EMUL51XA™-PC

NOHAU EMUL51XA

User Guide

www.icetech.com

Tel: 650.375.0409

Fax: 650.375.8666

http://www.icetech.com

Email: sales@icetech.com
Email: support@icetech.com

Fouth Edition

Warranty Information

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.

Emulator hardware is warranted for 90 days of purchase excepting any electronic or mechanical damage as a result of exceeding electronic or mechanical tolerances.

- Warning -

Prevent damage to the emulator or your project device.

Never turn the emulator off while connected to a powered unit under test.
Never plug the emulator into a unit under test that is energized.

Table of Contents

Introducing EMUL51XA-PC

The EMUL51XA-PC is a personal computer-based emulator for the Philips' 80C51XA family of microcontrollers. The EMUL51XA-PC consists of an emulator board that "plugs in" directly to the PC, or is an external high-speed parallel box; a five-foot-long (1.5 m) twisted-pair ribbon cable; various pod boards and an optional tracing board. The EMUL51XA-PC design supports all Philips microcontrollers that are based on the 80C51XA core.

The EMUL51XA-PC software is a *Microsoft Windows 3.x/'95/NT* application. It follows the *MS Windows* Multiple Document Interface Standard. That means it has the same look and feel as applications produced by *Microsoft* for *MS Windows*.

The EMUL51XA-PC user interface is consistent with most other *MS Windows* applications and includes dynamically changing menus, moveable and scrollable "child" windows, function-key shortcuts for menu items, and context-sensitive help. Anyone familiar with *MS Windows* applications will be able to use EMUL51XA-PC with little or no other assistance. EMUL51XA-PC also supports the *MS Windows* Dynamic Data Exchange protocol and can export data written to off-chip data RAM to other *MS Windows* applications.

The EMUL51XA-PC hardware is modular. The software user interface implements an effective high-level debugger. It has support for local variables, C typedefs and C structures. The trace board options add bus-cycle tracing, triggering and filtering, shadow RAM, data display, and timing functions.

How to use this manual

This manual was written with different kinds of users in mind. All users should have *MS Windows* installed and have learned the skills taught in the Basic Skills chapter of the *Microsoft Windows* User's Guide. Many of the EMUL51XA-PC features are designed around the features of the supported chips. Being familiar with the chip is a prerequisite to understanding how to use the emulator productively.

If you are new to emulators of any kind,

read the manual completely. You may skip the sections that describe pods and accessories you do not have.

IIf you have used emulators with other microprocessors,

but are not familiar with the chip being emulated, you are strongly encouraged to review the features of the chip you have, then thoroughly read the section that

describes the applicable pod board before running the emulator. The tutorial in Chapter 10 will familiarize you with Seehau debugger and trace operations.

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

read the section that describes the pod you are using, then begin using EMUL51XA-PC, referring to on-line help as needed. After a few days of use, skimming the reference chapters may highlight useful features. Chapter 10 will highlight features unique to the EMUL51XA emulator.

Manual Conventions

Type the words in double quotes exactly as shown, but without the quotation marks, except for the <Enter>, <Ctrl>, <Tab>, <Ins>, <Del> 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 in **boldface** to set them apart from the rest of the text.

Any screen may be saved to *Windows'* **Clipboard Viewer**, or pasted to **Paint** after using <Alt> Print Screen.

Notes and hints are printed in italics, and

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

Quick Installation Instructions

System Requirements

The EMUL51XA-PC requires a personal computer with at least 2 megabytes of RAM (8 megabytes for *Windows `95*); a CPU that is either 80386, 80486, or Pentium-compatible; a hard disk with at least 3 megabytes of unused space; and *Microsoft Windows 3.1, 3.11 or Windows `95, Windows NT* or *OS/2 2.1* (or higher) installed. A mouse is not required, but is strongly recommended.

One ½ height ISA (or EISA) bus slot is required if you are not using the external "high-speed parallel" printer port connection.

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.

Installing the Emulator

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

1. Turn off the power.
2. Remove the PC cover.
3. Remove the slot cover (if present) for an available 8-bit slot.
4. Insert the emulator board into the slot and use a screw to secure the emulator.
5. You may now close the cover, and attach the ribbon cable to the emulator and the pod.

Note: This installation is not applicable with high-speed parallel external boxes.

Installing the Piggyback Trace Board (If Used)

1. Turn off the power.
2. Remove PGA pin protector from the pod exposing gold-plated bondout PGA pins.
3. Mate the PGA male pins and the two side connectors to the Internal/External Trace Board (IETR); press together.
4. Install 5 VDC skynet power supply on the same power strip as the PC so that the pod and trace powers are cycled in unison.

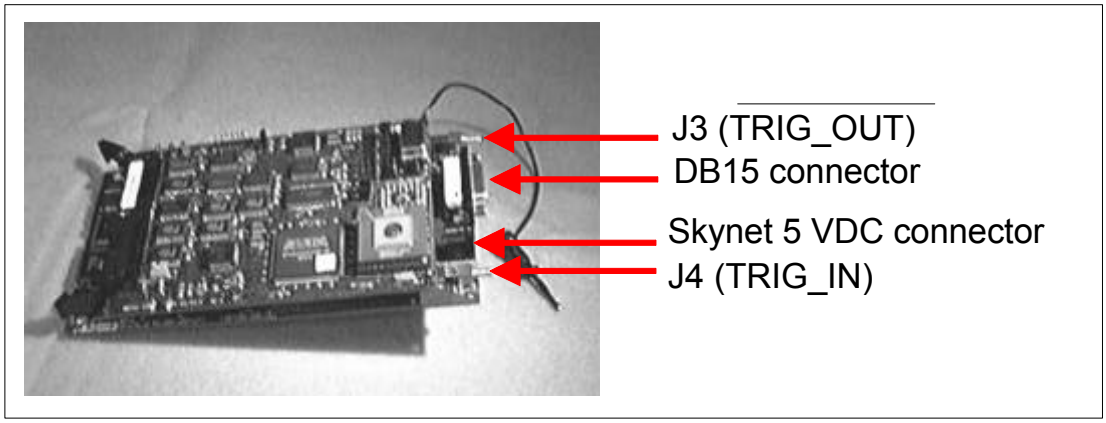


Figure 1: Assembling pod and trace boards

Installing the Pod

With the PC power off, line up the cable connector 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, insert the cable connector into the slot firmly, and close the side clamps. Remove any anti-static foam from the pins on the bottom of the pod when running the standalone confidence test.

Installing the EMUL51XA-PC 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 EMUL51XA-PC 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 Nohau program group containing the EMUL51XA-PC icon. Double-clicking on this icon will start the EMUL51XA-PC application.

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

Initial Software Configuration

The Windows software is used for all EMUL51XA-PC products. The type of target processor in the software configuration must agree with the type of pod you are using. If not, you may see an error message. To ensure that you do not get this error, we include a utility that you must run when you first install the emulator, or when you change your pod type. This utility is called **INI_XA**. (You can also run this utility any time you want to check the values in the initialization file.)

To invoke **INI_XA**, double-click on the icon the EMUL51XA-PC program group labeled **INI_XA**. If the EMUL51XA.ini settings are not self-consistent, you will see a warning message, otherwise you will see the window shown in Figure 2.

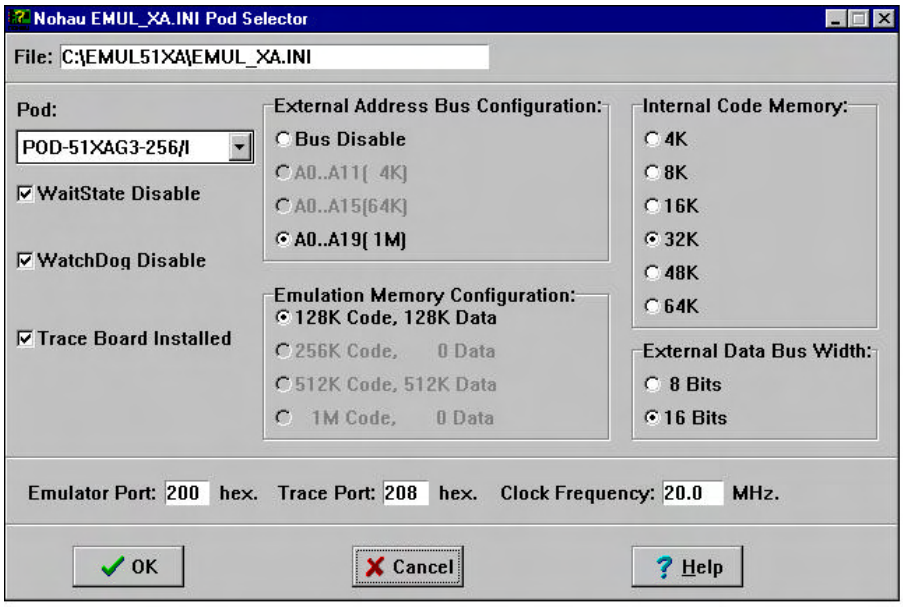


Figure 2: Choosing a target processor with INI_XA

The **Emulator Port:** field must agree with the values you set in the J1 jumper on the emulator board, as described in the section “Setting the I/O address jumpers - J1” on page 58. The **Trace Port** address can be ignored.

If using the IETR trace and 3.3 volts direct current (VDC) target, make the changes in the **Trace Setup** screen for 3.3 VDC (See Figure 48 on page 80.)

Bus width: cannot be changed outside this setup window. The bondout supports internal code memory of future derivatives. It should reflect the actual internal code ROM being used.

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

Confidence Test

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

The Confidence Test will read the pod name and I/O address from the EMUL_XA.INI file If these values are incorrect, you must run INI_XA.EXE. It will take less than a minute to complete the tests. Many of the tests repeat with slight variations. If any tests report unexpected errors, email Customer Support for assistance at support@icetech.com.

Chapter 1: Software User Interface

Detailed Installation Instructions

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

The EMUL51XA-PC floppy disk includes an *MS Windows*-compatible SETUP.EXE program. To install this software, run SETUP.EXE by typing "WIN A:SETUP" at the DOS prompt before entering *Windows* or, from within *MS Windows*, by selecting the **Run** item in the **Program Manager File** menu and 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 select **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 EMUL51XA-PC software. Either accept the suggested directory 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.

When installed, there will be an **EMUL51XA** Program group with an **XA Emulator** icon. Double-clicking on the icon will start the 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.

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 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 assume 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 one. The new project will be saved when exiting the debugger or when changing project settings (again).

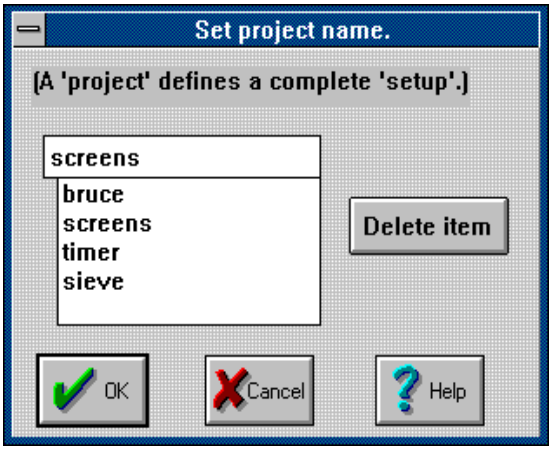


Figure 3: Set Project Name Dialog Box

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. To delete a project, highlight the project name, then click on the **Delete item** button.

Figure 4: EMUL51XA-PC Title Bar

Setting the Paths..

The name of the current project appears in the emulator software title bar. Figure 4 is an example of the EMUL51XA title bar for the project named **SCREENS** (used to create the screen shots for this manual).

The next item in the **Config** menu is **Paths ..** , which opens the dialog box shown in Figure 5. 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 semicolon (;) 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 shown in the **Load code..** dialog box.

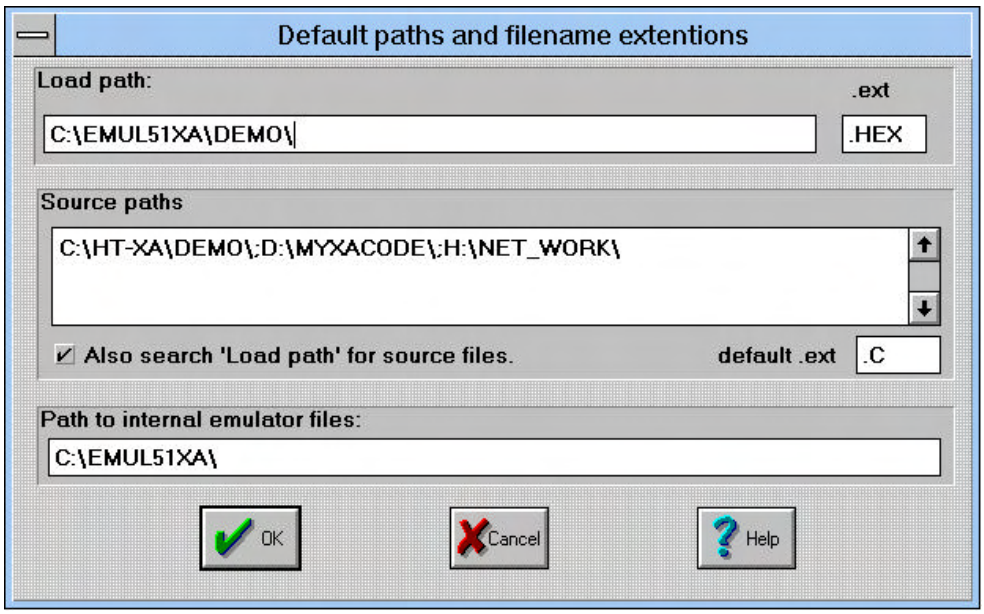


Figure 5: Paths Dialog Box

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 EMUL51XA-PC is loading, the debugger will first look at that source file when updating the **Source** window.

The second field, **Source paths: User source code:**, 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. When the small check box is checked, it will tell EMUL51XA-PC to look for source files in the **Load path:** directory as well. Simple projects may have all the source and object files in the same directory (the **Load path:**) and may not need any directories in the **Source paths:** field.

Note: The ".ext" field specifies the source file extension. If your C modules have the extension ".c", enter that. To see assembler source (.asm) in the Source window, enter ".asm" (this works only if line number information is included in the sym file).

The **Emulator internal files:** field will be set during the installation and may 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 EMUL51XA-PC software, such as register-definition files and dynamically loaded libraries. Normally, the installation Program will set this to the proper directory. If you copy

or move EMUL51XA-PC software to a new directory or disk drive, remember to change this field.

Mapping memory

The POD-51XA/G3/I includes either 256K or 1 M, divided respectively into 128K code / 128K data or 512K code / 512K data. This RAM is called emulation RAM, or emulation memory. The entire address range for both ROM and ROM can be mapped to either the target or the emulator memory in blocks as small as 16 bytes.

The POD-51XA/G3/E can also map all memory, 256K or 1M, to code space and none to data space. 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.

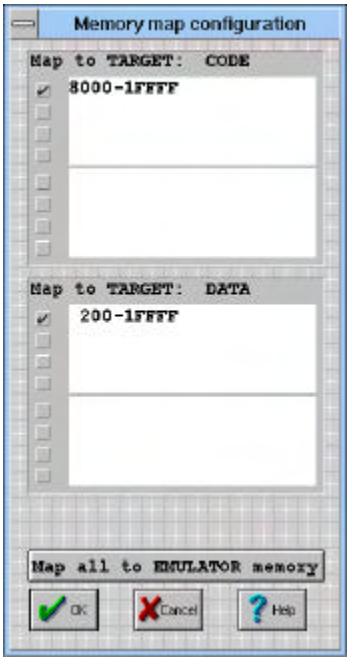


Figure 6: Mapping Memory Addresses to the Target

There are 8 address ranges in the CODE and DATA dialog boxes respectively. Each address range can be as large as 128K (512K if 1M POD), or as small as 16 bytes.

The POD-51XA/G3/E can be configured as 256K (1M) code memory, in which case you must map any external data memory to target.

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, code and data to the emulator.

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.

*Note: If you need to change your setup, un-check the box, select **OK**, then recheck the box and enter the corrected address range.*

Emulator Hardware Configuration

Run the Program XA INI generator, then ensure that pod jumpers match these selections, i.e., 8-bit buswidth and bus address width. See chapters 6 and 7 for more information.

Warning:	***The settings in this dialog box must agree with the emulator jumper settings, the pod processor type, and the application startup code. If this is not the case, EMUL51XA-PC will not work properly.***

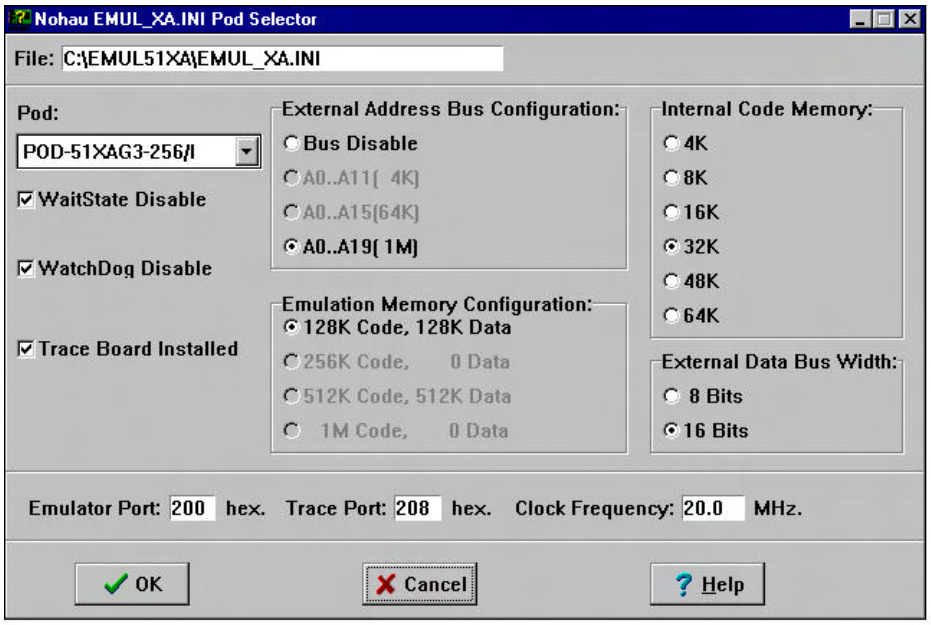


Figure 7: POD-51XA/G3 Hardware Configuration Screen

The emulator always boots into the **Hardware Configuration** screen. **Port address** refers to the physical emulator ISA bus address (see Chapter 3: Emulator Board for details), which is set to 200, or the HSP box printer port address, e.g., 378 for LPT1.

