’s, not the fetch timing, since each of the NOP’s execute at a different time.

Relative timestamp

The timestamp is a 40-bit integer which is large enough to uniquely number all frames in a period of many hours. 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 cycles at addresses FF:00E7 to FF:00EF 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 49 milliseconds long. Rather, it was 49 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 -2.

Also note that the timestamp for the first frame shown, frame -4, shows the time since the very first frame in the buffer, not the time to the previous frame. This is why the timestamp show 53 milliseconds, not 49 milliseconds, as is shown for the other times through the loop.

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). 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.

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 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 trace 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 page69.

Trace Memory

This box, shown in Figure 23, 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.

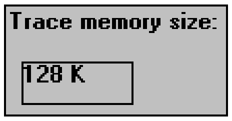

Trace memory size:
128 K

Figure 23: 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.*

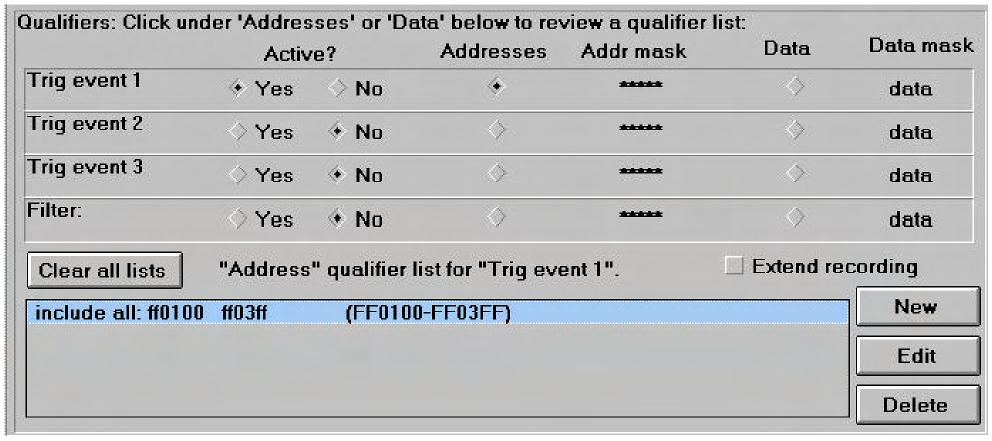
The **Last trigger repeat count** field contains the number of times the highest numbered trigger *conditions* must be met before the trigger itself occurs. If a trigger condition is set for an opcode fetch at address FF:0400 and the **Last trigger repeat count** is set to 10, the first 9 fetches from address FF:0400 will be counted and the trigger will occur when that opcode is fetched for the 10th time.

This count may be as high as 65536, 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 page80 for information about setting and changing the trigger conditions.

Figure 24 shows that trigger 1 has one condition and it will be met by "any" kind of bus cycle that includes any address from FF:0100 to FF:03FF.



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

	Active?		Addresses	Addr mask	Data	Data mask
Trig event 1	◆ Yes	◇ No	◆	******	◇	data
Trig event 2	◇ Yes	◆ No	◇	******	◇	data
Trig event 3	◇ Yes	◆ No	◇	******	◇	data
Filter:	◇ Yes	◆ No	◇	******	◇	data

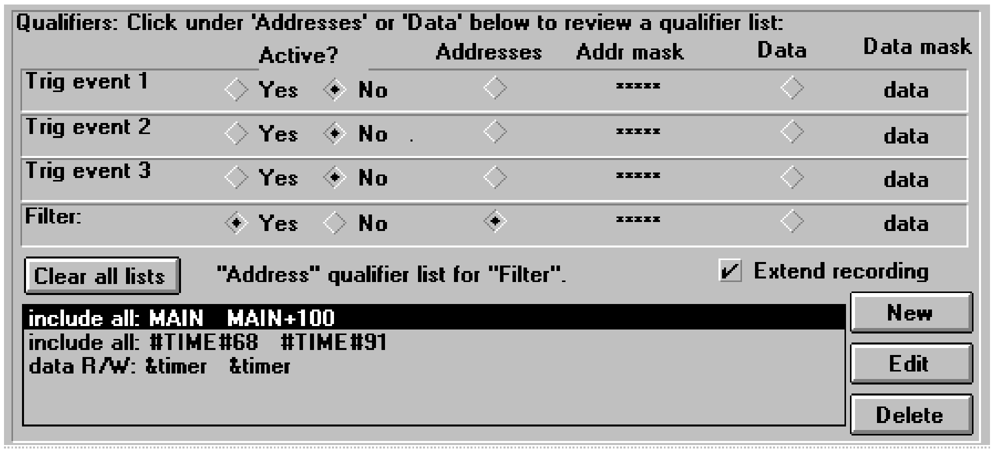
"Address" qualifier list for "Trig event 1".

☐ Extend recording

```
include all: ff0100   ff03ff          (FF0100-FF03FF)
```

Figure 24: Trigger on anything

With **Filter mode: Normal**, bus cycles are recorded until a trigger stops the recording. Recording execution 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 25, the trace board will record everything from the start of main to **`MAIN`** + 100H, the opcode fetches between line 81 and 87 from the module **`TIME`**, and only the data bus cycles (read and write cycles) to and from address **`timer`** (37H).



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

	Active?		Addresses	Addr mask	Data	Data mask
Trig event 1	Yes	[✓ No]	◇	*****	◇	data
Trig event 2	Yes	[✓ No]	◇	*****	◇	data
Trig event 3	Yes	[✓ No]	◇	*****	◇	data
Filter:	[✓ Yes]	No	◆	*****	◇	data

"Address" qualifier list for "Filter". [✓ Extend recording]

```
include all: MAIN   MAIN+100
include all: #TIME#68   #TIME#91
data R/W: &timer   &timer
```

Figure 25: 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 25 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 91 in the module **TIME**. Line 91 is the closing brace in the C source file from the example in Figure 25. 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 91 with a small post trigger sample count. Line 91 will be executed, the trigger will occur, then a few frames will be captured from after line 91, 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 25, every time the trace board captures an opcode fetch from line 91, 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 91. When you use **Extend recording**, the trace board may contain up to 32K instances of where line 91 was executed and where the program counter went afterwards. Now THAT's efficient.