External Address Bus Configuration, Buswidth, Watch Dog disable and **Wait State disable** are self-explanatory, and should agree with your user-code programming of the BCR register and hardware state of the BUSW pin.

For the POD-51XA/G3/I, the **Internal Code Memory** by default is 32K for the G3; however the G37 bondout supports up to 64K of internal memory. The POD-51XA/G3/I must run with the /EA pin pulled high at **RESET**, and will run with internal bus timing up to the size specified by this window, or three clocks per cycle. If using this for ROM-less operations, you might select 4K, after which the code will execute externally with timing per BTRL bits CR1, CR0, CRA1 and CRA0. ROM-less designs should use the POD-51XA/G3/E with /EA pulled low at **RESET**. Select emulation memory options to match the POD-51XA/G3/E jumper settings of JP19 and JP21, JP22 (see Chapter 7: POD-51XA/G3/E).

The **Clock Frequency** must be set to ensure correct POD logic timing (see Chapter 6: POD-51XA/G3/I and Chapter 7: POD-51XA/G3/E for limitations).

Miscellaneous Configuration

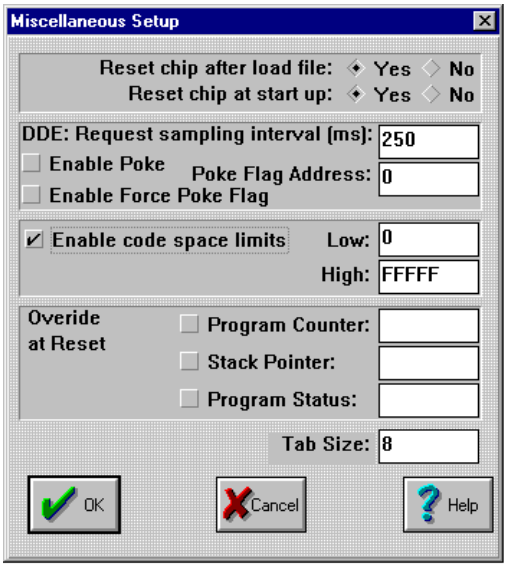


Figure 8: Miscellaneous Setup Dialog Box

The **Miscellaneous** item in the **Config** menu opens a dialog box that controls special features of EMUL51XA-PC:

when and if automatic resets occur
DDE parameters
the source code address range for limiting breakpoint locations
optional reset vector values Program counter, stack pointer and status values

By default, the emulator resets the controller when the EMUL51XA-PC software is started and after an object file is loaded. The **Reset chip at start up:** and the **Reset chip after load file:** radio buttons can disable either of those resets, which may be helpful during particularly difficult or unusual debugging circumstances.

Dynamic Data Exchange

The Dynamic Data Exchange (DDE) feature allows one *MS Windows* application to send data to another. EMUL51XA-PC uses DDE protocols and can export 16-bit values from Shadow RAM to other applications, such as Word for Windows or Excel.

Note: The Internal/External Data Trace Board (IETR) contains the Shadow RAM. DDE will not work without this piggyback trace card. It is not necessary, however, to open a Shadow RAM window to establish a link to the other application.

Most applications use the copy/paste mechanism for establishing the link. EMUL51XA-PC does not include a **Copy** menu item, so establishing the link is slightly different for each client application.

To establish a link from EMUL51XA-PC to Excel, select the destination cell, then enter the following formula:

=emul_xa/shadow! '<ShadowRam address in hex>'

Hint: <Space> characters in the formula will confuse Excel; you may also need to adjust the address lower or higher when dealing with byte writes.

Address options are <address in hex, b>; meaning "look for a byte transfer" and <address in hex, 1>; meaning "look for a 32-bit-long transfer."

Excel, by default, looks for 16-bit values.

Using the time.abs example, we would use

=emul_xa/shadow! '0A05D, b'

to see the timer.sec update on the Excel spreadsheet. Once in Excel, you can easily perform mathematical and graphing operations on the data.

Once EMUL51XA-PC has established the link, the Shadow RAM value will appear in that cell, and the cell will be updated as often as indicated in the **Config/Misc ..** dialog box field labeled DDE sampling interval. This field may be as small as 100 ms, or as large as 32 seconds.

The final feature, **DDE Poke**, is not implemented at this time. This lets one write to memory from the Shadow RAM window as long as the hex value 1234H is present at the **Poke Flag Address** location. After changing external data memory, the **Poke Flag Address** location is cleared to zero. The idea is to synchronize writes with the user Program.

Check the **Enable code space limits** box and set the **Low** and **High** addresses to encompass just these instructions. Configured this way, EMUL51XA-PC will not allow 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 EMUL51XA-PC resets the controller. If you have some test code without a startup module, filling in this field will force the Program counter to the specified value each time you reset the controller. Similarly, you can enable interrupts with a PSW of 8000 here rather than placing the value at address zero in code memory space.

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. The controller is reset by pulling the reset line low. When the emulator software is first started, or possibly after an accident on the pod or target, the states of the two large logic chips on the pod are not known. In a **Full Reset** before the controller is reset, the large logic chips on the pod are reloaded with their configuration information (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, however this is not needed.

Window Colors

Under the **Config** menu is the **Color..** menu item. Open this dialog box to set the colors of EMUL51XA-PC child windows. For example, all **Program** windows can be set to have a dark blue background with white text to differentiate them from other kinds of windows. At the same time, all **Data** windows can be green with black text, and all **Source** windows set to have white background and red text. It is possible to make each window very distinctive.

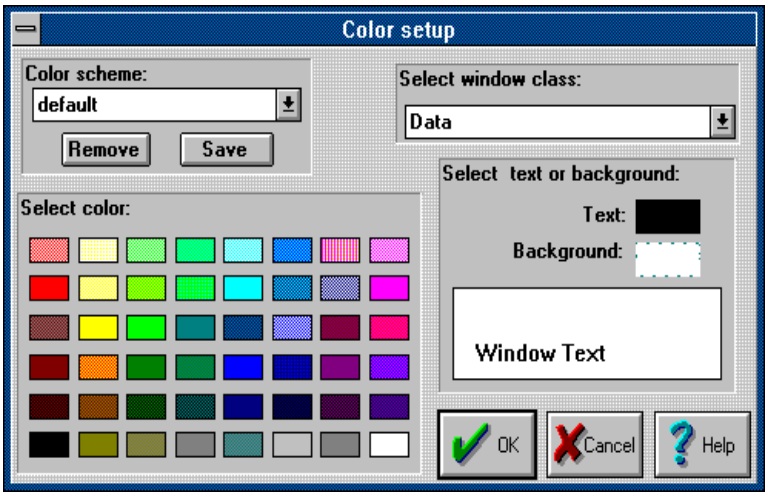


Figure 9: The Color Setup Dialog Box

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. EMUL51XA-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 EMUL51XA-PC. For information about the **Trace** menu item, please refer to Chapter 4.*

Performance Analysis (PPA)

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

Performance Analysis is a statistical analysis of execution behavior. Once every second, a percentage of the bus cycles are collected, sorted into their respective bins, and the results are displayed on the screen.

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

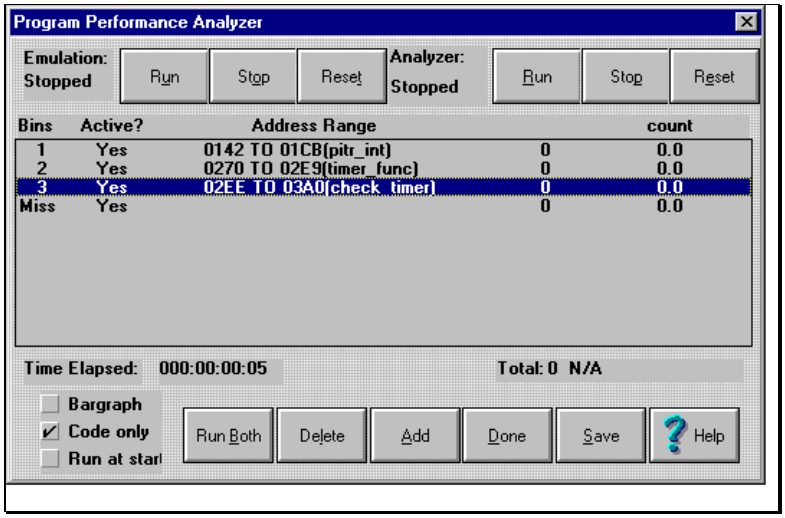


Figure 10: Performance Analysis Control Options

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

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

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

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

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

Figure 10 shows that data read and write cycles were ignored; the **Code only** option is checked. Only instruction fetches are counted in the statistics. **Run at start** causes the PPA to begin automatically.

If you prefer a graphic display, you may turn on the **Bargraph** option and see the data displayed in a form similar to that shown in the same figure.

To add bins of your own, click on the **Add** button to open the list of functions shown in Figure 10. You may add any of the functions as an address range, or you may create an address range not on the list.

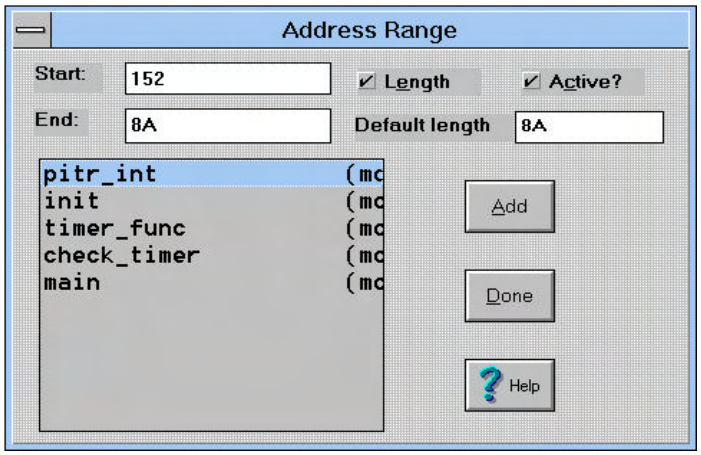


Figure 11: Adding a Bin

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

The **Length** field controls how the **End** field is used. With a check, the **End** field displays the length (as does default length after **Add**). Without a check mark in the **Length** field, the **End** field displays the end address in hexadecimal notation.

Using a very similar screen, any bin can be edited by double-clicking on that line in the PPA Control window of Figure 11. This is how you activate and deactivate bins.

Once you have collected the data you want, EMUL51XA-PC allows you to either save or discard the changes you just made to the list of bins. Only one bin configuration can be saved, not one per project like most configuration settings.

This bin configuration will be automatically restored the next time you use performance analysis.

Memory Coverage

The EMUL51XA-PC trace option includes the hardware necessary to monitor memory and correlate its use with your C-source code. If instructions are executed, the trace board will mark those addresses.

Note: The IETR distinguishes between prefetched and executed instructions. ONLY executed instructions are marked as covered. In the following discussion "coverage" means "executed."

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

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

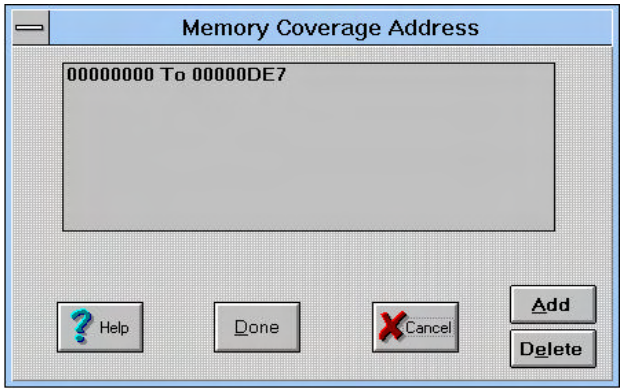


Figure 12: Editing the Coverage Address Range

You may either edit the existing range or you may add one or more address ranges. To edit the existing range, double-click on the line containing the address range. Double-click on the empty line just below the existing range; this will add another range, or use **Add** to include other addresses.

EMUL51XA-PC supports any number of address ranges. The ranges may be located anywhere in the address. The only practical limit is that the sum of all ranges must be less than 256 kilobytes with the IETR 128 trace and less than 1024 KB with the IETR 512 trace. You will need to use the trace setup screen **Trigger Memory Mapping..** to cover the correct 256 KB or 1024 KB address range.

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

Each square represents a memory word: two bytes starting on an even byte. Squares are grouped into segments 8 squares across. There are four rows of segments for each address. Figure 13 shows a **Memory Coverage** window with seven segments in each row. In this case, each row of dots represents 70 hex bytes of memory. Each row of blocks represents 1C0 hex bytes. As the window gets wider, each row contains more blocks of squares, and the addresses will get farther apart.

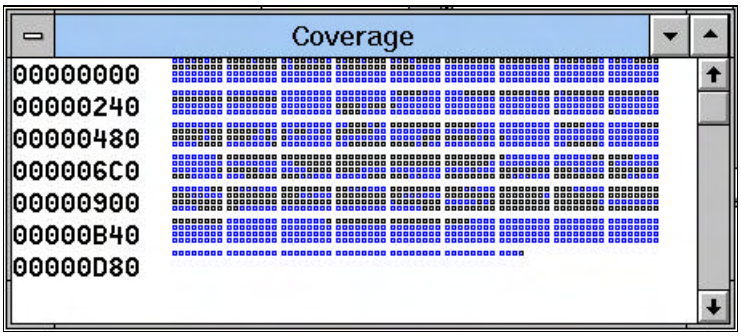


Figure 13: Typical Memory Coverage Window

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

As shown in Figure 14, there are new symbols in the **Source** and **Program** windows. In the **Program** window, you will find either a **:** or an **x** between the address and the rest of hexadecimal value at that address. The **:** means that the opcode has not been executed; **x** indicates that it has been.

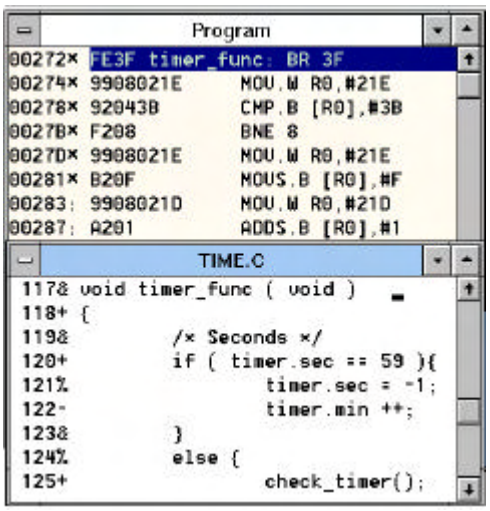


Figure 14: Program and Source windows in Coverage Mode

In the **Source** window, there are four new indicators between the line number and the line text: **+ = & %:**

+	All opcodes from this line were executed.
%	Some opcodes from this line were executed.
-	No opcodes from this line were executed.
&	This line generated no executable code.

Summary Memory Coverage Report

A screen full of coverage information is helpful, however there are additional reports that can satisfy government safety and other regulations.

When you select **Report** from the **Coverage** menu, you will see a dialog box similar to Figure 15.

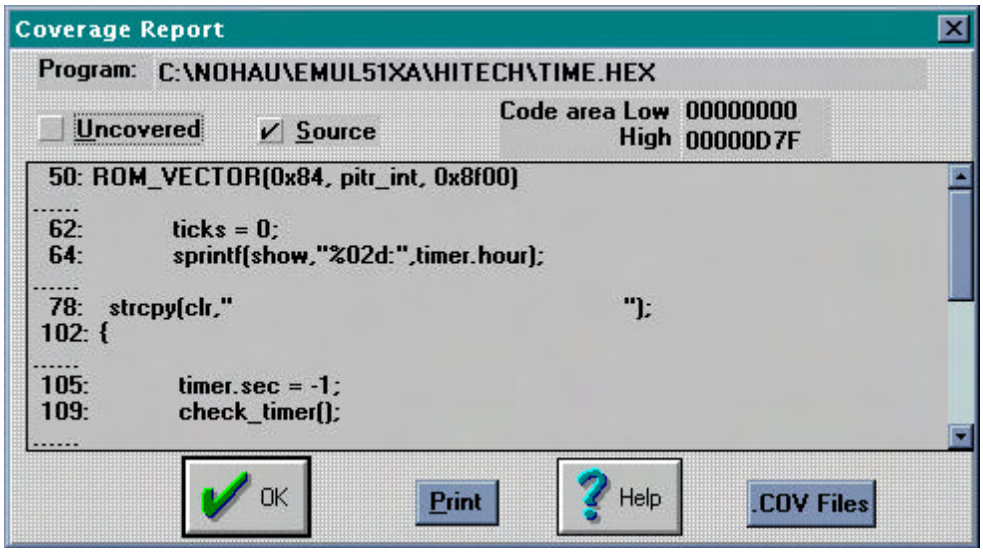


Figure 15: Summary Coverage Report

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

To obtain a written copy of the report, click on the **Print** button. This will send the summary to the current default printer. A different kind of report, one that's more detailed, can be made by clicking on the **.COV files** button. Figure 16 is the dialog box that configures these reports.

The summary report is good for small test runs that can be completed without turning off your PC or exiting the EMUL51XA-PC software. More typically software testing will extend over a period of days The detailed coverage reports let you combine multiple test runs in a single document (for each source file).

Detailed Coverage Report

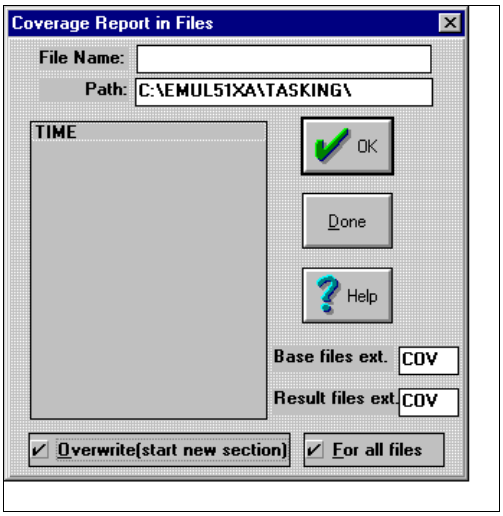


Figure 16: Producing Detailed Coverage Reports

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

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

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

Warning:	***Be sure to close the coverage window when analysis is done.***

Menus

The primary means of controlling the debugger, thus the emulation, is through menus. The EMUL51XA-PC 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 to the right of the item name, and also 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 EMUL51XA-PC that writes "S" record files. Holding down the Shift key or turning on CAPSLOCK is not necessary. Even though the keyboard shortcuts are all shown in capital letters, the shortcuts are not case-sensitive.

File Menu

Load code ..	F3	Loads an absolute file.