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.

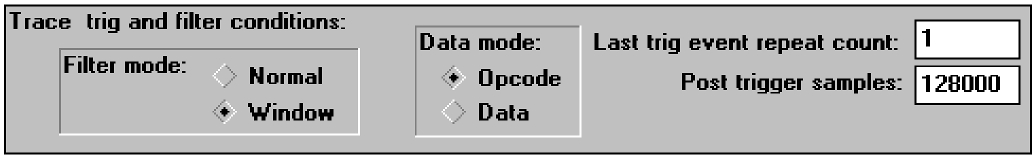
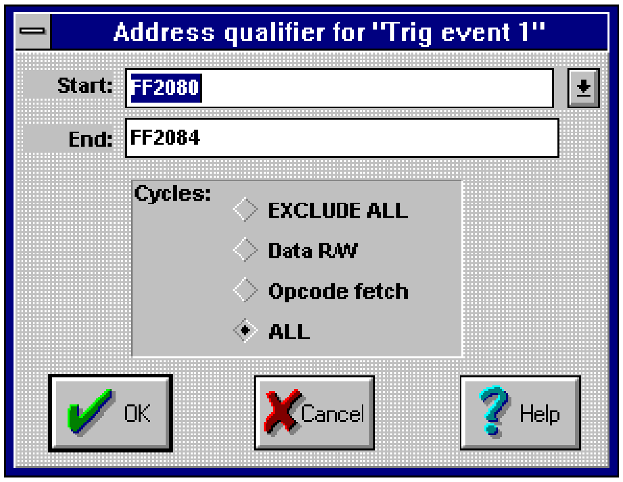

Trace trig and filter conditions:
Filter mode: ◇ Normal [✓ Window]
Data mode: [✓ Opcode] ◇ Data
Last trig event repeat count: 1
Post trigger samples: 128000

Figure 26: 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 27. 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 27 shows how to monitor a single address: put that address in both the **Start:** and **End:** range fields.



Address qualifier for "Trig event 1"

Start: FF2080

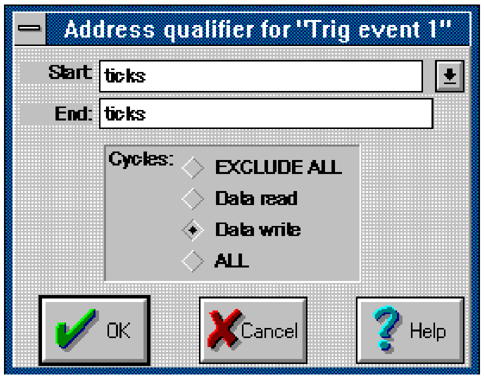
End: FF2084

Cycles:
EXCLUDE ALL [ ]
Data R/W [ ]
Opcode fetch [ ]
[✓ ALL]

Figure 27: 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 28.

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



Address qualifier for "Trig event 1"

Start: ticks

End: ticks

Cycles:
EXCLUDE ALL [ ]
Data read [ ]
[✓ Data write]
ALL [ ]

Figure 28: 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.

Trigger on data values

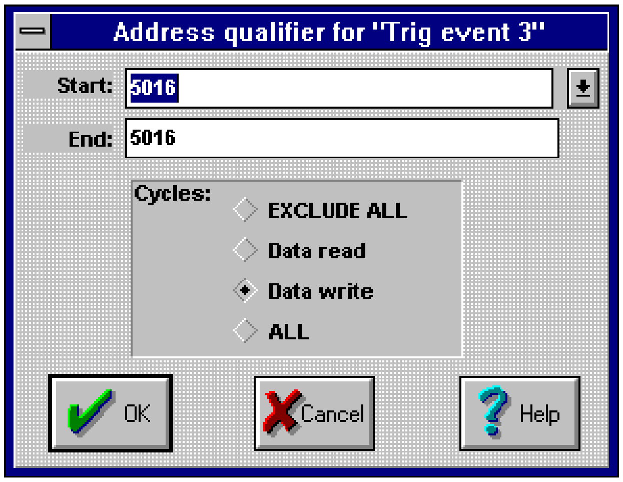
If you have a Data Trace Board (DTR) you can trigger and filter on both the address and the value on the data bus. not only on fetch, read and write on addresses, but also on data values.

Why trig on a data value?

Sometimes problems in program flow only occur when an address contains a particular value. Triggering on that address AND that value lets you ignore hundreds or even thousands of writes to that address that do not cause a problem. This kind of triggering makes it easy to find when that address got that value.

For example, to find out when address 5016 got the value 33 choose **Data mode: Data** under **Trace trig and filter conditions**, click on button in the **Addresses** column for **Trig event 1**, delete all the other address conditions for Trigger 1, add the new address range that starts and ends with 5016 and only triggers on data write cycles. Now click on the button for **Trig event 1** in the **Data** column, remove all other value conditions for Trigger 1, and click on the **New** button to open the dialog box shown in Figure 29.

{bmc manual\graphics\address.lbmp}


Address qualifier for "Trig event 3"

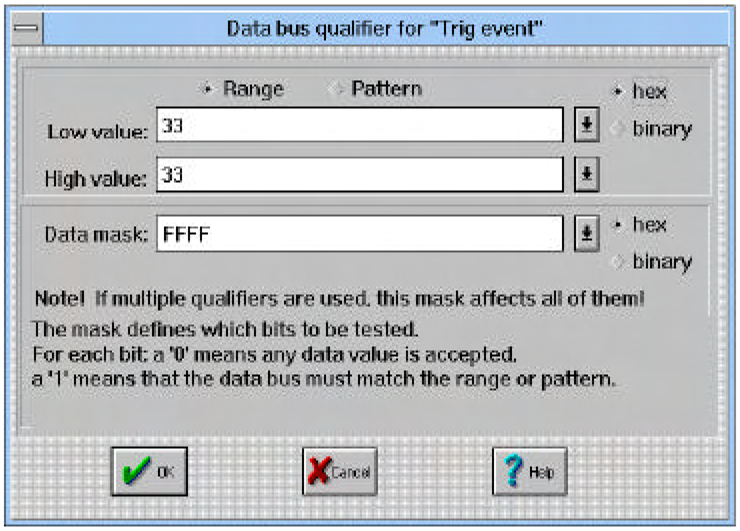
Start: 5016

End: 5016

Cycles: EXCLUDE ALL [✓ Data write] Data read ALL

Figure 29: Trig Address Qualifier

To activate the trigger on a data value, check the box **DATA** in the same TRIGGER EVENT as you have the address trigger. The address trigger and the data trigger are ANDed together. If you click on the **NEW** button, you will activate the window for **Data bus qualifier for "Trig event x"**. If you want to trigger on one specific data value, you should choose **Pattern**, (see Figure 30). If you want to trigger on a value within a range, you should choose **Range** (see Figure 30).