EMUL51XA-PC supports many popular compiler object file formats.

Load default symbols ..	<Alt> FC	Loads symbols defined by the MCU manufacturer.

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

Save code as ..	<Alt>FS	Writes the contents of RAM or ROM to a HEX record file.

Any region of memory can be saved to a file for reloading later. Selecting this menu item opens a dialog box that lets you select an address range. Please note that only the S19 file format is supported at this time.

Remove Symbols

Remove Symbols	<Alt>FR	Deletes all line number and symbolic information.

Show Load Info

Show load info ..	<Alt>FI	Displays 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 17.

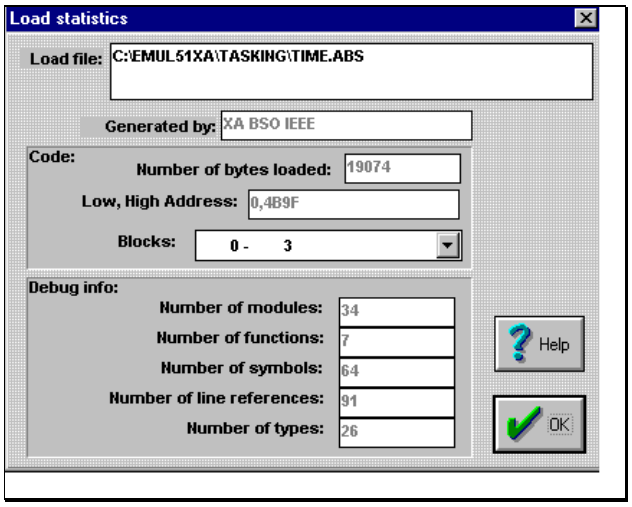


Figure 17: Example Load Statistics

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

Preferences	<Alt>FP	Controls the way the emulator loads object files.

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

Exit

Exit	<Alt>X	Quits EMUL51XA-PC

Exiting the EMUL51XA-PC software will update the current debugger configuration to either the .ini file or to the current .pro file, if one is selected.

View/Edit Menu

Copy to clipboard	<Ctrl><ins>	Copies text (without formatting or font information) of the entire active window to the clipboard.
User defined symbols	<Alt>VU	Opens a dialog box that lets you select the module from which you can view symbols.
Default CPU symbols	<Alt>VD	Views and edits memory-mapped registers by name and by the bit.
C call stack ..	<Alt>VC	Opens a child window that displays the C call stack and passed parameters needed to reach the current Program counter.
Evaluate ..	<Ctrl>E	Opens 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	Opens a dialog box that displays the contents of a single variable, structure, or array in detail.
Add a watch point ..	<Ctrl>W	Opens a child window that displays groups of variables that is updated every time emulation halts.
Search..	<Ctrl>S	Opens 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 page 65 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 backward.

Run Menu

Step into	F7	Executes one instruction, including a jump instruction. If a **Source** window is selected, executes all the instructions for one line of source.
Step over	F8	Executes one instruction or all the instructions in a subroutine. If a **Source** window is selected, executes all the instructions for one line of source. Due to some kinds of optimizations, this feature may not always be available.
Animate ..	<Ctrl>F7	Executes instructions continuously and slowly,

		highlighting each instruction or each line as it is executed.
Go	F9	Begins executing instructions from the current PC at full speed until the next breakpoint.
Go to cursor	F4	Executes the instructions from the PC to the current cursor position.
Go to ..	<Ctrl>F9	Executes the instructions from the PC to the specified address.
Go to return address	<Alt>RN	Executes the instructions from the PC to the next found function return. Due to certain optimizations, this feature may not always be available.
Go FOREVER	<Alt>RF	Executes instructions from the current PC after disabling all breakpoints.
Break Emulation	F9	Suspends execution as if a breakpoint was encountered.
Soft Reset (get vector)	Alt FS	
Reset Chip!	<Ctrl>F2	Resets CPU without executing any instructions.
Reset Chip and Go		Resets CPU and begins execution from reset vector when emulator is already running.

Breakpoints Menu

Note: Customers setting breakpoints normally use a software breakpoint. We suggest you use a hardware breakpoint only when your code is in PROM or EPROM. A hardware breakpoint will result in a one instruction skid (version 1.01 or later).

Toggle	F2	Disables or enables existing breakpoints.
Hardware breakpoints..		Opens a dialog box that lets you set up address ranges for hardware breakpoints (that don't

use the trace board).

Break on external data access .. Permits breaking on external read or write cycle; address resolution is 16 bytes. (See Figure 18.)

Hardware breaks only If this menu item is checked, all Program/source window breakpoints will result in hardware breakpoints.

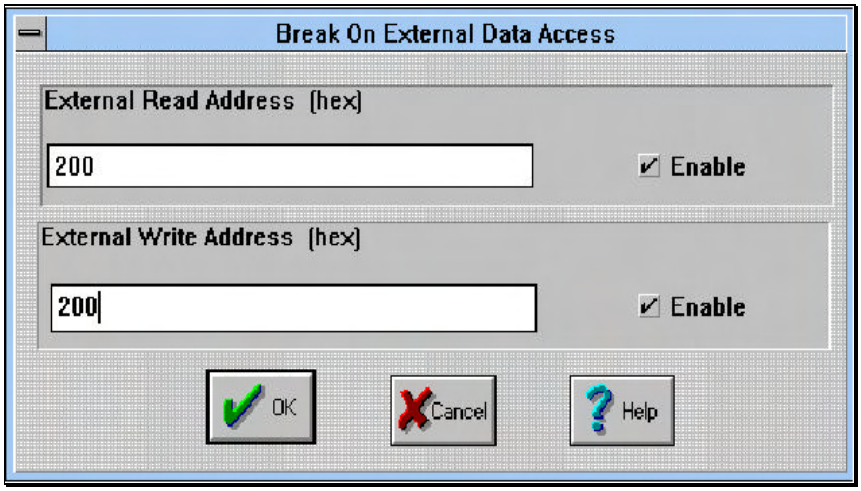


Figure 18: Break on external data access

At ..	<Alt>F2	Sets a breakpoint by address, line, or line in module.
Setup ..	<Alt>BS	Opens a breakpoint editing dialog box.
Disable all	<Alt>BI	Disables all breakpoints from being active while remaining in the list.
Delete All	<Alt>BD	Clears all existing breakpoints.
Break now!	<Ctrl>C	Immediately halts the emulation.

Config Menu

Project name ..	<Alt>CN	Chooses a configuration or project from a list of existing projects, or

		creates a new one.
Paths ..	<Alt>CP	Sets the default directories for finding load files, source files, and emulator files.
Memory map ..	<Alt>CM	Assigns memory to either emulation RAM or the target (16 byte resolution).
Emulator Hardware ..	<Alt>CE	Sets the emulator board address, controller type, and Special Function registers reset values.
Miscellaneous ..	<Alt>CM	Sets automatic PC & SP reset value, DDE sampling interval, and memory scroll range values.
Full Reset	<Alt>CF	Reloads on-pod logic & performs reset.
Color ..	<Alt>CC	Assigns colors to windows.
Trace ..	<Alt>CT	Please refer to Chapter 4 for information about the Trace Setup dialog box.
Memory Coverage ..	<Alt>CV	Opens the dialog box that controls Memory or Code coverage. See earlier detailed discussion.
PP Analyzer	<Alt>CA	Opens a Performance Analysis control window and starts recording addresses.

These next nine menus 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	Scrolls the selected **Program** window to the specified address.
Origin (at Program counter)	<Ctrl>O	Scrolls the **Program** window to display the PC address.
Set new PC value at cursor	<Ctrl>N	Sets the Program counter to the address at the cursor.
Module	<Ctrl>F3	Opens a dialog box that allows quickly scrolling the **Program** window to the start of any module.
Function	<Ctrl>F	Opens 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	Scrolls (or opens) a Source window to show the source at the current **Program** window cursor.
Toggle breakpoint	F2	Enables or disables a breakpoint at the cursor.

Source Menu

Address..	<Ctrl>A	Scrolls the selected **Source** window to the specified address, which may be a function name or a label.
Origin (at Program counter)	<Ctrl>O	Scrolls the **Source** window to display the Program counter address.
Set new PC value at cursor	<Ctrl>N	Sets the Program counter to the address at the cursor.
Module	<Ctrl>F3	Opens a dialog box that allows quick scrolling the **Source** window to the start of any module.
Function	<Ctrl>F	Opens 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 ..	<Alt>SC	Opens a window that displays the C call stack and passed parameters to reach the current Program counter.

View assembly code	<Ctrl>V	Scrolls (or opens) a **Code** window to the current source window cursor).
Toggle breakpoint	F2	Enables or disables a breakpoint at the cursor.

Data Menu

Address..	<Ctrl>A	Scrolls the selected **Data** window to the specified address.
Original Address	<Ctrl>O	Scrolls the selected **Data** window to the last address used in an Address.. menu command.
Edit ..	<Enter>	Alters the contents of the highlighted location.
Block move..	<Ctrl>B	Moves a segment of RAM to another location (in RAM).
Fill..	<Ctrl>F	Fills RAM with the specified value or pattern.
Data Display as..	<Ctrl>D	Sets the data display mode (ASCII, hexadecimal bytes, long integers, etc. See Figure 20 for the complete list of formats).
Address space..	<Alt>DS	Sets the address space for the selected **Data** window.
Inhibit readback on	<Alt>DI	Toggles data window to prevent read-after-write check.

Register Menu

Edit	<Alt>/E/E	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.
Modify Display	<Alt>EM	lets you un-select registers from the window

Trace Menu

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

Stack Menu

PaRameters in Hex	<ALT>SP	Displays the function paRameters in hex instead of in their declared type.
Show function	<ALT>SS	Not implemented at this time.

Watch Menu

Add ..	<Insert>	Opens a dialog box for adding a variable to the **Watch** window. You may mouse to select the variable, structure member (point to right of .), or string element.
Edit ..	<Enter>	Opens a dialog box for editing an existing variable in the **Watch** window.
Remove ..	<Delete>	Deletes 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. Detailed operation follows the hot-key descriptions.

New window Program	Alt>WNP	Opens a new **Program** window.
Source	<Alt>WNS	Opens a **Source** window.
Data	<Ctrl>WND	Opens a **Data** window.
Registers	<Alt>WNR	If one is open, it will ask, “Are you sure?”
Special Registers	<Alt>WNE	
Trace	<Alt>WNT	Opens a new **Trace** window.

Watch	<Alt>WNW	Opens a new **Watch** window
Toggle help line	<Alt>WNH	Turns on or off text at the bottom of the EMUL51XA-PC window.
Repaint	<Ctrl>R	Repaints the screen.
Tile windows	<Alt>WT	Resizes and arranges the windows within the EMUL51XA-PC application.
Cascade windows	<Alt>WC	Resizes and overlaps the windows within the EMUL51XA-PC application.
Arrange Icons	<Alt>WA	Lines up any closed EMUL51XA-PC icons at the bottom of the main window.
Zoom	F5	Expands selected window to fill the EMUL51XA-PC window.
Next window	<Ctrl>F6	Changes the currently selected (highlighted) window.
Close	<Ctrl>F4	Closes the currently selected window.

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

Help Menu

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

The other help selections are self-explanatory.

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 EMUL51XA-PC: **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 EMUL51XA-PC exits, and will be restored when the software restarts.

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

Register Windows

The **Registers** window displays the CPU registers. All registers are displayed in hexadecimal notation. Clicking anywhere in the **Registers** window will select that window (make it the active window).

Local menu device lets you **Edit** a register value and **Modify** the included registers.

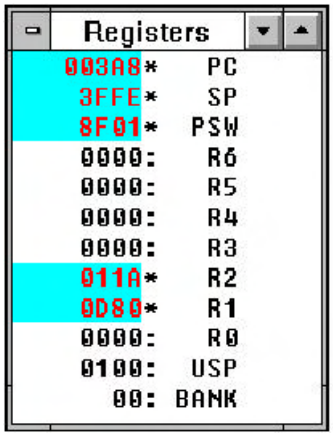


Figure 19: Registers Screen

Data Windows

Use **Data** windows to examine or modify emulation or target memory directly. EMUL51XA-PC uses the controller to read and write RAM, so the **Data** window cannot be updated while the emulation is running. Instead, asterisks will be displayed until data can be read from memory.

Data can be displayed or modified in a variety of formats as shown in Figure 20. Keep adding new windows for each display format and starting address.

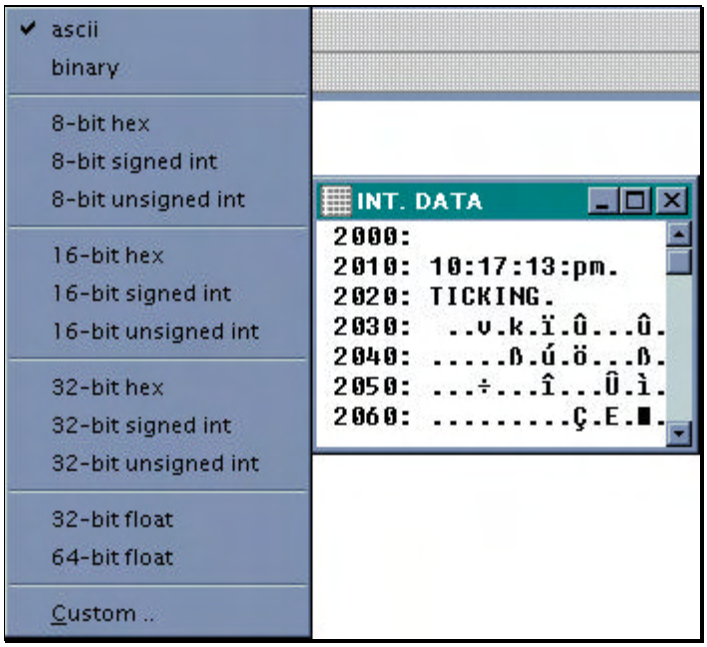


Figure 20: Data Window Menu

Selecting any **Data** window displays the **Data** menu shown above.

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.

Figure 21: Editing Memory with a Data Window

On the far right side of the Edit data dialog box is a small check box labeled with the letter **C**. This box impacts how the emulator interprets the data you enter. For example, if you have a symbol named "abcd" and you are displaying in 16-bit hex, it is not clear whether to interpret "abcd" 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 21).

Data windows can be assigned to read from and write to either internal or external code space, data space, or Shadow RAM.

Shadow RAM is a separate data space on the IETR that reflects (“shadows”) all external writes to memory. **SHADOW** windows update approximately four times per second without any intrusion on CPU execution.

Special Registers

This window displays SFRs, including individual SFR bits. Change any value by selecting the SFR of interest and using the <Enter> key. The right mouse displays user options. Use the **Add** option to build a watch-like window of SFRs and individual SFR bits of interest.

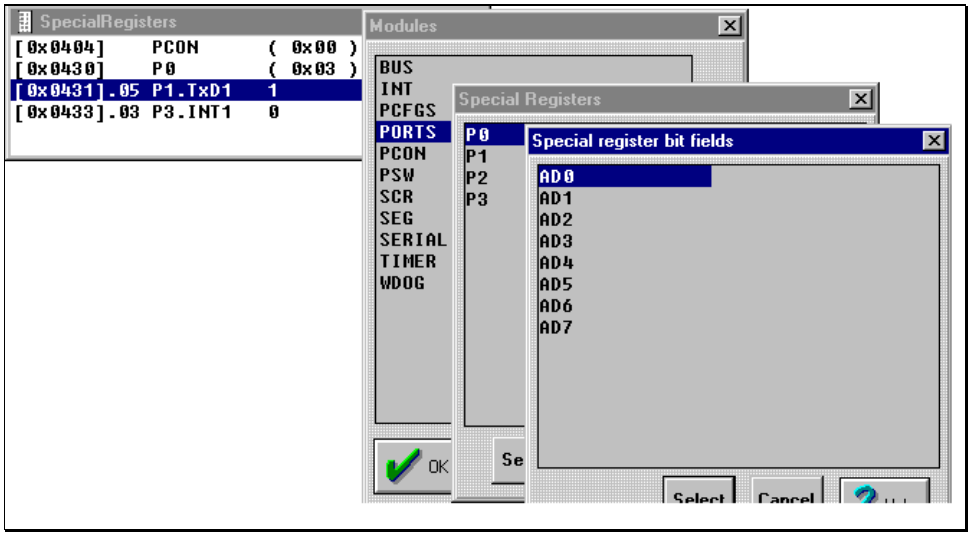


Figure 22: View SFRs

Another method of viewing SFRs is with the **View/Edit** menu. You will be able to view all SFR bits as well as bit usage in a dialog box.

Warning:	***Caution should be exercised when changing some SFRs such as PCON. Setting the PD bit here will cause Power-Down mode, shutting down the bondout clock and resulting in loss of communication to the emulator board.***

Custom Display Format

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

Program Windows

A **Program** window disassembles and displays code memory. One line in the **Program** window is always highlighted. This is the cursor. The color of the highlighting and the window depend upon how you have configured your color settings. (See page 18 for information about how to change the color settings.)

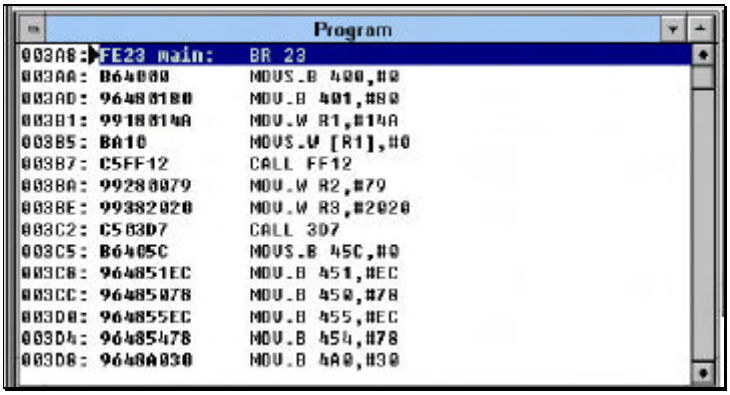


Figure 23: Program Window

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 set a breakpoint. A breakpoint is indicated by displaying the address with white letters on a black or dark background. This second mouse click (not a double-click) creates the breakpoint. To deactivate (not delete) that breakpoint, click again on the same instruction. The address will no longer be highlighted and the breakpoint will be inactive. To delete the breakpoint, use the **Setup..** dialog box from the **Breakpoints** menu. Any highlighted instruction can be a temporary breakpoint. The **Run** menu item **Go until cursor, F4,** will use the cursor as a temporary breakpoint.

In-line Assembler

Simply highlight the instruction or address you wish to change in the **Program** window. 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.