Data bus qualifier for "Trig event"

[✓ Range] Pattern [✓ hex] binary

Low value: 33

High value: 33

Data mask: FFFF [✓ hex] binary

Note! If multiple qualifiers are used, this mask affects all of them!
The mask defines which bits to be tested.
For each bit: a '0' means any data value is accepted,
a '1' means that the data bus must match the range or pattern.

Figure 30: Data Bus Qualifier

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).

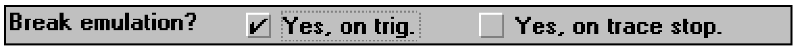

Break emulation? [✓ Yes, on trig.] Yes, on trace stop.

Figure 31: 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 4: Pod Boards

General Pod Features

The pod is the part of the emulator system that interfaces to your target. It has an emulation ("bondout") chip, RAM, a crystal, and logic to glue all those pieces together.

How It Works

Clicking on the **Reset** button tells the emulator to pull the RST line high, resetting the controller. When the RST line is 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 one of the **Step** buttons, the **Go** button, or select one of the **Step, Animate** or **Go** items from the **Run** menu.

When you click on the **Break** button, the emulation stops, and the pod executes monitor code which is not visible to the user.

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 may not be able to either.

Indicator Lights

The pod boards contain 4 status lights and two configuration lights. The configuration lights are surface-mounted LEDs labeled PGA not Prgmd and PRU not Prgmd. These are initially lit when the computer is turned on. After software configures the pod logic, the LEDs will go off. They will then normally remain unlit as long as the computer power is on.

The status LEDs are labeled Run, Reset, Idle, and Power Down. The Reset light will be lit when the emulator or the target hardware reset the pod microcontroller. The Run light will be lit whenever the controller is executing user code (as opposed to monitor code). The Idle light will be lit when the user code puts the microcontroller into idle mode during emulation. Likewise, the Power Down light will be lit when user code puts the microcontroller into the power-down mode during emulation.

Pod Resources

The pod board has many resources and your target may also have the same resources. Jumpers and software settings allow you to choose whether to use pod or target resources.

Power to the microcontroller can come from the emulator or from the target. The Stand Alone jumper supplies power to the microcontroller when no target system is connected to the pod. Always remove the Stand Alone jumper before connecting the pod to a target system.

The Stand Alone jumper must be installed to oprate the pod without a target system.

A target system must have its own power supply; supplying power from the pod to a target system is not supported.

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.

EMUL251-PC Windows uses a special ICE-ready bondout technology controller to emulate the N87C251SB . This special chip has special internal capabilities that allow the emulator to provide its features. 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 emulator software allows you to map any address either to the pod or the target. For convenience, there is a button that maps all RAM to the pod with one click of the mouse button.

Internal Addressing and Single Chip Mode

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 $\overline{EA}$ pin high during RESET will configure the 83C251SB or 87C251SBfor 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 ROM in the emulated controller. The pod contains a built-in Port Replacement Unit that reconstructs the low speed I/O ports for the target. Therefore, the pod can emulate both single-chip and external memory designs.

Chapter 5: POD-251SB

POD-251SB, Rev. C

Introduction

This pod board contains an Intel N87C251SB bondout microcontroller chip (suitable for emulating the Intel 8XC251SB), a 16 MHz crystal, 256 kilobytes emulation RAM for instructions and data, circuits for driving the cable bus, two PLDs, three CPLDs and two FPGA chips.

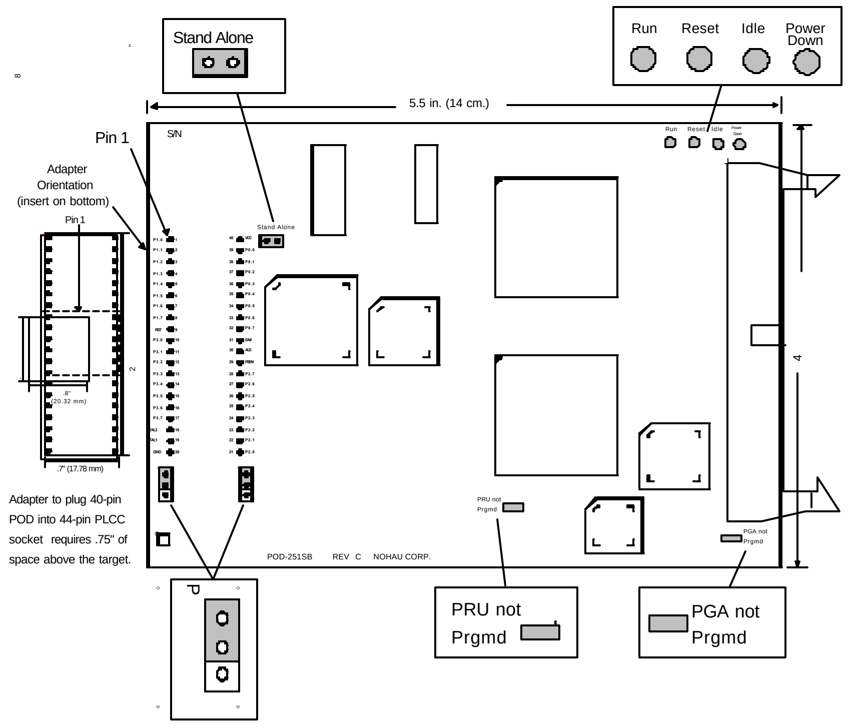

Stand Alone
Run
Reset
Idle
Power Down
5.5 in. (14 cm.)
Pin 1
S/N
Adapter Orientation (insert on bottom)
Pin 1
.8" (20.32 mm)
.7" (17.78 mm)
Adapter to plug 40-pin POD into 44-pin PLCC socket requires .75" of space above the target.
PRU not Prgmd
PGA not Prgmd
POD-251SB REV C NOHAU CORP.
4

Figure 32: POD-251-SB, Rev. C

Identifying the Pod

The pod has printing at the edge of the board. The revision is printed between the words "POD-251SB" and "NOHAU CORP". This section covers REV C. If your pod is REV D, turn to the section for that revision on page100.