The in-line assembler will translate the input line according to the syntax described in the P51XAG3 data books and replace the former opcode(s) and data with the new opcode(s) and data. Note that the assembler will write as many bytes as required for the new instruction. This may overwrite part or all of subsequent instructions. Be sure to examine the subsequent instructions as well as the new instructions for correctness.

Source Windows

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

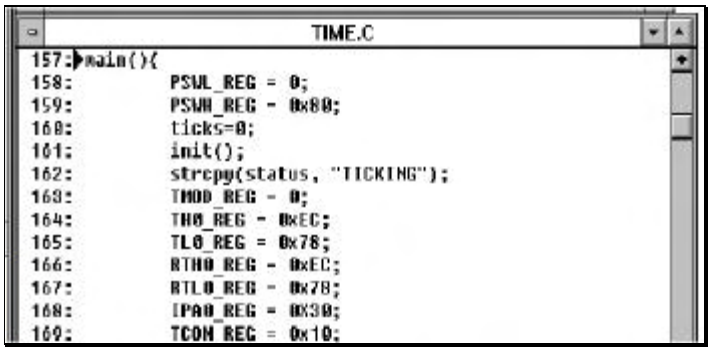


Figure 24: Source Window

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.

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.

Trace Window

For information about the **Trace** window, please refer to Chapter 4: Internal/External Data Trace Board.

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 (see Figure 25). These windows are opened by selecting their respective items in the **View/Edit** menu.

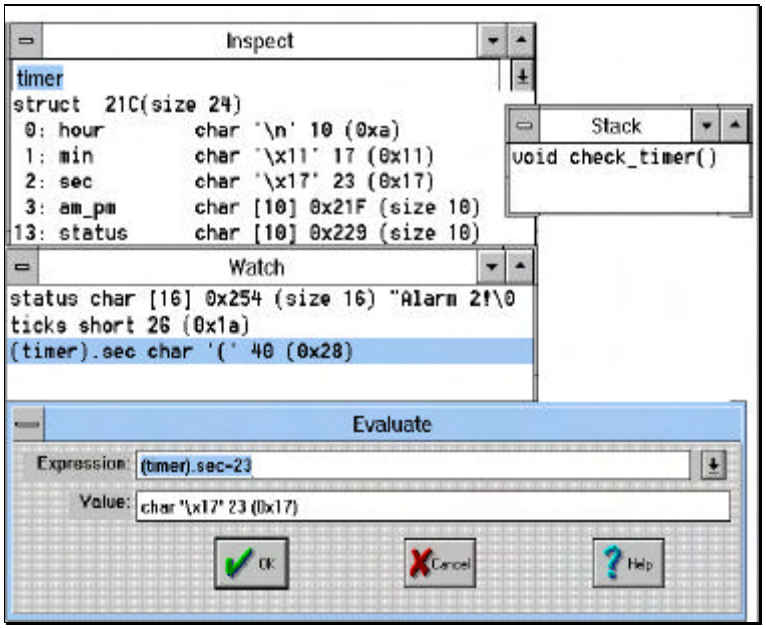


Figure 25: C call stack, Evaluate, Inspect & Watch Windows

The **Inspect** window is useful when displaying structures or arrays. To open an **Inspect** window, either select the **Inspect ..** menu item in the **View/Edit** menu or double-click in the **Source** window on the variable you would like to inspect. Double-click in an open **Inspect** window on a structure member or array element to open an **Inspect** window detailing that field.

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

*Note: If you have an open **Inspect** window with an assignment statement, every time the emulator stops executing, the expression will be evaluated and the variable will be updated. Assignments are better done with the **Evaluate** window.*

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

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

Evaluate Window

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

Hint: *This window can also serve as a hexadecimal calculator, using the C syntax 0x____ for hex numbers. For example, to determine a timer overflow value from 50000 decimal, simply type 0xFFFF - 50000.*

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 paRameters passed to them.

Addresses are displayed and entered using hexadecimal notation or global symbol names.

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 **Help** button opens the MS Windows Help application to the page that describes the current context. The **Reset** button resets the controller. The **Step** button emulates one source line or opcode depending upon which window was last active. The **Go** button starts full speed emulation that will continue until a break occurs. While emulating, the **Go** button changes to **Break**, and halts emulation when clicked. The **Trace Beg** button resets the trace board and starts bus cycle recording according to the conditions set in the **Trace Setup** dialog box.

Figure 26: The Tool Bar

Help Line

At the bottom of the EMUL51XA-PC window is a line of text that, depending upon the context, explains what the selected item is or what it does. This kind of context-sensitive help is turned on and off with the **Toggle help line** item in the **Windows** menu.

Chapter 2: Emulator Macro User Guide

Introduction

The Nohau Emulator Macro System consists of two program files together with this guide. The System requires the 16-bit version of *Microsoft Visual Basic 4 Professional Edition,* or the standard edition of VB3. Macro creation uses many of the features of *Visual Basic*, such as debugging. The macro writer uses a low-level DLL and a library of useful functions. Additional functionality is easily added by the macro writer. Run macros from *Visual Basic* for testing and debugging or as stand-alone executables.

Note: Install Visual Basic before using these files.

Currently only VB5 is sold at retail outlets. Order the Learning Edition, then call Microsoft to downgrade to VB3 standard edition with its 16-bit DLL support. If you want features in the Professional Edition, order VB5 Professional and downgrade to VB4 Professional.

General description of the emulator macro setup

There are two files provided in the Emulator directory:

emul51xa.bas	the *Visual Basic* API
emul51xam.ico	the icon to use for executable macros

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

Visual Basic Supplemental User Guide

It's easy to write macros:

1. Execute Visual Basic
2. Set up your icon
3. Make the form invisible
4. Add the emul51xa.bas file to your project

5. Name your project
6. Write your macro code
7. Create your executable macro
8. Exit Visual Basic
9. Test it!

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

Warning:	***DO NOT change emul51xa.bas. Use another module if additions are needed.***

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

Require Variable Declaration	**=**	**Yes**
Syntax Checking	**=**	**Yes**
Default Save As Format	**=**	**Text**
Save Project Before Run	**=**	**Yes**

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

Procedure for writing a macro

1. Execute Visual Basic, or if it is running, create a new project from the Microsoft Visual Basic main menu with **File/New Project**.
2. Set up an icon in the main form **Properties** window (use the icon "emul51xam.ico" for this purpose).
    a. In the **Properties** window, click on the ICON selection.
    b. At the top of the **Properties** window, click on the "..." button.
    c. In the **Load icon** dialog box, select the directory where the icon is located (i.e., c:\emul51xa).
    d. Select the emul51xam.ico icon under **File Name**; Click **OK**.

3. Set visible to “false” in the property window to make the main form invisible.
    a. In the main form **Properties** window, select the “visible” entry.
    b. Click drop down arrow in the **Change** window.
    c. Click on **FALSE**.

4. In the *Microsoft Visual Basic* **Main Menu**, select **File/Add File..** to add the emul51xa.bas file to the project.
    a. In the **Add File** dialog box, select the c:\emul51xa directory.
    b. Select emul51xa.bas; click **OK**.
5. Name the form in the **Project** and save.
    a. In the **Properties** box, click on **NAME**.
    b. Type in the new name (e.g., "LoadTime").
    c. In the *Microsoft Visual Basic* main menu, select **File/Save Project As..**
    d. Save changes to "LoadTime.frm"? Choose **Yes**.
    e. The **Save File As** dialog box should show "LoadTime.vbp" in c:\emul51xa directory; click OK**.**
    f. In the **Save Project As** dialog box, change the file name of the project file to "LoadTime.mak"; click OK**.**
    g. The result is a project window with the name "LOADTIME.VBP"

6. Write code in the Form_Load subroutine.
    a. In the **Project** window, select the form ("LoadTime.frm").
    b. Click the **View Code** button.
    c. In the **Forms Code** window, select "FORM" from the object drop-down box.
    d. Type in the line (indented): "EmulInit"; press Enter.
    e. Type in the body of the macro code (see example below).

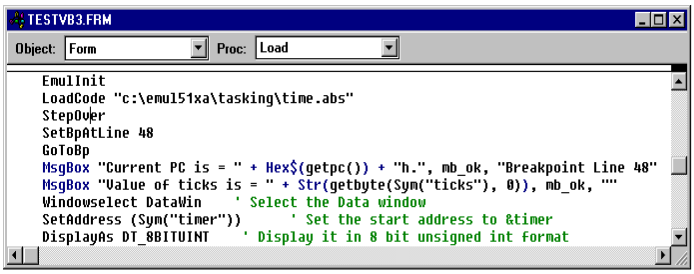


Figure 27: Macro Code Example

f. Type "END ", allowing the macro to exit.

7. When the Program is finished, make an executable Program from the **File** menu using the **File/Make.EXE** command.

   a. Select **File/Make.EXE**.

   b. The MAKE.EXE dialog box appears; check to ensure it shows the correct file name and directory.

NOTE: Change the application title at this time if desired.

   c. Click OK.

8. **File/Exit**; For any "Save Changes?" Choose **Yes**.

Example of a Macro:

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

SUBROUTINE	DESCRIPTION
Option Explicit	Force explicit declaration of variables.
Sub Form_Load ( )	This subroutine executes when the macro loads and appears in the **Form, Load** section of the main form.
Dim byt As Integer	Declare byt as an integer variable.
Dim ret As Integer	Declare ret as an integer variable.
Dim dwd As Long	Declare dwd as a long variable.
Const ShadowRam = 1	Const ProgramWin = 2

SUBROUTINE	DESCRIPTION
Const RegisterWin = 3	Const DataWin = 4
Const SourceWin = 5	Constants for activating the windows of project:proj1 (For the current version of the macro language, we suggest that you use “project” to specify the MDI child window locations and their order. See “Projects” on page 10.)
EmulInit	Initialize the macro library and the emulator (if not running).
LoadProject “proj1”	Load project setup
LoadCode “c:\emul51xa\tasking\time.abs”	Load the code and symbol file.
WindowSelect SourceWin	Select the **Source** window.
StepOver	Step a couple of times.
WindowSelect DataWin	Select the **Data** window.
SetAddress &H5000	Set the start address to 0x5000
DisplayAs DT_8BITHEX	Set the format to 8-bit hex; the DT_xxx constants are found in emul51xa.bas.
WindowSelect ShadowRam	Select the **ShadowRam** window.
DisplayAs DT_ASCII	Set the format to ASCII.
WindowSelect ProgramWin	Select the **Program** window.
SetAddress &H3000	Set the address to 0x3000
SendKeys “nop{ENTER}nop{ENTER}nop{ENTER} jmp 3000{ENTER}”, True	Assemble some code into the **Program** window. The last paRameter (True) is used to wait for SendKeys to finish before going on the next statement.
PutPC &H3000	Set the PC to the assembled code.
Byt = 0	Clear the byt variable

SUBROUTINE	DESCRIPTION
ret = GetShadowByte (&H5010, byt)	Get the byte at 0x5010.
MsgBox "Shadow byte at 0x5010 = " + Hex$ (byt) + "h,", MB_OK, "LoadForm"	Display the value (MB_OK is found in emul51xa.bas) in a message box.
Go	From 0x3000
For dwd = 0 To 100	A wait loop
DoEvents	Let another Windows application run.
Next	
Break	Stop executing
dwd = GetPC ( )	Get the current PC into a previously declared variable "dwd".
MsgBox "The PC is: Ox" + Hex (dwd) + ".", MB_OK, "Test Macro"	Display the variable "dwd" as a hex value in a message box.
EmulReset	Prepare to run the previously loaded code module.
SetBpAtLine 138	Set a breakpoint at source line 138 in the current source module.
GoToBP	Go until a breakpoint is encountered
End	Exit from macro.
End Sub	

9. Enter **File Manager** and select emul51xa directory.
    a. Drag the executable file (in this case "LOADTIME.EXE") from the **File Manager** into a file folder in the **Program Manager**.
    b. Execute the Program by double-clicking on the icon.
    c. Drag the project file ("LOADTIME.MAK") from the **File Manager** into a file folder in the **Program Manager**.
    d. To edit your macro, execute *Visual Basic* by double-clicking on the icon produced from step c.

Subroutine Reference

Constants

Use constants for the *Visual Basic* message box subroutine and the **Display As** values for the different data formats. Refer to emul51xa.bas for the names of these constants. Any other constants needed are documented in the *Visual Basic* or *Visual Basic Professional Help* files.

Global Variables

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

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

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

Nohau Subroutines

Note: ***spacex*** *used in all read or write subroutines refers to data (0) or code (1) address space.*

SUBROUTINE	DESCRIPTION
Sub Break ( )	Stops emulator execution.
Sub DisplayAs (ntype As Integer)	Changes the current display format in the active window. (Note that this can only be used in windows that have a display format.) The window must be made active with Window Select.
Sub EmulInit ( )	Sets up the macro for execution and starts the emulator Program if it is not running.
Sub EmulReset ( )	Performs a normal emulator reset.
Function FindEmulWindow ( ) As Integer	Used internally to set up the global variables, and is not normally used in macros.
Function GetByte ByVal address As Long, spacex As Integer	Gets a byte from data memory.
Function GetDWord ByVal address As Long, spacex As Long	Gets a long or double word from data memory.
Function GetPc ( ) As Long	Gets the current value of the Program counter.
Function GetPsw ( ) As Long	Gets the current PSW.
Function Get ShadowByte ByVal address As Long, spacex As Integer	Gets a byte from ShadowRam.
Function Get ShadowWord ByVal address As Long, spacex As Integer	Gets a word from ShadowRam.
Function GetSp ( ) As Long	Gets the current stack pointer value.
Function GetWord ByVal address As Long, spacex As Integer	Gets a word from data memory.

SUBROUTINE	DESCRIPTION
Sub Go_ ( )	Starts executing the emulator at the current address.
Sub GoToBP ( )	Starts executing the emulator from the current address and waits until a breakpoint occurs.
Sub LoadCode (codefilename As String)	Loads a Program for emulation.
Sub LoadProject (pname As String)	Loads an emulator project setup file.
Sub ModuleSelect (module As String)	Selects a new module in the current Program.
Sub PutByte ByVal address As Long, byte, spacex As Integer	Writes a byte into the data memory.
Sub PutDWord ByVal address As Long, ByVal dwrd As Long, spacex As Integer	Writes a long or double word into data memory.
Sub PutPc (ByVal address As Long)	Sets a new value in the Program counter.
Sub PutPsw (ByVal pswreg As Long)	Sets a new value in the PSW.
Sub PutSp (ByVal address As Long)	Sets a new value in the stack pointer.
Sub PutWord ByVal address As Long, ByVal wrd, spacex As Integer	Writes a word into data memory.
Sub RePaint ( )	Redisplays everything in the emulator main window.
Sub SaveTraceText (filename As String, startframe As Integer, stopframe As Integer)	Save the current trace from the “startframe” number to the “stopframe” number into the file name specified as a text file.

Sub SetAddress (address As Long)	Sets an address in all windows that can have an address setting. The correct window must be made active with **Window Select**.
Sub setBpAtAdr (ByVal address As Long)	Sets a breakpoint at the requested address.
Sub SetBpAtLine (ByVal lineno As Integer)	Sets a breakpoint at the requested line number.
Sub SetTrigger (ByVal trignun As Integer, ByVal active As Integer)	Make the trigger “1,2,3, or filter” active or inactive.
Sub StepInto ( )	Executes one instruction, including a jump instruction.
Sub StepOver ( )	Executes one instruction or all the instructions in a subroutine.
Function Sym (Symname As String) As Long	Gets numeric value of a symbol.
Sub WaitUntilReady ( )	Wait for the emulator to become READY.
Sub WindowSelect (number As Integer)	Activate one of the current child windows, e.g., **Data** window, by the number on the Window pop-up menu.

Chapter 3: Emulator Board

EMUL/LC-ISA Emulator Board

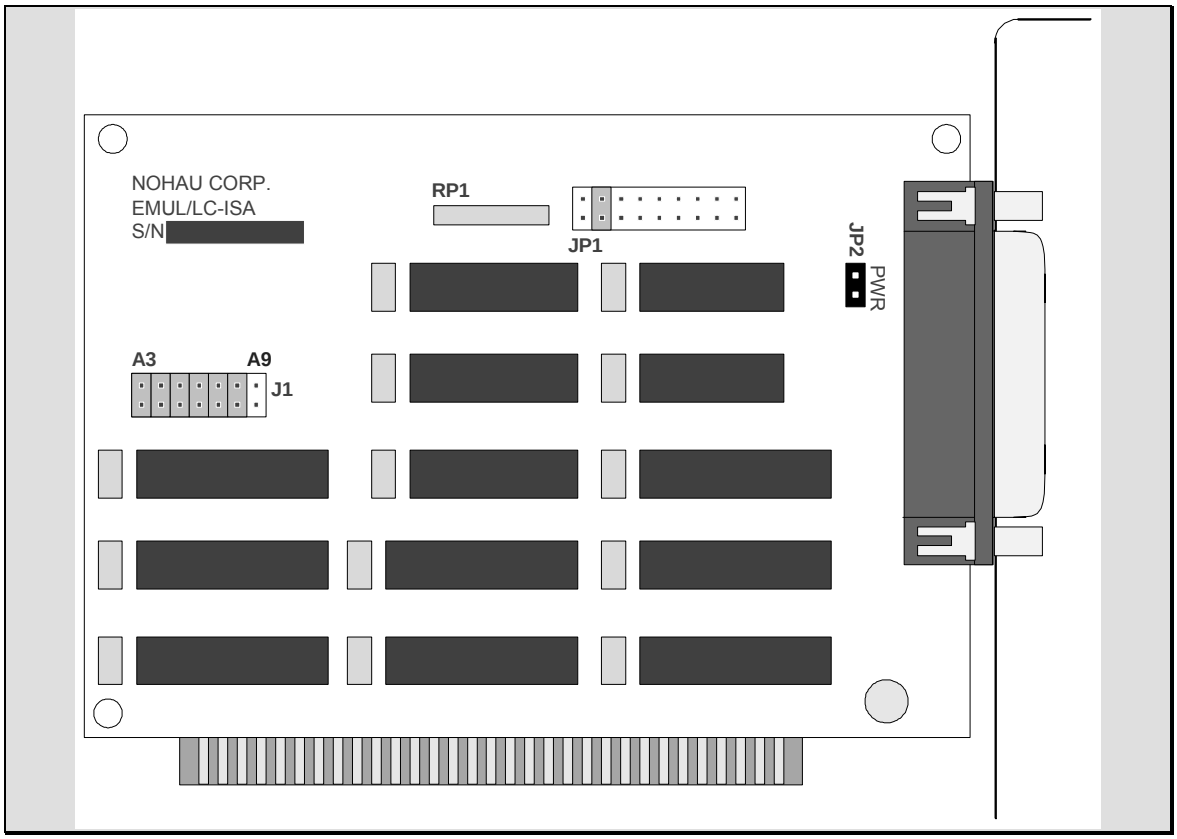


Figure 28: EMUL/LC-ISA Emulator Board

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

Note: The power jumper J2 should be left in for EMUL51XA-PC users. Existing users of the ¾ length, 16-bit emulator cards can use these in place of the EMUL/LC-ISA card.