Dimensions

The pod board is 5.5 inches by 4 inches (14 cm. by 10.1 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.

Theory of Operation

The 87C251 microcontroller supplied on the POD-251SB, Rev. C, is an ICE-Ready device. This allows a special mode in which the pod can execute from its EPROM area at full internal access speed without having its internal EPROM programmed. Users can download code to the pod's EPROM area and experiment with different selections in the configuration bytes CONFIG0 and CONFIG1 without using an EPROM programmer.

The POD-251SB, Rev. C, allows users to accurately set breakpoints at any instruction address and map any memory location to the target or to the POD-251SB, Rev. C's emulation memory. The pod does not use any stack area or other user resource, or impose any restrictions on the user's hardware or software design.

For proper operation with a target, the page mode select button in the **Config Emulator Hardware** menu should be selected (or not selected) depending on design of the 8XC251 target. It may be necessary to preconfigure this page mode option in stand-alone mode to “initialize” the EMUL251.ini file to match that of the target.

When changing from stand-alone to target operation, check the "stand-alone" jumper XJ1. Also, verify the source code or binary code compatible selection in the **Config Emulator Hardware** menu. This selection should match the compiler / assembler mode option (source mode or binary mode) for any file that is downloaded for execution.

Important: After making changes in the Config Window, when asked "Implement Reset Now?" you must click on the YES button for the change to take effect.

All tracing for the POD-251SB, Rev. C, is execution trace, which records only those instruction addresses that were actually executed. External data transferred, as well as P3, P2 and P0 port transfers are also recorded in the trace.

Interface

The POD-251SB, Rev. C, can interface directly to a 40-pin DIP target socket or through an adapter to a 44-pin PLCC target socket. Clip-over "ONCE mode" adapters for 40 and 44 pin targets may be introduced but were not available when this manual version was written.

Emulation Memory

The POD-251SB, Rev. C, has 256K bytes of emulation memory. Logical to physical translation is controlled by the RD1 and RD0 bits in the CONFIG0 register. The **`CONFIG`** register values do not interfere with the user's code. These configuration bits are accessed by the emulator hardware menu selection under the **Config** selection of the software interface. The emulation hardware automatically configures emulation memory spaces based on RD1 and RD0 as described in the External Memory Interface chapter of Intel's 8XC251SB Embedded Microcontroller User's Manual.

Address ranges can be mapped into target memory with a precision of 2 bytes for non-page mode, and 256 bytes for page mode. The pod's 16K internal EPROM is always mapped to emulation memory when not in external mode. Although the pod has special protection circuitry built in, it is still preferable to turn on the PC before applying power to the target and turn off the target power before turning off the PC power.

Warning: **Always remove the** Stand Alone **jumper on the pod before connecting the pod to a target system.**

POD-251SB Headers

JP2, JP3, XJ1

JP1 and JP2 determine on-pod or on-target crystal selection.

JP1 and JP2 each have two jumper positions: TARGET and POD. When set in the TARGET position, the pod controller will receive the clock signal from the target crystal. With both in the POD position, the bondout controller on the pod will use the crystal on the pod.

Note: *When the clock jumpers are in the POD position, the XTAL1 and XTAL2 signals from the pod are disconnected from the target.*

External Drive Clock

This section only applies if you are driving a clock signal into the XTAL1 input of the 8XC251SB. If you are using both pins with a crystal or resonator, this section does not apply.

Due to a layout error on this revision of the pod, the XTAL1 and XTAL2 pins are swapped. If you are driving the 8XC251SB with a clock signal, the signal must be crossed over to the other pin.

One way to do this is to cross the signal over on the pod.

1. Remove both jumpers JP2 and JP3 (both labeled "POD/TARGET").

2. Add a short wire from JP3 pin 1, labeled "TARGET" to JP2 pin 2, the middle pin.

3. This swap is only necessary with an external driving clock signal, not with a resonator or quartz crystal.

Stand Alone Jumper XJ1

XJ1, labeled Stand Alone, should be inserted in stand-alone mode, or disconnected if a target is present. Always disconnect or remove the XJ1 jumper before the pod is connected to the target.

Note: *The EMUL251™-PC has special circuitry that protects the pod and target if the target's power or the pod's power or both are not present. This protection circuitry will only function if XJ1 is disconnected.*

Configuration Bytes

The POD-25ASB, Rev. C, uses an ICE-Ready part. In ICE mode, the configuration bytes CONFIG1 and CONFIG0 are set by selecting the appropriate options under the **Config Emulator Hardware** menu. This allows the user to vary WSB, WSA, RD1, RD0, PAGE, SRC, INTR and the EMAP bits without reprogramming the pod's EPROM memory.

Warning: The ICE-Ready 87C251 should never have its internal EPROM programmed.

Important: After making changes in the Config Window, when asked "Implement Reset Now?" you must click on the Yes button for the change to take effect.

POD-251SB, Rev. D

Introduction

This pod board contains an Intel N87C251SB bondout microcontroller chip (suitable for emulating the Intel 8XC251SB), a 16 MHz crystal, 256 kilobytes emulation RAM for instructions and data, circuits for driving the cable bus, two PLDs, three CPLDs and two FPGA chips.