Detailed Installation Instructions

Setting the I/O address jumpers—J1

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

The EMUL/LC-ISA requires 8 consecutive I/O addresses from the PC's I/O address space (0 Hex -- 3FF Hex) that begin on an address that is a multiple of 8. Set the emulator board address using the jumpers in header J1. These addresses must not conflict with any other I/O device.

Each pair of pins in J1 represents 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 pins represents a 0 in the address. A pair of pins with no jumper represents a 1. Below are four examples where the Least Significant Bit (LSB) is on the left, as it is on the board if you are holding the board so you can read the silk-screened labels, with the 25-pin D connector on the right.

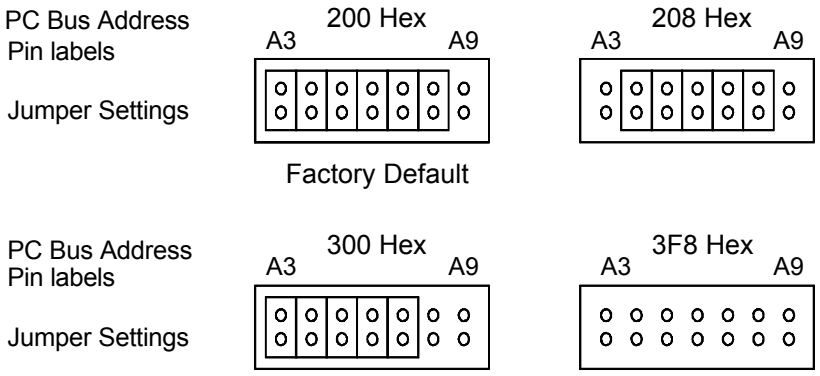


Figure 29: Emulator Header J1

Setting the Target Communication Rate – Header JP1

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

Set the fixed synchronous communication rate by using Figure 30 to look up the clock rate in the lower row and place one jumper on the header JP1,

between the pins indicated in the upper row. There must be only ONE jumper on this header.

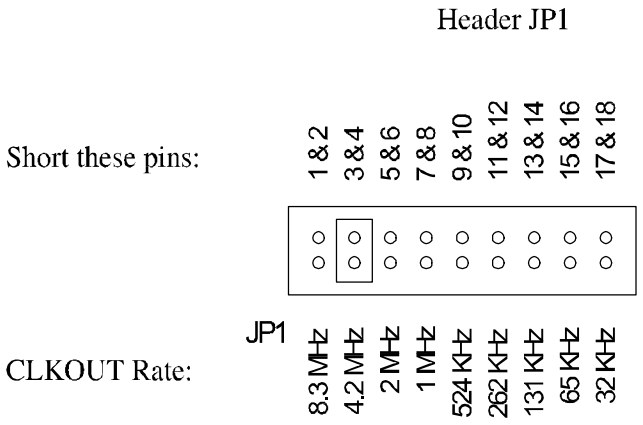


Figure 30: Header JP1

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

Communication Rate Jumper

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

The PWR Header – JP2

Note: Leave this jumper in place.

With the jumper in place, +5 volts are supplied from the PC's power supply through the LC connector, up to .5 amps.

Power Supply to Pod/Target

The power supply to the pod/target is controlled by jumper(s) on the pod boards. See Chapter 5: Pod Boards, or refer to specific chapters on pods 51XA/G3/I and 51XA/G3/E.

Chapter 4: Internal/External Data Trace Board

EMUL51XA-PC needs RAM to record a history of the data used and instructions executed. The trace board contains this RAM, and the pod board has the logic and connectors necessary to support a trace board. The IETR card includes 128 bits of RAM for each trace record. Trace boards are available in two trace depths: 128K or 512K. It supports both 5.0 VDC and 3.3 VDC low-voltage operation (See VCC option in **Trace Setup** screen.).

Detailed Installation Instructions

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

If one is installed, click on the **Yes** button next to **Board installed:** . The I/O address may be left at the default, 208.

Finally, if running 3.3 VDC target select, correct target VCC: 3.3 VDC (See Figure 48).

Trace Board Power Jumper

The power jumper is located between the DB15 and the battery eliminator connector. The jumper's name and function are different on rev C and rev D trace boards.

Rev C: U1 (three pin)

External power default:	jumper positioned away from edge of board
Internal power:	jumper positioned closest to edge of board

Rev D: JP1 (two pin)

External power default:	jumper not installed
Internal power:	jumper installed

Note: External power is recommended.

External Inputs and Controls

The Trace board records eight external digital inputs with every frame. These signals are input through the 15-pin D connector. Additionally eight digital inputs are recorded from JP15 Pod inputs.

The trace sampling resolution can be at the bus or clock frequency, depending on the selection **Recording** per clock in the **Trace Setup** screen.

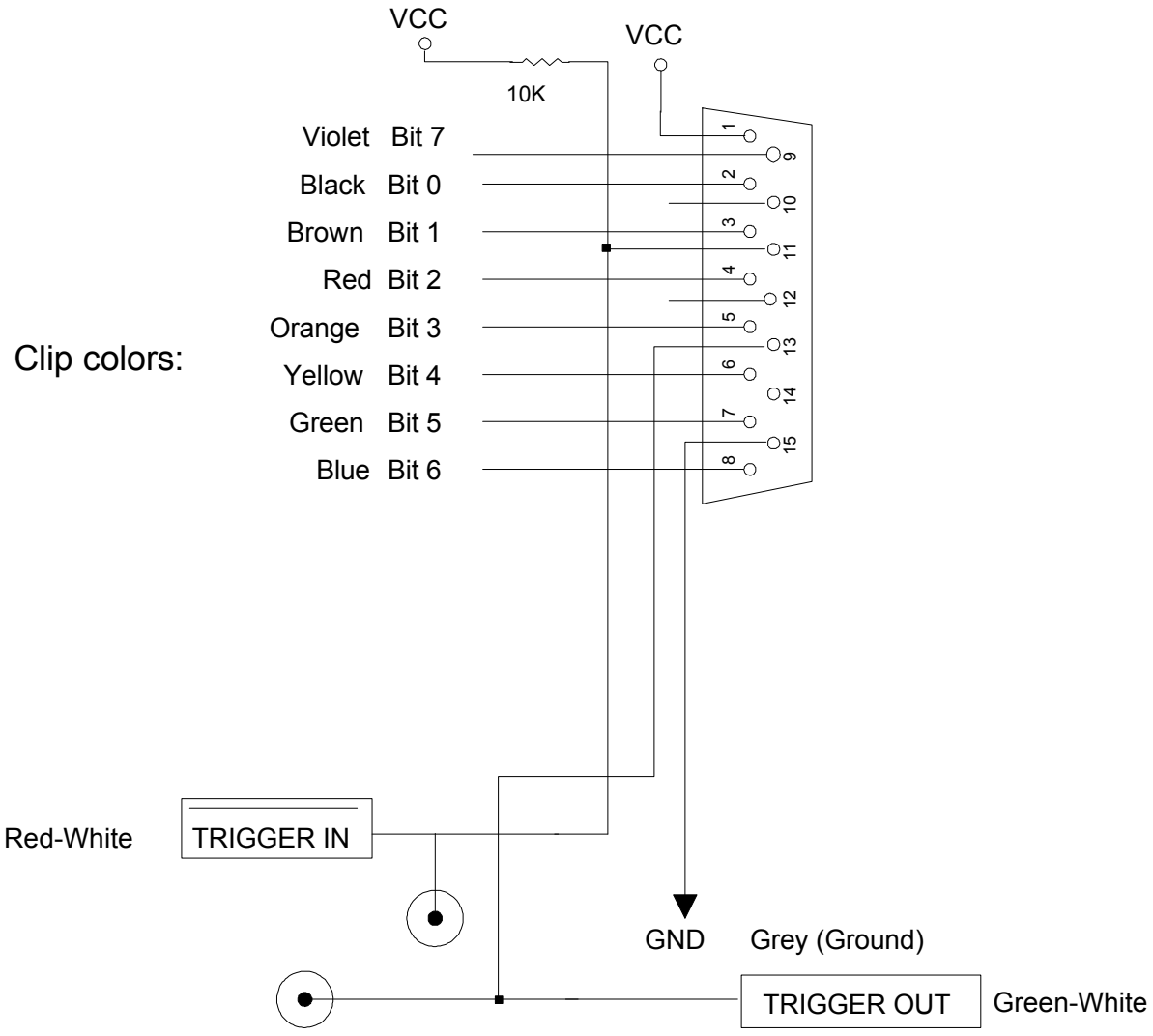


Figure 31: Trace Board Connectors

Two of the microclips duplicate the trigger controls found in the SMB connectors: TRIGGER IN is inverted. TRIGGER OUT goes HIGH for the duration of a valid trigger approximately 250ns using a 20 Mhz clock.

The TRIGGER IN microclip can prevent triggering when this line is held low and **Trig Inhib** selected. 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. See Figure 48 and text for further discussion.

Introduction to Tracing

A trace history is a time ordered recording of bus or clock cycles (with some other helpful information). Events that do not affect the CPU internal or external bus, such as testing a CPU internal data register, will not get recorded. By default, with no filter or triggers defined, all external and internal bus activity is recorded. All tracing emulators record bus events and not actual instruction execution, so they all must have some way to deal with the effects of the instruction pipeline. The trace board for EMUL51XA-PC includes pipeline decoding and marks opcode fetches that are not executed. As a result, the display software can show the trace records as though the pipeline did not exist, but it can also display the uncorrected bus cycles just as they were recorded.

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

When **Filter** is enabled, every bus cycle is examined to see if it meets the conditions in the **Filter**: field of the **Trace Setup** dialog box. If it does 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 hardware breakpoints. These are useful if you are executing out of ROM or need to break on certain hardware conditions. For information about how to create triggers and hardware breakpoints, see **Triggers** on page 39.

Trace Window

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

Pipeline Effects

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

Bus Cycle Order

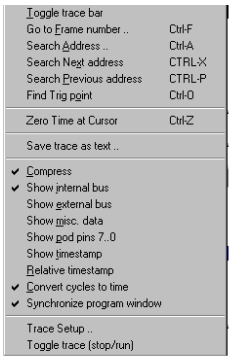
All bus cycles are shown in the order fetched, not in the order executed. The **Trace** window shows the executed fetches, with the last row possibly showing only fetched, not executed, instructions.

Bus Width

The trace buffer always records 16 bits of data for each bus cycle even though the other 8 bits may be ignored by the CPU.

Bus width is determined by the BUSW pin target connection, and cannot be set with software change to the BCR registers. See chapters 6 and 7 for further discussion under BUSWIDTH headers.

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.

Find frame number ..

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

Search Address

This menu item opens a dialog box, shown in Figure 32, to get the desired address, then searches from the beginning in the frame buffer for the first record that contains the specified address.

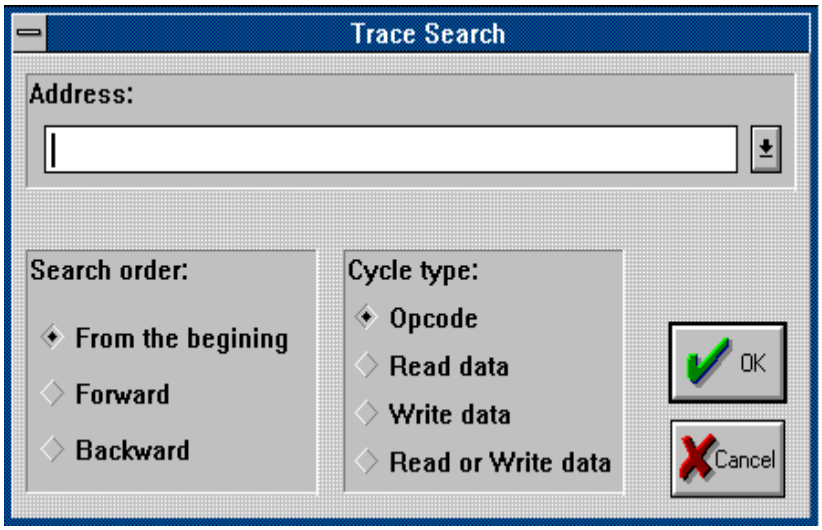


Figure 32: Trace Search Dialog Box

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.

Zero time at cursor

<CTRL>Z causes the selected absolute time stamp frame to be zeroed.

Save trace as text ..

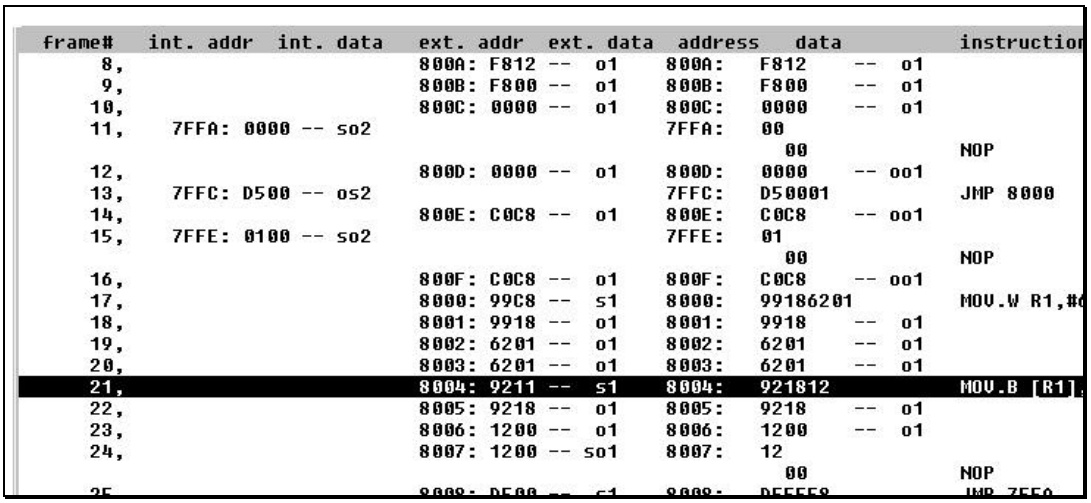
You can save any portion of the trace buffer to a text file suitable for inclusion in documents or processing with text manipulation or word processing tools. Selecting this menu item will open a dialog box that lets you set a range of frame and the name of the file where the text goes.

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 time stamps, arrange for the **Trace buffer** window to show them as well.

Compress/Uncompress

The **Compress** option post-processes the trace buffer information, discarding all unexecuted prefetches after branching institutions and code fetches of multiword op codes beyond the initial word fetch. This makes the trace display more readable and the saved trace text files smaller in length. The **Compress** option is also useful when verifying code-thread execution for FAA and FDA certifications. Make sure you save the compressed display as an ASCII text file.

Note the difference between uncompressed and compressed displays in Figure 33 and Figure 34:



frame#	int. addr	int. data	ext. addr	ext. data	address	data	instruction
8,			800A: F812 --	o1	800A:	F812 -- o1	
9,			800B: F800 --	o1	800B:	F800 -- o1	
10,			800C: 0000 --	o1	800C:	0000 -- o1	
11,	7FFA: 0000 -- so2				7FFA:	00	
						00	NOP
12,			800D: 0000 --	o1	800D:	0000 -- oo1	
13,	7FFC: D500 -- os2				7FFC:	D50001	JMP 8000
14,			800E: C0C8 --	o1	800E:	C0C8 -- oo1	
15,	7FFE: 0100 -- so2				7FFE:	01	
						00	NOP
16,			800F: C0C8 --	o1	800F:	C0C8 -- oo1	
17,			8000: 99C8 --	s1	8000:	99186201	MOV.W R1,#6
18,			8001: 9918 --	o1	8001:	9918 -- o1	
19,			8002: 6201 --	o1	8002:	6201 -- o1	
20,			8003: 6201 --	o1	8003:	6201 -- o1	
21,			8004: 9211 --	s1	8004:	921812	MOV.B [R1],
22,			8005: 9218 --	o1	8005:	9218 -- o1	
23,			8006: 1200 --	o1	8006:	1200 -- o1	
24,			8007: 1200 --	so1	8007:	12	
						00	NOP
25			8008: DFF8 --	s1	8008:	DFFFF8	JMP 7FF0

Figure 33: Internal and external uncompressed bus display using 8-bit wide external code memory

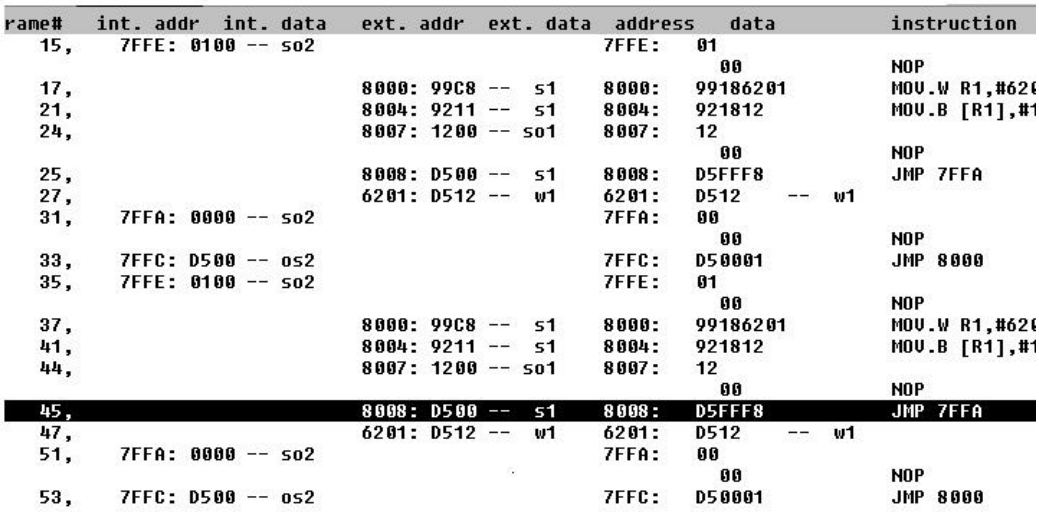
Bus Cycle Trace Annotation

Figure 33 contains keys to the bus cycle type and length in bytes. A number follows to indicate the number of bytes for the read, write, or opcode fetch cycle:

w1	for one-byte write cycle
r1	for one-byte read cycle (not shown in example)
os2	16-bit fetch with start-of- instruction on an EVEN address
so2	16-bit fetch with start-of-instruction on an ODD address
ss2	16-bit fetch with start-of-instruction on ODD and EVEN addresses (not shown, but could be two sequential NOPs)
s1	8-bit fetch start-of-instruction
so1	8-bit start-of-instruction on ODD address

Observe that this program runs in internal 16-bit wide code memory to address 8000, or 32KB, the current internal code memory ROM size selected when initializing the XA bondout in the EMUL_INI configuration screen, and thereafter runs in 8-bit wide memory (requires inserting JP4 buswidth jumper on the pod when running standalone without target board).