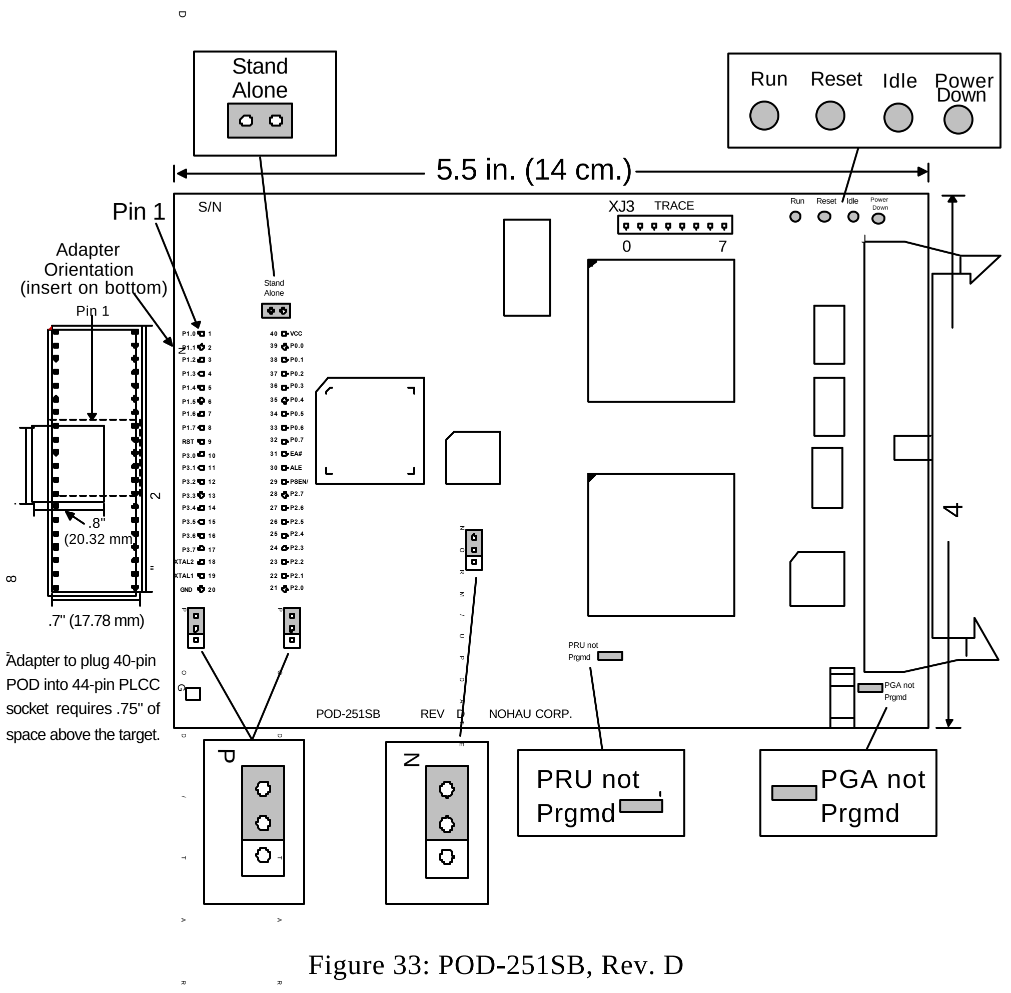


Figure 33: POD-251SB, Rev. D

Identifying the Pod

The pod has printing at the edge of the board. The revision is printed between the words "POD-251SB" and "NOHAU CORP". This section covers REV D. If your pod is REV C, turn to the section for that revision.

Dimensions

The pod board is 5.5 inches by 4 inches (14 cm. by 10.1 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.

Theory of Operation

The 87C251 microcontroller supplied on the POD-251SB, Rev. D is an ICE-Ready device. This allows a special mode in which the pod can execute from its EPROM area at full internal access speed without having its internal EPROM programmed. Users can download code to the pod's EPROM area and experiment with different selections in the configuration bytes CONFIG0 and CONFIG1 without using an EPROM programmer.

The POD-251SB, Rev. D, allows users to accurately set breakpoints at any instruction address and map any memory location to the target or to the POD-251SB, Rev. D's emulation memory. The pod does not use any stack area or other user resource, or impose any restrictions on the user's hardware or software design.

For proper operation with a target, the page mode select button in the **Config Emulator Hardware** menu should be selected (or not selected) depending on design of the 8XC251 target. It may be necessary to preconfigure this page mode option in stand-alone mode to "initialize" the EMUL251.ini file to match that of the target.

When changing from stand-alone to target operation, check the "stand-alone" jumper XJ1. Also, verify the source code or binary code compatible selection in the **Config Emulator Hardware** menu. This selection should match the compiler / assembler mode option (source mode or binary mode) for any file that is downloaded for execution.

Important: After making changes in the Config Window, when asked "Implement Reset Now?" you must click on the YES button for the change to take effect.

All tracing for the POD-251SB, Rev. D, is execution trace, which records only those instruction addresses that were actually executed. External data transferred, as well as P3, P2 and P0 port transfers are also recorded in the trace.

Interface

The POD-251SB, Rev. D, can interface directly to a 40-pin DIP target socket or through an adapter to a 44-pin PLCC target socket. Clip-over "ONCE mode" adapters for 40 and 44 pin targets may be introduced but were not available when this manual version was written.

Emulation Memory

The POD-251SB, Rev. D, has 256K bytes of emulation memory. Logical to physical translation is controlled by the RD1 and RD0 bits in the CONFIG0 register. The **CONFIG** register values do not interfere with the user's code. These configuration bits are accessed by the emulator hardware menu selection under the **Config** selection of the software interface. The emulation hardware automatically configures emulation memory spaces based on RD1 and RD0 as described in the External Memory Interface chapter of Intel's 8XC251SB Embedded Microcontroller User's Manual.

Address ranges can be mapped into target memory with a precision of 2 bytes for non-page mode, and 256 bytes for page mode. The pod's 16K internal EPROM is always mapped to emulation memory when not in external mode. Although the pod has special protection circuitry built in, it is still preferable to turn on the PC before applying power to the target and turn off the target power before turning off the PC power.

Warning: **Always remove the Stand Alone jumper on the pod before connecting the pod to a target system.**

POD-251SB Rev. D Headers

JP1, JP2, JP3, XJ1, XJ3

JP2 and JP3 determine on-pod or on-target crystal selection.

JP2 and JP3 each have two jumper positions: TARGET and POD. When set in the TARGET position, the pod controller will receive the clock signal from the target crystal. With both in the POD position, the bondout controller on the pod will use the crystal on the pod.