Figure 35 shows the same program running in 16-bit wide external code memory.



rame#	int. addr int. data	ext. addr ext. data	address data	instruction
15,	7FFE: 0100 -- so2		7FFE: 01	
			00	NOP
17,		8000: 99C8 -- s1	8000: 99186201	MOV.W R1,#62(
21,		8004: 9211 -- s1	8004: 921812	MOV.B [R1],#1
24,		8007: 1200 -- so1	8007: 12	
			00	NOP
25,		8008: D500 -- s1	8008: D5FFF8	JMP 7FFA
27,		6201: D512 -- w1	6201: D512 -- w1	
31,	7FFA: 0000 -- so2		7FFA: 00	
			00	NOP
33,	7FFC: D500 -- os2		7FFC: D50001	JMP 8000
35,	7FFE: 0100 -- so2		7FFE: 01	
			00	NOP
37,		8000: 99C8 -- s1	8000: 99186201	MOV.W R1,#62(
41,		8004: 9211 -- s1	8004: 921812	MOV.B [R1],#1
44,		8007: 1200 -- so1	8007: 12	
			00	NOP
45,		8008: D500 -- s1	8008: D5FFF8	JMP 7FFA
47,		6201: D512 -- w1	6201: D512 -- w1	
51,	7FFA: 0000 -- so2		7FFA: 00	
			00	NOP
53,	7FFC: D500 -- os2		7FFC: D50001	JMP 8000

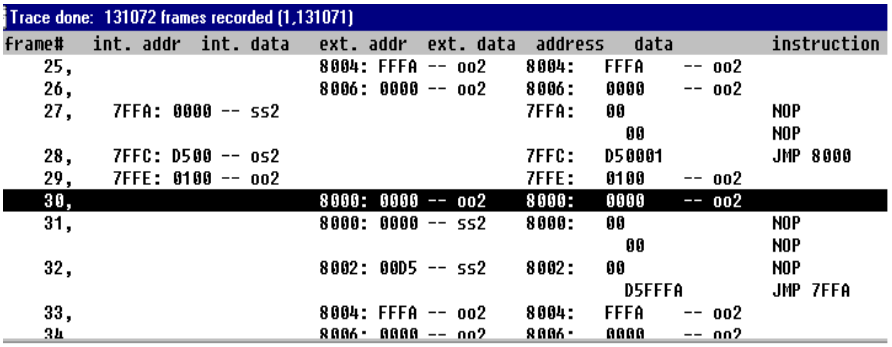
Figure 34: Internal and external compressed bus display using 8-bit wide external code memory

Show Internal Bus

This display shows all code that executes in internal code RAM under separate columns.

Show External Bus

This display shows external code and read/write bus cycles under separate columns.



Trace done: 131072 frames recorded (1,131071)

frame#	int. addr int. data	ext. addr ext. data	address data	instruction
25,		8004: FFFA -- oo2	8004: FFFA -- oo2	
26,		8006: 0000 -- oo2	8006: 0000 -- oo2	
27,	7FFA: 0000 -- ss2		7FFA: 00	
			00	NOP
28,	7FFC: D500 -- os2		7FFC: D50001	JMP 8000
29,	7FFE: 0100 -- oo2		7FFE: 0100 -- oo2	
30,		8000: 0000 -- oo2	8000: 0000 -- oo2	
31,		8000: 0000 -- ss2	8000: 00	NOP
			00	NOP
32,		8002: 00D5 -- ss2	8002: 00	NOP
			D5FFFA	JMP 7FFA
33,		8004: FFFA -- oo2	8004: FFFA -- oo2	
34		8006: 0000 -- oo2	8006: 0000 -- oo2	

Figure 35: Internal and external bus in compressed format using 16-bit wide external code memory

*Note: This sample code executes in internal code RAM until crossing the 32KB boundary to external EPROM at 8000H. Here, both **Show** options have been selected.*

Show misc data

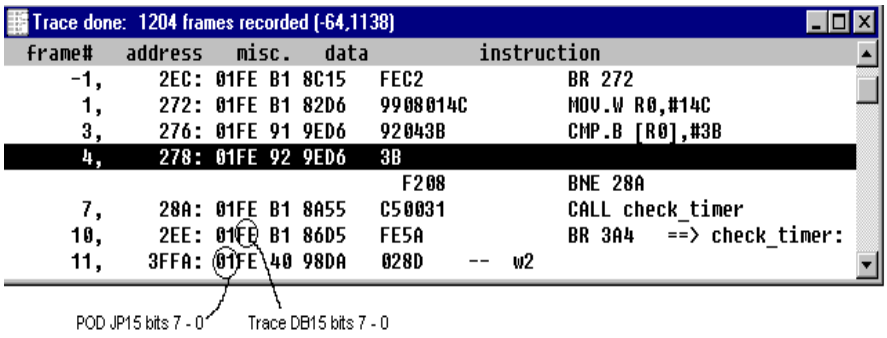


Figure 36: Miscellaneous External Data Display

Selecting this menu item will display another 32 bits for each record. The leftmost word records external signal lines from the DB15 connector, and 8 signals on the TRACE header on the pod. The remaining byte and word are used by Support and Engineering.

Show pod pins 7..0

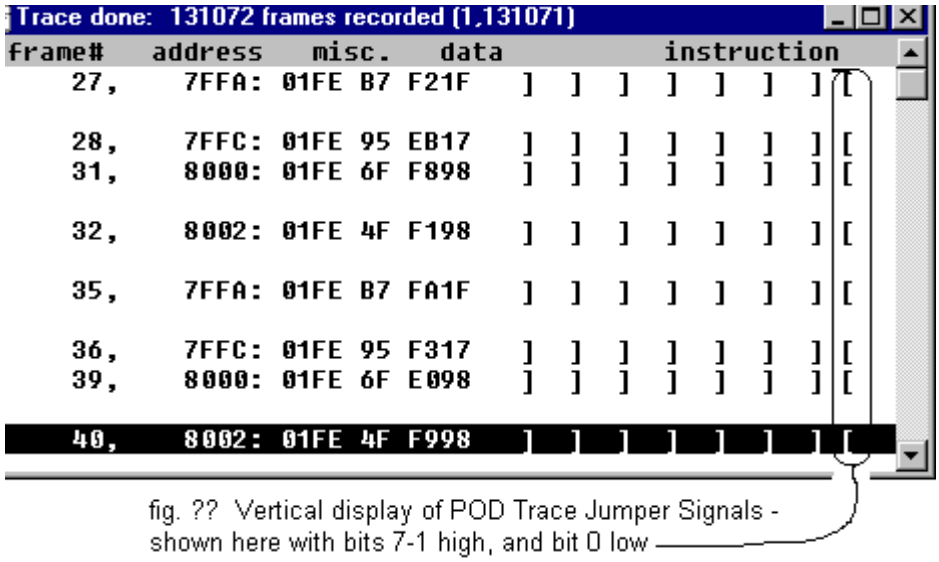


Figure 37: Vertical logic analyzer type display of digital I/O signals connected to Pod JP15

Note: Vertical display of POD Trace Jumper Signals are shown here with bits 7-1 high, and bit 0 low.

Show time stamp

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

Benchmarking Using Time stamp

When you have captured a trace and are looking at the time stamp information, keep in mind that the time stamp reflects fetch activity. Therefore, do not try to look at the time stamp for an individual instruction to determine the execution time. For example, if you want to know how long it will take to execute a multiply instruction, type in 10 multiply instructions in the **Program** Window. Then, type in a jump instruction to the start of the multiply sequence. Executing this sequence and looking at the trace result will let you determine the execution time by comparing the time the first byte of a multiply instruction was fetched to the time the first byte of the subsequent multiply instruction was fetched.

Relative time stamp

The time stamp is a 40-bit integer which is large enough to uniquely number all clock cycles in a 15.27-hour period running at 20.0 MHz. The default display mode for the time stamp 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.

In Figure 38 the JMP 8000 instruction takes 650 ns, 2 NOPs 200 ns, in external memory and 150 ns in internal code memory (20 MHz clock).

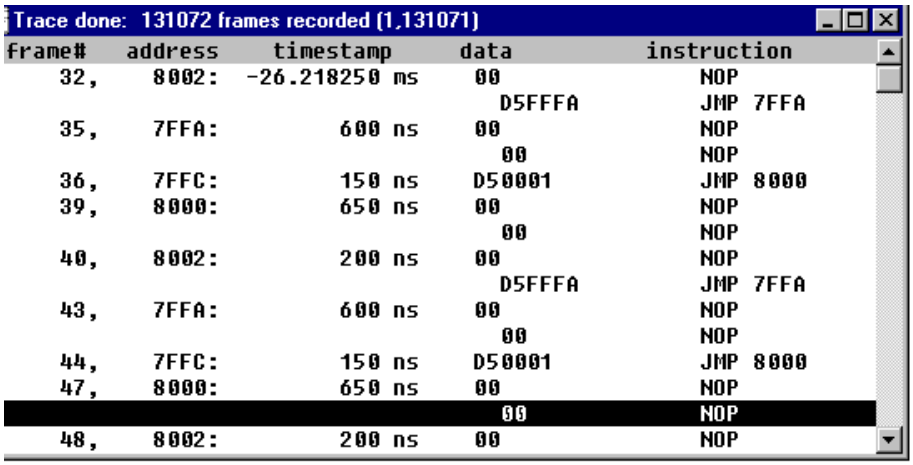


Figure 38: Example of relative time stamp displayed in seconds

Convert cycles to time

The actual time stamp in the trace record is a count of clock cycles. When this menu item is checked, the time stamp is displayed in seconds (or fractions thereof) as shown in Figure 38. 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 time stamps will be incorrect also. When unchecked, the **Trace** window displays the time stamp in clock cycles.

Synchronize Program window (to C source and Program Disassembly)

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. This would normally always be enabled.

Trace setup ..

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

Toggle trace (stop/trace)

If the trace board is recording cycles, this menu item will turn off cycle recording. Conversely, selecting this menu item before the trace board has started or after a trigger has occurred will turn on recording, just like clicking on the **Trace** button. The F10 key also toggles between **Stop** and **Trace**.

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 execute normally, however you will not see Shadow RAM update or Trace data.

Address

The **I/O Address** field is the field that identifies the start of the trace board I/O addresses. This is ignored with IETR traces.

Trace Memory

This box, shown in Figure 39, displays the number of records that can be stored in the trace board. This field is set automatically whenever the emulator and trace board are reset.

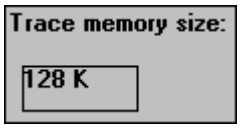


Figure 39: Size of Trace Memory

Trigger Memory/Trigger Memory Mapping

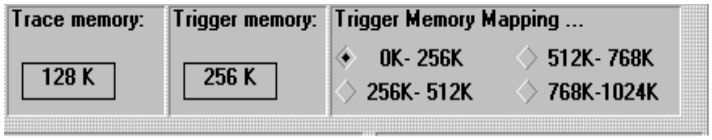


Figure 40: Trace and Code Coverage Memory Mapping

The trace memory must be mapped to cover the code and date address range of interest not only for tracing but the memory coverage function as well. Generally the default 0-256KB will suffice. Very large Programs could take advantage of the IETR 512 trace, which covers memory addresses in 1MB increments.

Triggers

Please refer to Figure 41 for the following discussion.

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 frame, bus cycles or crystal clocks 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 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 **Yes, 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 condition must be met before the trigger occurs. If a trigger condition is set for an opcode execution at address 400 and the **Last trigger repeat count** is set to 10, the first 9 fetches from address 400 will be counted and the trigger will occur when that opcode executes for the 10th time.

This count may be as high as 256.

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 Figure 44 and Figure 45 for information about setting and changing the trigger conditions.

Figure 41 shows that trigger 1 has one condition and it will be met by executing function check_timer. Then after 1,000 more bus samples, the trace capture stops. The prior 130,000 samples remain.

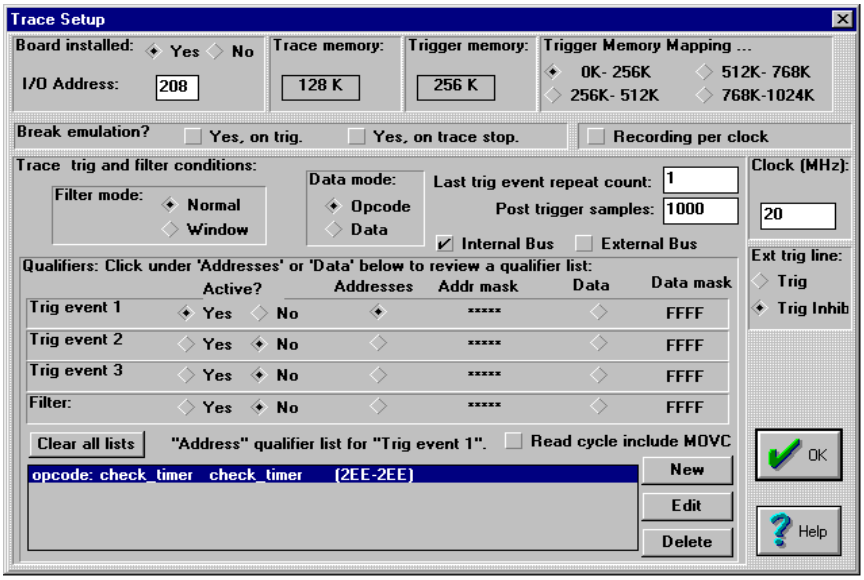


Figure 41: Trigger on anything

With **Filter mode: Normal**, bus cycles are recorded until a trigger stops the recording. Recording bus cycles is a separate activity from deciding to trigger or not, and so it has a separate set of conditions. The **Filter:** field can contain up to 2000 statements that are logically OR'ed to decide whether to record the bus cycle or not. In Figure 42, the trace board will record everything from the start of main to MAIN + 100H, the opcode fetches between line 93 and 100 from the module TIME, and only the data bus cycles (read and write cycles) to and from address timer.sec (14CH)

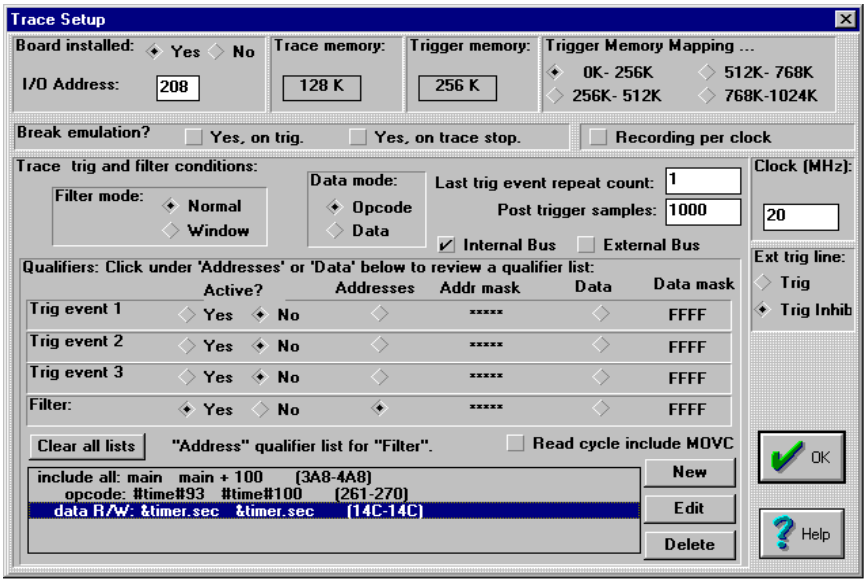


Figure 42: Selective Recording

Note: Trigger events are sequential: Trig 1, then Trig 2, then Trig 3. Trig 2 (or Trig 3), by itself, will never occur without Trig 1 (or Trig 1 and Trig 2) being active.

Filter Mode: Window

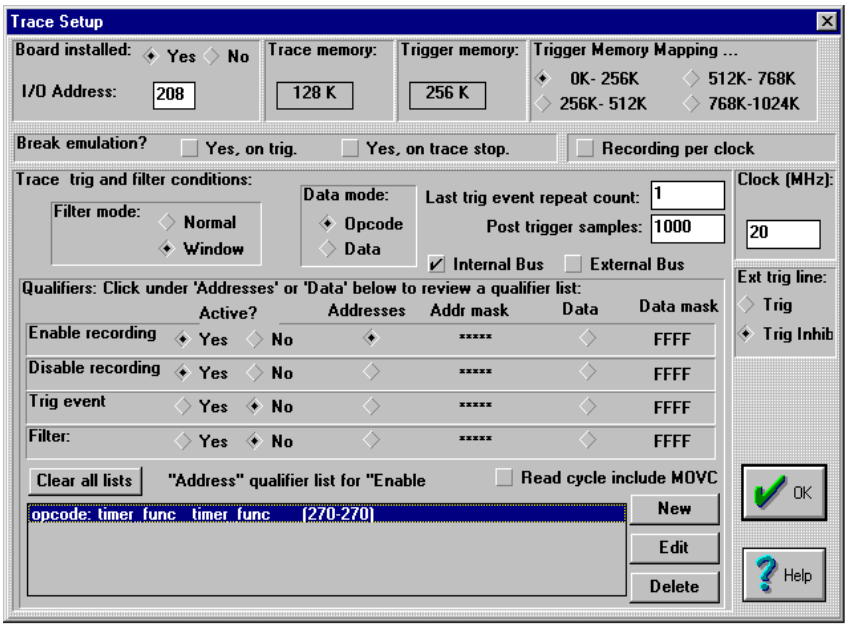


Figure 43: Filter Mode Window

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 can further qualify captured cycles.

Editing the Trigger Conditions

To set up the trigger conditions, click on one of the trigger addresses and select **New** or **Edit**. This will open an **Address qualifier** window like the one in Figure 44. 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 44 shows how to trigger on C source line 61 in module time.c by putting that address in both the **Start:** and **End:** range fields.

Note: When in data mode, care must be taken to specify the address of a structure or array member by using the C character &, e.g., & timer.sec. Single variables, e.g., ticks, are generally interpreted correctly according to context. The expression evaluator will slow the resultant hex address in brackets, e.g., & timer.sec [14C-14C].

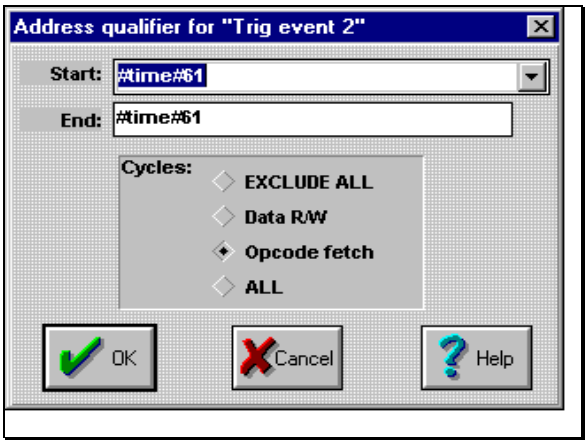


Figure 44: Data mode = Opcode

By default the **Data** mode field has the **Opcode** box marked. Marking the **Data** box gives you the options shown in Figure 45.

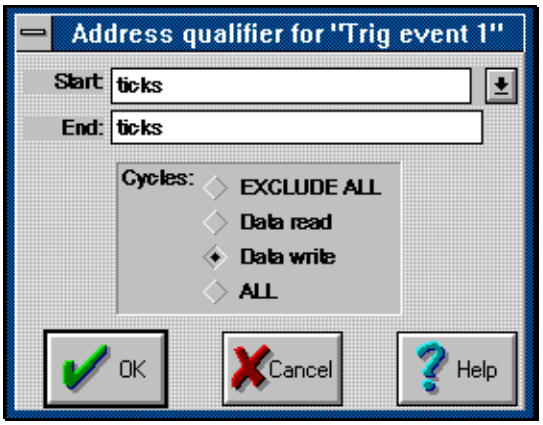


Figure 45: Data mode = Data

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.

We also want to qualify the value on the data bus. To do this, click on the **Data Diamond** and enter low and high values of the variable in question. If we wanted to trigger on a write to ticks when it reached 50H, we would use 50 twice. The data mask may remain at FFFF in this case, because ticks is of type INT and occupies an entire word. Character variables, however, will require masking either the lower or upper byte with 00FF (ignore upper byte), or FF00 (ignore lower byte). A write of 12H to location 6201H would display as “6200 12xx—w2” in the trace display, where xx could be any value. A correct data mask would then be FF00, such that the mask AND-ed with 12xx would result in a trigger on data value of 12H.

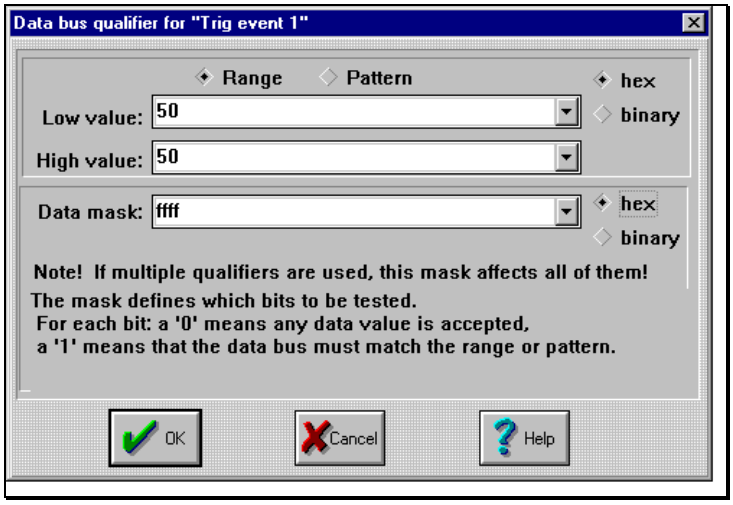


Figure 46: Data Bus Qualifier Entry

Read Cycle includes MOVC

MOVC operands may be treated as data-read cycles by checking this option in the **Trace Setup** screen. Figure 47 illustrates triggering on a word read from code memory location 7FFA. The MOVC can trigger from any code-memory address, as this bus is always visible with the XA bondout chip.

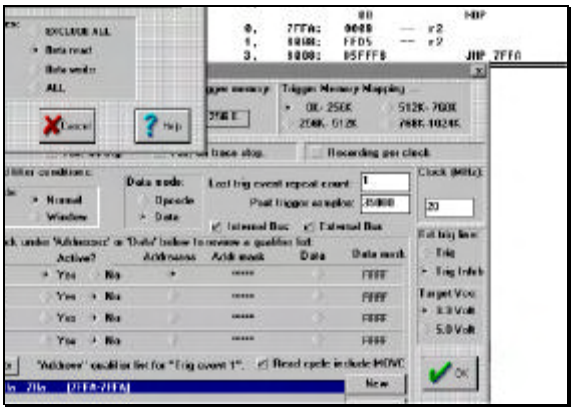


Figure 47: Triggering on MOVC read cycle with IETR

See Figure 48 for the following discussion.

Break Emulation? Box

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

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

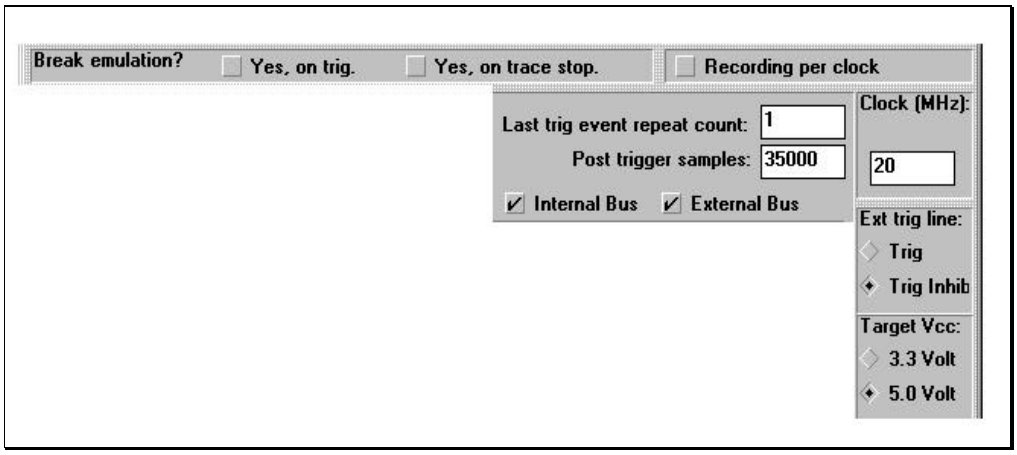


Figure 48: Trace Setup Options for On-the-fly Breakpoints and Sampling Interval

Recording per clock

This IETR can record either on every bus cycle (default), or on every clock cycle, by checking the **Recording per clock box**. This finer resolution may be useful to hardware designers, however it will reduce the number of unique cycles due to the extra code read cycle duration (see BTRL register).

Last trig event repeat count

The last trig event repeat count defaults to 1. It refers to the number of times the last active trigger condition occurs before starting the post trigger sample countdown. If you had only one trigger and the repeat count is 5, then on the fifth trigger match the trace would stop **Post Trigger samples** later.

Note: If you defined two sequential triggers, Trig1 then Trig2, the logic would be Trig = trig1, then (trig2 Repeat Count Times.) The repeat count applies only to the last ACTIVE trig condition, be it 1, 2, or 3.

Post trigger samples

This number decrements by every clock or bus cycle to zero after a valid trigger, then the trace stops capturing data (or breaks, if **Break emulation? Yes** was selected).

Internal bus/External bus

The **Post trigger samples** counted may be only Internal Code bus activity, External bus activity, or both. This is determined by checking **Internal bus, External bus**, or both boxes. If all code executes from external FLASH or EPROM, check only **External bus**. Similarly, when running in single-chip mode, check **Internal bus**. If your design has both internal code and an external bus, you would want to check both.

External trig line

This refers to the /Trigger_IN or J4 SMB connector at the edge of the IETR board. This signal is active LOW. Checking **Trig Inhib** prevents valid triggers from triggering when a LOW is presented by your external hardware on J4. This should be the default check box during most of the debug session, i.e., you normally want triggers to take effect without requiring an active LOW on trig_in. Trig = Trig_in AND (trig1, then trig2, then trig3).

Trig, on the other hand, does just the opposite. This requires an active LOW to trigger, i.e., Trig = /Trig_in AND (trig1, then trig2, then trig3).

Target VCC

If running a low-voltage target, select 3.3 volt versus the default 5.5-volt DC.

Chapter 5: Pod Boards

Features Common to All Pods

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

How It Works

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

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. With crystal jumped to your target and data memory mapped to target, those resources must be active if the POD is to function.

Indicator Lights

The pod boards contain three lights. They are labeled READY, RESET, and RUN. The red RESET light will only be lit when the emulator resets the controller. The green RUN light will be lit whenever the controller is executing user code (as opposed to monitor code). The amber READY light indicates the status of the WAIT line during external data reads and writes.

Trace Input Pins

Next to the indicator lights and the test point is an array of 8 pins labeled TRACE. These pins may be connected to any logic signal and will record the state of that signal with every trace record. Pins 0 through 3 are sampled on the falling edge of ALE, with the address. Pins 4 through 7 are sampled on the rising edge of the RD/WR strobes, with the data. For more information about displaying these bits and Trig-In/Trig-Out, please see pages 62 and 68.

Duplicate Resources

The pod board has many resources and your target may also have the same resources. If the same resource appears on both the target and the pod board, there may be a conflict that will prevent correct emulation. The only way to avoid this conflict is to remove or disable either the target or the pod resource for all the resources that appear on both.

Running the pod standalone permits C source debugging but without target I/O. However, the real value of an ICE is to run your development/target board transparently, just as if the actual XA chip was in its socket. For this you will need to change the crystal jumpers from POD to TARGET, remove the PWR jumper (target provides power to the BONDOUT chip), and plug into the target board XA socket.

Note: PODS support 3.3 VDC by removing the PWR jumper to use low-voltage target power.

The POD-51XA/G3 also includes a MAX232 chip that supports both UART ports. Should you desire to use the on-pod MAX232, insert one or both RXD jumpers (JP13 and JP14) and connect your terminal directly to the J1 or J2 on-pod UART port connections.

EMUL51XA-PC uses a special "bondout" controller to emulate the P51XAG3. This special chip has extra pins that give the emulator extra features. The bondout controller can map memory, halt execution, set breakpoints, etc. This is why your Program must execute in the controller on the pod and not in the controller on your target board.

In summary:

RESOURCE:	WHAT TO DO WHEN THE TARGET HAS IT:
RAM	Map the RAM to the Target (see Software Chapter)
Crystal	Move JP1 and JP2 to “TARGET” side of header
Serial Port(s)	Do not use J1 or J2; remove RXD jumpers J13 or J14.
Power Supply	Remove the jumper from the PWR header

The black wire with the microclip 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.

There is no collision of RAM or crystal if you use the POD resources. If you use the POD crystal, you should replace it with the exact frequency used by the target.

Chapter 6: POD-51XA/G3/I

Introduction

This pod board contains a Philip's P51XAG3 bondout microcontroller chip (suitable for emulating the Philip's 80C51XA), a 16/20/25/30 MHz crystal, 128 or 512 kilobytes emulation RAM for instructions and 128 or 512 kilobytes for data, circuits for driving the bus cable, and two large FPGA chips. Low-voltage 3.3 VDC is supported by using target power (remove JP16 PWR jumper).

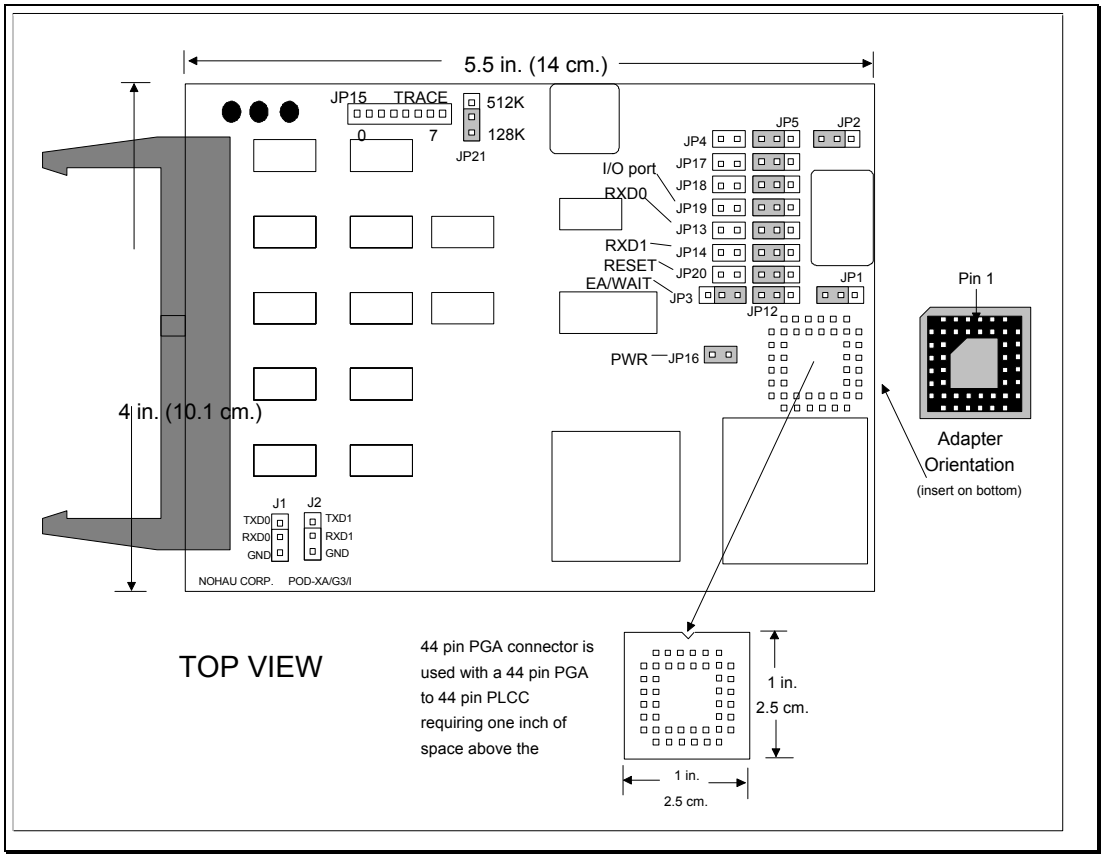


Figure 49: POD-51XA/G3/I

You must configure the software to match the hardware jumper configuration for data bus width and number of address lines.

If you use 16-bit data bus, you must have 20 bits of address. Again, the software setup must match this setting.

Dimensions

The pod board itself is 5.5 inches by 4 inches (14 cm. by 10.1 cm). The pod requires one inch (2.5 cm) of space above the target.

The location and dimensions of the 44-pin PGA connector is shown in ": POD-51XA/G3/I". This 44-pin PGA connector is attached to a 44-pin PGA to PLCC adapter. The dimensions of the 44-pin PGA to PLCC adapter are shown in Figure 50.

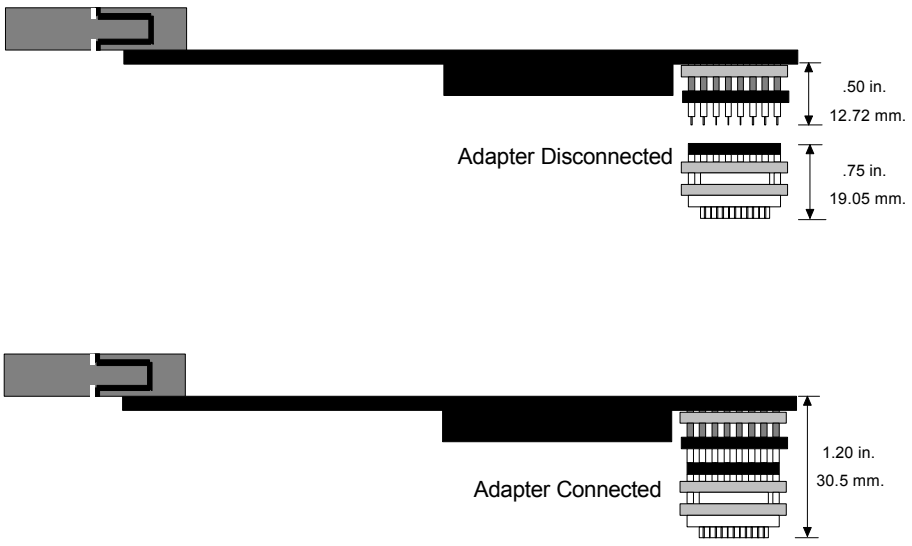


Figure 50: Adapter Dimensions

Emulation Memory

The microcontroller can directly address 128K bytes of code memory and 128K bytes of data memory. If target designs require larger memory size, a pod with 512K bytes of code memory and 512K bytes of data memory is available. Email technical suppor for information about ordering a 512K-byte pod at support@icetech.com.

Headers

In Figure 49, all the headers are shown with their jumpers in the factory-default positions. When shipped from the factory, all jumpers are in place for stand-alone operation (without a target) running code from internal code memory after reset, and 20 bits of address and 16 bits of data. When you connect the pod to a target, be sure to examine all jumpers and make sure that they are all correctly placed.

Clock Headers (JP1 and JP2)

These two headers 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 oscillator on the pod. It is important to use a 50 percent duty-cycle crystal/oscillator.

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

PWR Header (JP16)

Remove this jumper when the target has its own power supply. With this jumper in place, the target may get 5.0 VDC from the pod as long as the current requirement is less than 0.5 amps. Higher currents will cause a significant voltage drop along the current path and the pod may also be damaged. Remove this jumper for 3.3 VDC low-voltage operation.

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

RXD Headers (JP13 and JP14)

If your target outputs debugging information on the serial port, you may wish to connect an RS232 device like a terminal or a PC. This pod includes a MAX232 chip to convert the signal levels from RS232 to TTL levels. The MAX232 chip will drive the serial port input pin on the controller if you place a jumper on either RXD header. To keep the MAX232 chip from driving the serial input pin on the controller, remove the jumper on the RXD header.

The labels TXD1/RXD1 at J1are for serial port 0. The labels TXD2/RXD2 at J2 are for serial port 1.

Note: The Rev A & Rev B pods do not bring TXD1/RXD1 to J2.