Note: *When the clock jumpers are in the POD position, the XTAL1 and XTAL2 signals from the pod are disconnected from the target.*

External Drive Clock

This section only applies if you are driving a clock signal into the XTAL1 input of the 8XC251SB. If you are using both pins with a crystal or resonator, this section does not apply.

Due to a layout error on this revision of the pod, the XTAL1 and XTAL2 pins are swapped. If you are driving the 8XC251SB with a clock signal, the signal must be crossed over to the other pin.

One way to do this is to cross the signal over on the pod.

1. Remove both jumpers JP2 and JP3 (both labeled "POD/TARGET").

2. Add a short wire from JP3 pin 1, labeled "TARGET" to JP2 pin 2, the middle pin.

3. This swap is only necessary with an external driving clock signal, not with a resonator or quartz crystal.

Stand Alone Jumper XJ1

XJ1, labeled Stand Alone, should be inserted in stand-alone mode, or disconnected if a target is present. Always disconnect or remove the XJ1 jumper before the pod is connected to the target.

Note: *The EMUL251™-PC has special circuitry that protects the pod and target if the target's power or the pod's power or both are not present. This protection circuitry will only function if XJ1 is disconnected.*

JP1: EEPROM-based Pod Logic

To prevent the ISP 22V10 components from being inadvertently programmed with the wrong information, the JP1 jumper should always be in the NORM, not the UPDATE position when the EMUL251 software is run from *Windows*.

Two of the logic components on this revision of the pod are EEPROM-based. They are Lattice ISP 22V10s. The parts are programmed at the factory with the correct information. Normally the user does not need to update these parts. These parts must be updated if ICE Technology generates an update or if they become accidentally reprogrammed incorrectly.

Reprogramming or Updating the ISP 22V10s

If it is necessary to reprogram or to update the ISP 22V10s, follow this procedure:

1. Exit EMUL251.
2. Move JP1 from the NORM position to the UPDATE position.

3. Run the ISP programming software. At the time of this writing, the software is a DOS program named ISPDOS.EXE, but a *Windows* program is planned. Files such as SRIPGM7.BIN and ADRSA7DQ.BIN are typical data files and should be in the same directory as the programming EXE file.

4. Move JP1 from UPDATE back to NORM.

XJ3: Trace

This header is to allow the optional trace board to record logic signals the user connects to these pins. At the time of this printing, the feature was not implemented.

Configuration Bytes

The POD-251SB, Rev. D, uses an ICE-Ready part. In ICE mode, the configuration bytes CONFIG1 and CONFIG0 are set by selecting the appropriate options under the **Config Emulator Hardware** menu. This allows the user to vary WSB, WSA, RD1, RD0, PAGE, SRC, INTR and the EMAP bits without reprogramming the pod's EPROM memory.

Warning: The ICE-Ready 87C251 should never have its internal EPROM programmed.

Important: After making changes in the Config Window, when asked "Implement Reset Now?" you must click on the Yes button for the change to take effect.

POD-251SB, Rev. E

Introduction

This pod board contains an Intel N87C251SB bondout microcontroller chip (suitable for emulating the Intel 8XC251SB), a 16 MHz crystal, 256 kilobytes emulation RAM for instructions and data, circuits for driving the cable bus, two PLDs, three CPLDs and two FPGA chips.

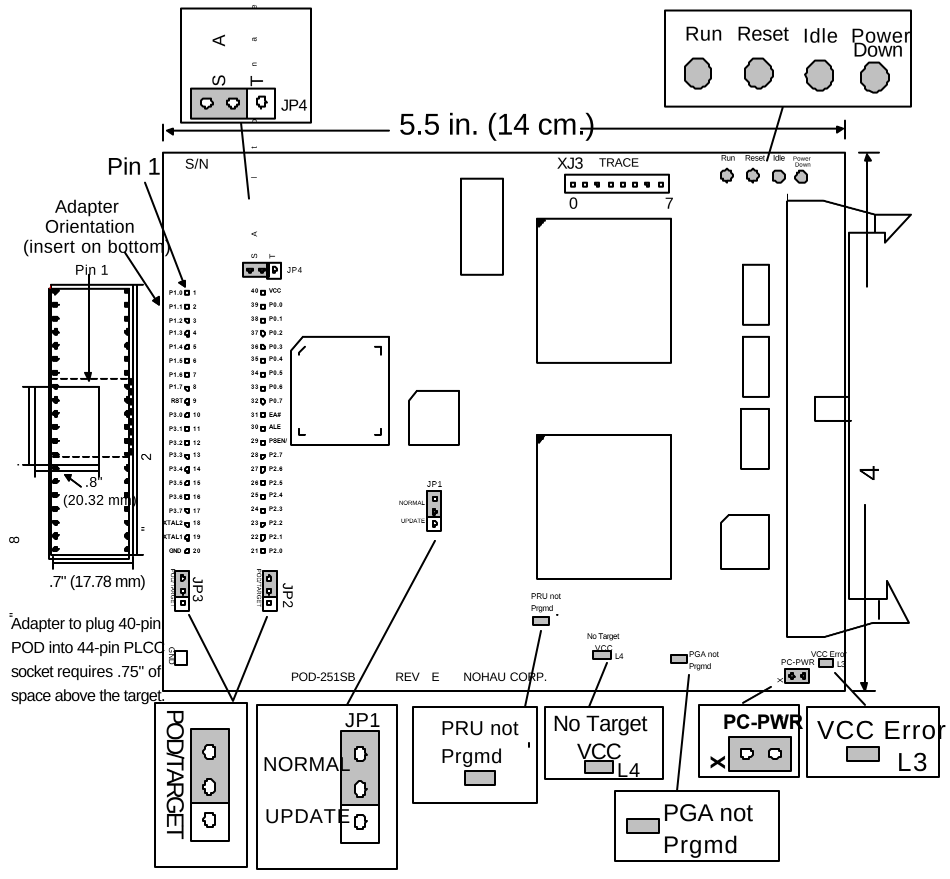


Figure 34: POD-251SB, Rev. E

Identifying the Pod

The pod has printing at the edge of the board. The revision is printed between the words "POD-251SB" and "NOHAU CORP". This section covers REV E. The label on the connector reading “E.Bx” indicates it has a “B-step” chip.

If your pod is REV C it is no longer supported; please contact customer support at support@icetech.com If your pod is Rev. D, turn to the respective section in the manual for that revision.

Note: *If your Rev. D pod has no label on the connector, the chip is an “A-step”. If your Rev. D pod has a label on the connector reading “D.Bx”, the “B” indicates it has a “B-step” chip.*

Dimensions

The pod board is 5.5 inches by 4 inches (14 cm. by 10.1 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.

Theory of Operation

The 87C251 microcontroller supplied on the POD-251SB, Rev. E, is an ICE-Ready device. This allows a special mode in which the pod can execute from its EPROM area at full internal access speed without having its internal EPROM programmed. There is 16K of downloadable EPROM area, separate from the 256K emulation memory. Users can download code to the pod’s memory and experiment with different selections in the configuration bytes UCONFIG0 and UCONFIG1 without using an EPROM programmer.

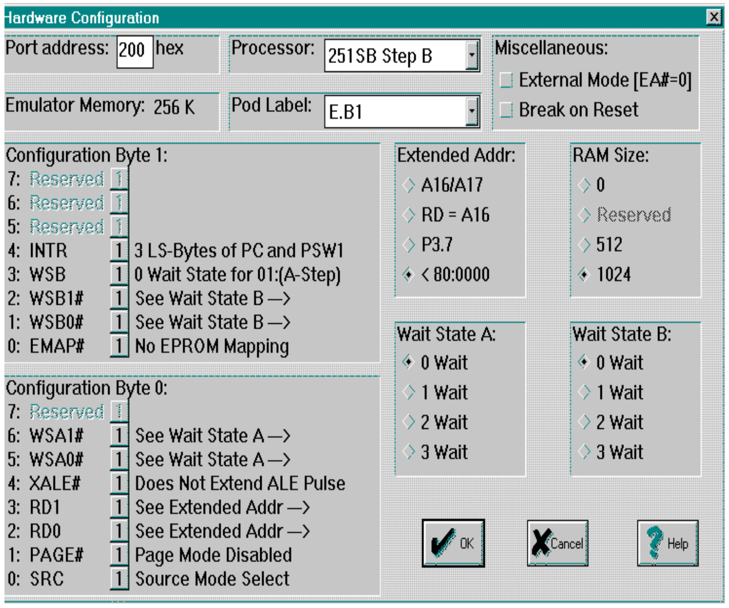
The POD-251SB, Rev. E, allows users to accurately set breakpoints at any instruction address and map any memory location to the target or to the POD-251SB, Rev. E's emulation memory. The pod does not use any stack area or other user resource, or impose any restrictions on the user's hardware or software design.

For proper operation with a target, the page mode select button in the **Config Emulator Hardware** menu should be selected (or not selected) depending on design of the 8XC251 target. It may be necessary to preconfigure this page mode option in stand-alone mode to "initialize" the EMUL251.INI file to match that of the target.

When changing from stand-alone to target operation, check the "stand-alone" jumper JP4.

Note: *If you are using the PC Card emulator board, the XJ5 jumper should be removed, and power should come in through J1. Always remove the XJ5 jumper pin when using an external DC power supply through J1. For more information See "XJ5: PC-Power" on page113.*

Also, verify the source code or binary code compatible selection in the **Config Emulator Hardware** menu. This selection should match the compiler / assembler mode option (source mode or binary mode) for any file that is downloaded for execution.



Hardware Configuration

Port address: 200 hex
Processor: 251SB Step B
Miscellaneous:
☐ External Mode [EA#=0]
☐ Break on Reset
Emulator Memory: 256 K
Pod Label: E.B1

Configuration Byte 1:

7:	Reserved	1	
6:	Reserved	1	
5:	Reserved	1	
4:	INTR	1	3 LS-Bytes of PC and PSW1
3:	WSB	1	0 Wait State for 01:(A-Step)
2:	WSB1#	1	See Wait State B —>
1:	WSB0#	1	See Wait State B —>
0:	EMAP#	1	No EPROM Mapping

Configuration Byte 0:

7:	Reserved	1	
6:	WSA1#	1	See Wait State A —>
5:	WSA0#	1	See Wait State A —>
4:	XALE#	1	Does Not Extend ALE Pulse
3:	RD1	1	See Extended Addr —>
2:	RD0	1	See Extended Addr —>
1:	PAGE#	1	Page Mode Disabled
0:	SRC	1	Source Mode Select

Extended Addr: A16/A17; RD = A16; P3.7; [✓ < 80:0000]
RAM Size: 0; Reserved; 512; [✓ 1024]
Wait State A: [✓ 0 Wait]; 1 Wait; 2 Wait; 3 Wait
Wait State B: [✓ 0 Wait]; 1 Wait; 2 Wait; 3 Wait

Figure 35: Hardware Configuration Window

Note: *Changing Configuration Byte values with a --> in the description will automatically affect the corresponding Wait State or Extended Address selection.*

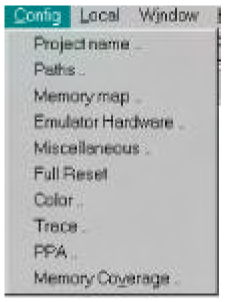
Important: After making changes in the Config Window, when asked "Implement Reset Now?" you must click on the YES button for the change to take effect.

All tracing for the POD-251SB, Rev. E, is execution trace, which records only those instruction addresses that were actually executed. External data transfers are also recorded in the trace.

Interface

The POD-251SB, Rev. E ,can interface directly to a 40-pin DIP target socket or through an adapter to a 44-pin PLCC target socket. Clip-over “ONCE mode” adapters for 40 and 44 pin targets may be introduced but were not available when this manual version was written.

Emulation Memory



The POD-251SB, Rev. E, has 256K bytes of emulation memory. Logical to physical translation is controlled by the RD1 and RD0 bits in the UCONFIG0 register. The **UCONFIG** register values do not interfere with the user’s code. These configuration bits are accessed by the emulator hardware menu selection under the **Config** selection of the software interface. The emulation hardware automatically configures emulation memory spaces based on RD1 and RD0 as described in the External Memory Interface chapter of Intel’s 8XC251SA, 8XC251SB, 8XC251SP, 8XC251SQ Embedded Microcontroller User’s Manual.

Address ranges can be mapped into target memory with a precision of 2 bytes for non-page mode, and 256 bytes for page mode. The pod’s internal EPROM is always mapped to emulation memory when not in external mode. Although the pod has special protection circuitry built in, it is still preferable to turn on the PC before applying power to the target and turn off the target power before turning off the PC power.

POD-251SB, Rev. E Headers

JP1, JP2, JP3, XJ1, XJ3

JP2 and JP3 determine on-pod or on-target crystal selection.

JP2 and JP3 each have two jumper positions: TARGET and POD. When set in the TARGET position, the pod controller will receive the clock signal from the target crystal. With both in the POD position, the bondout controller on the pod will use the crystal on the pod.

Note: *When the clock jumpers are in the POD position, the XTAL1 and XTAL2 signals from the pod are disconnected from the target.*

External Crystal or Drive Clock

If the target uses a crystal frequency other than 16 MHz, or if the target uses the XTAL2 output for a CMOS level XTAL output, the JP2 and JP3 jumpers should be selected to the "Target" position. (No "swapping", required in earlier revisions, is necessary.)

Stand Alone/Target Jumper JP4

The POD-251SB, Rev. E, has a two-position power jumper, JP4, for selection of the pod's 80251 VCC. A jumper pin on JP4 in the "Stand Alone" position connects the pod's VCC to the 80251's VCC. A jumper pin on JP4 in the "Target" position connects the target's VCC to the 80251's VCC.

Note: *It is permissible to have JP4 in the Stand Alone position when connected to the target. This will power the 80251 from the pod's VCC, and give additional protection to the 80251 when the target is off and the pod is on.*

With the POD-251SB, Rev. E, it is not possible to power the target from the pod's VCC with a simple jumper. It is not recommended to supply the target with power from the pod's VCC.

XJ5: PC-Power

If you are using the low cost emulator board, the XJ5 jumper should be removed and power should come in through J1. The recommended supply source for J1 is a SKYNET Electronic Co. Model: SNP-Q316 transformer. Always remove the XJ5 jumper pin when using an external DC power supply through J1.

J1 is a regulated DC input for the pod's 5v. Do not supply power through J1 if the XJ5 jumper is in place and power comes from the PC.

Note: *The Skynet Model SNP-Q316 Universal Input (115v or 230v), 5v @ 2A output, is orderable through Nohau and is the approved external power supply. The DC power supply should also be used if the voltage or the pod falls below 4.7v DC (with increased contact resistance on older cables and connectors).*

JP1: EEPROM-based Pod Logic

To prevent the ISP 22V10 components from being inadvertently programmed with the wrong information, the JP1 jumper should always be in the NORM, not the UPDATE position when the EMUL251 software is run from *Windows.*

Two of the logic components on this revision of the pod are EEPROM-based. They are Lattice ISP 22V10s. The parts are programmed at the factory with the correct information. Normally the user does not need to update these parts. These parts must be programmed if ICE Technology generates an update or if they become accidentally reprogrammed incorrectly. The README.WRI file will contain information about required updates.

Reprogramming or Updating the ISP 22V10s

If it is necessary to reprogram or to update the ISP 22V10s, follow this procedure.

1. Exit EMUL251.
2. Move JP1 from the NORM position to the UPDATE position.
3. Change directories to the emulator installed directory, EMUL251 by default.
4. Run the ISP programming software. This program may is a *Windows* program named ISPWIN. Files such as SRIPGM7.BIN and ADRSA7DQ.BIN are typical data files and should be in the same directory as the programming EXE file.
5. Move JP1 from UPDATE back to NORM.

XJ3: Trace

This header is to allow the optional trace board to record logic signals the user connects to these pins. At the time of this printing, the feature was not implemented.

Default position for jumper pins:

JP1 - Normal	Prevents inadvertent programming of 22V10's
JP2 - POD	Selects pod's crystal oscillator
JP3 - POD	Selects pod's crystal oscillator
JP4 - Stand Alone/Target	Pod VCC to 80251 bondout's VCC
XJ5 - Inserted	Supplies pod's power from PC

Added VCC Status LEDs

Two VCC status LEDs have been added to the POD-251SB, Rev. E, since earlier revisions:

L3 - VCC Error LED

This LED will be on if:

1) there is no PC VCC and there is pod VCC; or

2) there is no pod VCC and there is PC VCC.

Note: There is special protection circuitry on the POD-251SB, Rev. E to protect both the emulator and the pod under the conditions that light L3.

L4 - No Target VCC

This LED will be on if:

1) the pal is running stand alone with no target.

2) there is no target VCC and there is pod VCC;

3) there is target VCC and no pod VCC;

Note: It is not recommended to leave the target VCC on while the pod's VCC is off for extended periods. It is also not recommended to leave the pod VCC on while the target's VCC is off for extended periods.

Configuration Bytes

The POD-251SB, Rev. E, uses an ICE-Ready part. In ICE mode, the configuration bytes UCONFIG1 and UCONFIG0 are set by selecting the appropriate options under the **Config Emulator Hardware** menu. This allows the user to vary WSB, WSB1#, WSB0#, WSA1#, WSA0#, XALE#, RD1, RD0, PAGE#, SRC, INTR and the EMAP# bits without reprogramming the pod's EPROM memory.

Warning: The ICE-Ready 87C251 should never have its internal EPROM programmed.

Important: After making changes in the Config Window, when asked "Implement Reset Now?" you must click on the Yes button for the change to take effect.

Chapter 6: Troubleshooting

Overview

If you have trouble with your emulator, you may, of course, email or call customer support. 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.

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.

Step 2: PWR and XTAL jumpers

Remove the pod from the target, and install the Stand Alone jumper from the pod.

Move the XTAL jumpers to the pod position.

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

Step 3: Sample User Program

If you call technical support team, you may 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 FF0000
4. Hit <Enter>
5. Type:
   `NOP` <Enter>
   `NOP` <Enter>
   `NOP` <Enter>
   `LJMP 0000` <Enter>
6. Click on the **GO** button in the tool bar.
7. Click on **BREAK**.

Index