Trace (JP15)

These eight pins can be used to monitor any eight logic signals on your target board. They are displayed in the **Trace** menu as TR0..TR7 where TR0 is closest to the JP15 label and TR7 is closest to the RAM Select jumper (JP21).

Reset Header (JP20)

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

EA/WAIT Header (JP3)

This header has two jumper positions: POD and TARGET. When the pod operates in stand-alone mode (without a target), the jumper should be set in POD position. The pod will provide the EA/WAIT signal to the on-pod controller. When the pod is connected to the target, the jumper should be set in the TARGET position. The target EA/WAIT signal passes through this header.

BUSWIDTH Header (JP4)

When this pod operates in stand-alone mode (without a target), using a jumper on this header will make the pod controller run with 8 bits-wide data bus. Removing this jumper will make the pod controller run with 16 bits-wide data bus. When the pod is connected to a target, this header should be removed. The target BUSW signal will pass to the pod controller.

Note: This pod does not support user Programs that can override the buswidth setting by writing to the Bus Configuration Register (BCR). The buswidth is determined by the value of the BUSW pin when Reset is released.

I/O PORT Header (JP19)

If all pins on P0, P1 and P3 are used as I/O, place a jumper on JP19. Otherwise, remove this jumper. This helps the pod to recognize the signals on P3.6 and P3.7 as I/O, instead of WR, RD.

12/16-BIT and 12-BIT Headers (JP17 and JP18)

These two headers determine how many address lines the pod controller is using.

12/16-BIT	12-BIT	NUMBER OF ADDRESS LINES
ON	ON	12
ON	OFF	16
OFF	OFF	20

Note: This pod does not support user Programs that can override the setting of the number of address lines by writing to the Bus Configuration Register (BCR).

A12-A19 Headers (JP5-JP12)

By setting the 12/16BIT and 12BIT headers, the number of address lines is determined. The next step is to set the A12-A19 headers which each have two positions: P2.x and GND. Set these headers according to the following table:

NO. OF ADDRESS LINES	A12	A13	A14	A15	A16	A17	A18	A19
12	GND	GND	GND	GND	GND	GND	GND	GND
16	P2.0	P2.1	P2.2	P2.3	GND	GND	GND	GND
20	P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7

RAM Select Header (JP21)

This header has two jumper positions: 128K and 512K. When the pod has 128K code memory and 128K data memory, set this jumper to the 128K position. When the pod has 512K code memory and 51K data memory, set this jumper to the 512K position.

Note: This jumper is set when the pod is manufactured and should not need to be changed by the user.

Features and Limitations

The POD-51XA/G3/I is designed to be used for internal mode, i.e., the /EA is HIGH during reset.

The emulator will use six bytes of stack space in large memory model (four bytes small model), so add this to your stack size calculation to avoid a stack overflow exception at 0080H.

Emulation Memory

256K pod: 128K code and 128K data memory

1M pod: 512K code and 512K data memory

Software breakpoints

Wherever there is write-able RAM.

Hardware breakpoints

1) All code addresses, one instruction skid.

256K pod : 128K hardware breakpoints

1M pod : 128K hardware breakpoints

2) External data read / write addresses with 16 byte resolution

Operation frequency

1) 0.5M to 25M without wait states in 16 bit mode, i.e. any values for CR1, CR0, CRA1, CRA0 in BTRL, and any values for DW1, DW0, DWA1, DWA0, DR1, DR0, DRA1, DRA0 in BTRH.

2) 25M to 30M with one wait state in 16 bit mode, i.e. CR1, CR0 equal 00, or CRA1, CRA0 equal 00 in BTRL, or DW1, DW0 equal 00, or DWA1, DWA0 equal 00, or DR1, DR0 equal 00, or DRA1, DRA0 equal 00 in BTRH are not supported (see below).

3) 0.5M to 30M with one wait state in 8-bit mode, i.e. CR1, CR0 equal 00, or CRA1, CRA0 equal 00 in BTRL, or DW1, DW0 equal 00, or DWA1, DWA0

equal 00, or DR1, DR0 equal 00, or DRA1, DRA0 equal b00 in BTRH are not supported (see below).

External Bus Signal Timing Configuration

	CR1,CR0	CRA1, CRA0	DW1,DW0	DWA1, DWA0	DR1,DR0	DRA1, DRA0
00	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
01	Supported	Supported	Supported	Supported	Supported	Supported
10	Supported	Supported	Supported	Supported	Supported	Supported
11	Supported	Supported	Supported	Supported	Supported	Supported

Mapping

Both code and data with 16 bytes resolution.

For both 256K and 1M pod: 1M mapping capability.

Trace

1) Can trigger/break on code fetch/code read (MOVC)/address/data, external data read/write/address/data.

2) You must use the trace if you want to break at external access (external data read/write address and/or read/write data).

Shadow Memory

Supports speeds up to 30MHz (requires IETR).

Speed Limit

POD	FREQ	BUS CYCLE	CODE FETCH/DATA READ/DATA WRITE		(16 bit)
		5 CLK W/ALE, 4 CLK W/O ALE	4 CLK W/ALE, 3 CLK W/O ALE	3 CLK W/ ALE, 2 CLK W/O ALE	2 CLK W/ ALE, 1 CLK W/O ALE
POD-51XAG3-256/I-16	16MHz	WORKING	WORKING	WORKING	WORKING
POD-51XAG3-256/I-20	20MHz	WORKING	WORKING	WORKING	WORKING
POD-51XAG3-256/I-25	25MHz	WORKING	WORKING	WORKING	WORKING
POD-51XAG3-256/I-30	30MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-1M/I16	16MHz	WORKING	WORKING	WORKING	WORKING
POD-51XAG3-1M/I20	20MHz	WORKING	WORKING	WORKING	WORKING
POD-51XAG3-1M/I25	25MHz	WORKING	WORKING	WORKING	WORKING
POD-51XAG3-1M/I30	30MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-256/I-16	16MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-256/I-20	20MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-256/I-25	25MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-256/I-30	30MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-1M/I16	16MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-1M/I20	20MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-1M/I25	25MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-1M/I30	30MHz	WORKING	WORKING	WORKING	NOT WORKING

Chapter 7: POD-51XA/G3/E

Introduction

This pod board contains a Philip's P51XAG3 bondout microcontroller chip (suitable for emulating the Philip's 80C51XA), a 16/20/25/30 MHz crystal, 256 kilobytes or 1 MB emulation RAM, circuits for driving the bus cable, and two large FPGA chips. Low-voltage 3.3 VDC is supported by using target power (remove JP16 PWR jumper.).

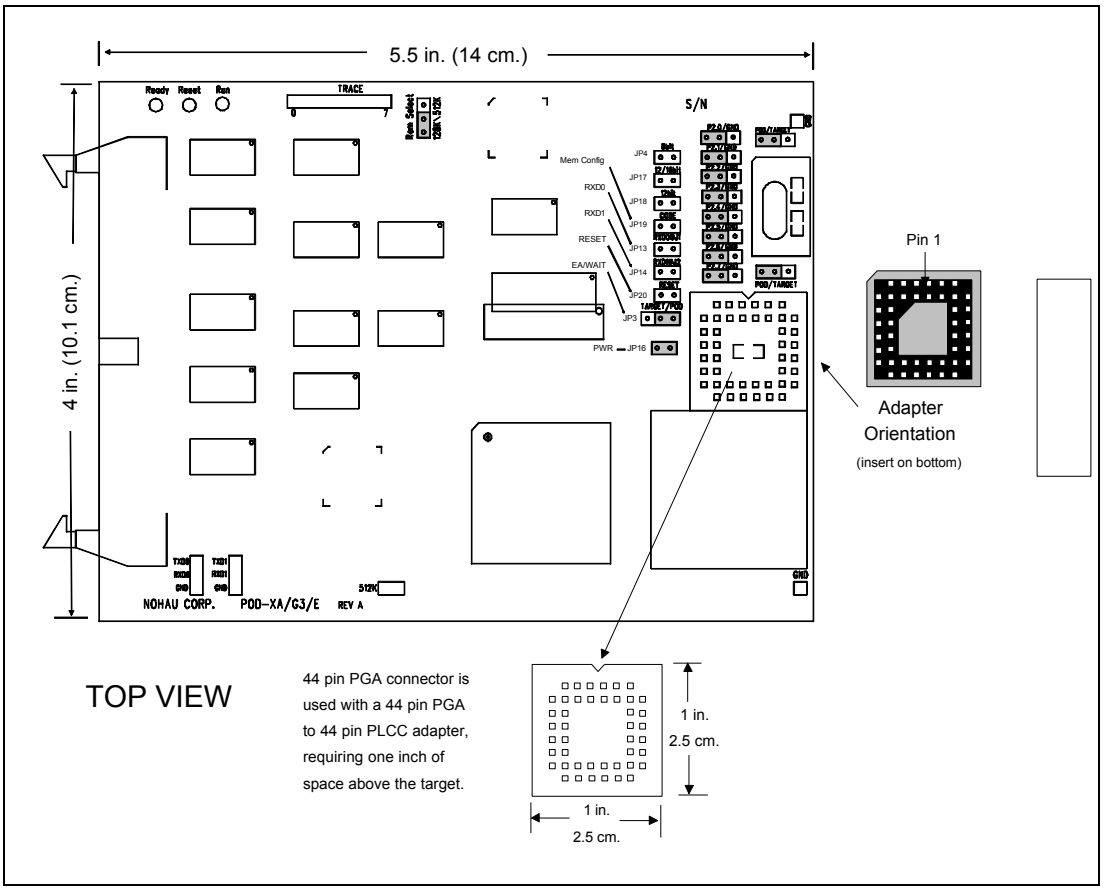


Figure 51: POD-51XA/G3/E

You must configure the software to match the hardware jumper configuration for data buswidth and number of address lines.

If you use 16-bit data bus, you must have twenty bits of address. Again, the software setup must match this setting.

Dimensions

The pod board itself is 5.5 inches by 4 inches (14 cm. by 10.1 cm). The pod requires one inch (2.5 cm) of space above the target.

The location and dimensions of the 44-pin PGA connector is shown in Figure 48. This 44-in PGA connector is attached to a 44-pin PGA to PLCC adapter. The dimensions of the 44 pin PGA to PLCC adapter are shown in Figure 52.

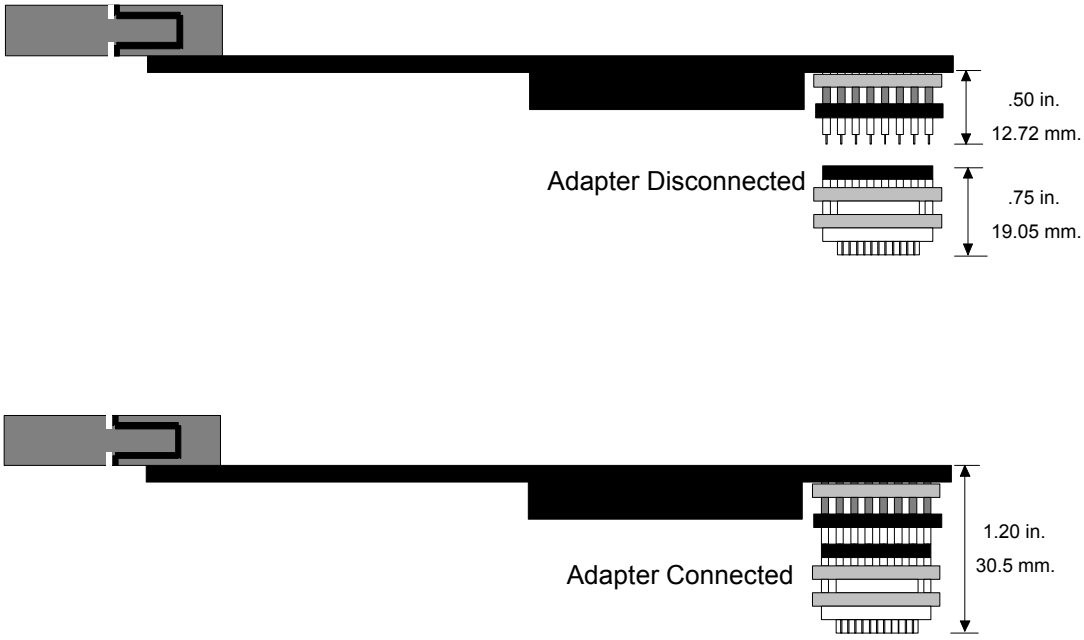


Figure 52: Adapter Dimensions

Emulation Memory

The microcontroller can directly address 128K bytes of code memory and 128K bytes of data memory. If target designs require larger memory size, a pod with 512K bytes of code memory and 512K bytes of data memory is available. Email support at suppor@icetech.com for information about ordering a 512K-byte pod.

Note: The memory can also be configured as 256K bytes of code memory without data memory or 1M byte of code without data memory (see "Memory Configuration Header (JP19)" on page 98).

Headers

In Figure 51, all the headers are shown with their jumpers in the factory default positions. When shipped from the factory, all jumpers are in place for stand-alone operation (without a target) running code from external code memory after reset, and 20 bits of address and 16 bits of data. When you connect the pod to a target, be sure to examine all jumpers and make sure that they are all correctly placed.

Clock Headers (JP1 and JP2)

These two headers 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 bond-out controller on the pod will use the oscillator on the pod. It is important to use a 50 percent duty-cycle crystal/oscillator.

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

PWR Header (JP16)

Remove this jumper when the target has its own power supply. With this jumper in place, the target may get 5.0 VDC from the pod as long as the current requirement is less than 0.5 amps. Higher currents will cause a significant voltage drop along the current path and the pod may also be damaged. Remove this jumper for 3.3 VDC low-voltage operation.

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

RXD Headers (JP13 and JP14)

If your target outputs debugging information on the serial port, you may wish to connect an RS232 device like a terminal or a PC. This pod includes a MAX232 chip to convert the signal levels from RS232 to TTL levels. The MAX232 chip will drive the serial port input pin on the controller if you place a jumper on either RXD header. To keep the MAX232 chip from driving the serial input pin on the controller, remove the jumper on the RXD header.

Trace Header (JP15)

These eight pins can be used to monitor any eight logic signals on your target board. They are displayed in the Trace menu as TR0..Tr7 where TR0 is closest to the JP15 label and TR7 is closest to the RAM Select jumper (JP21).

Reset Header (JP20)

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

EA/WAIT Header (JP3)

This header has two jumper positions: POD and TARGET. When the pod operates in stand-alone mode (without a target), the jumper should be set in POD position. The pod will provide the EA/WAIT signal to the on-pod controller. When the pod is connected to the target, the jumper should be set in the TARGET position. The target EA/WAIT signal passes through this header.

BUSWIDTH Header (JP4)

When this pod operates in stand-alone mode (without a target), using a jumper on this header will make the pod controller run with 8 bits wide data bus. Removing this jumper will make the pod controller run with 16 bits wide data bus. When the pod is connected to a target, this header should be removed. The target BUSW signal will pass to the pod controller.

Note: This pod does not support user programs that can override the bus width setting by writing to the Bus Configuration Register (BCR). The bus width is determined by the value of the BUSW pin when Reset is released.

Memory Configuration Header (JP19)

If this jumper is removed, the pod memory is configured as 128K code memory and 128K data memory. By placing a jumper on JP19, the pod memory is configured as 256K code memory without data memory. Or, if you have the 512K pod, it can be configured as a 1M byte code and no external data memory.

12/16-BIT and 12-BIT Headers (JP17 and JP18)

These two headers determine how many address lines the pod controller is using.

12/16-BIT	12-BIT	NUMBER OF ADDRESS LINES
ON	ON	12
ON	OFF	16
OFF	OFF	20

Note: This pod does not support user Programs than can override the setting of the number of address lines by writing to the Bus Configuration Register (BCR).

A12-A19 Headers (JP5-JP12)

By setting the 12/16-BIT and 12-BIT headers, the number of address lines is determined. The next step is to set the A12-A19 headers which each have two positions: P2.x and GND. Set these headers according to the following table:

NO. OF ADDRESS LINES	A12	A13	A14	A15	A16	A17	A18	A19
12	GND	GND	GND	GND	GND	GND	GND	GND
16	P2.0	P2.1	P2.2	P2.3	GND	GND	GND	GND
20	P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7

RAM Select Headers (JP21, JP22)

Header JP21 has two jumper positions: 128K and 512K. When using a 256K pod, set this jumper to the 128K position and remove the jumper on JP22. When using a 1M pod, set this jumper to the 512K position and place a jumper on JP22.

Note: This jumper is set when the pod is manufactured and should not need to be changed by the user.

Features and Limitations

The POD-51XA/G3/E is designed to be used for ROM-less operation (external mode), i.e., the /EA is LOW during reset.

The emulator will use six bytes of stack space in large memory model (four bytes small model), so add this to your stack size calculation to avoid a stack overflow exception at 0080H.

Emulation Memory

256K pod: 128K code and 128K data memory or,

256K code and 0K data memory.

1M pod: 512K code and 512K data memory or,

1M code and 0K data memory.

Software breakpoints

Wherever there is write-able RAM.

Hardware breakpoints

All code address, one instruction skid.

256K pod : 256K hardware breakpoints.

1M pod : 1M hardware breakpoints.

Operation frequency

1) 0.1M to 20M without wait states in 16-bit mode, i.e. any values for CR1, CR0, CRA1, CRA0 in BTRL, and any values for DW1, DW0, DWA1, DWA0, DR1, DR0, DRA1, DRA0 in BTRH.

2) 21M to 30M with one wait state in 16-bit mode, i.e. CR1, CR0 equal 00, or CRA1, CRA0 equal 00 in BTRL, or DW1, DW0 equal 00, or DWA1, DWA0 equal 00, or DR1, DR0 equal 00, or DRA1, DRA0 equal 00 in BTRH are not supported (see chart on External Bus Signal Timing Configuration).

3) 0.1M to 30M with one wait state in 8-bit mode, i.e. CR1, CR0 equal 00, or CRA1, CRA0 equal 00 in BTRL, or DW1, DW0 equal 00, or DWA1, DWA0 equal 00, or DR1, DR0 equal 00, or DRA1, DRA0 equal 00 in BTRH are not supported (see chart on External Bus Signal Timing Configuration).

External Bus Signal Timing Configuration

	CR1,CR0	**CRA1, CRA0**	**DW1,DW0**	**DWA1, DWA0**	**DR1,DR0**	**DRA1, DRA0**
00	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
01	Supported	Supported	Supported	Supported	Supported	Supported
10	Supported	Supported	Supported	Supported	Supported	Supported
11	Supported	Supported	Supported	Supported	Supported	Supported

Mapping

Both code and data with 16 bytes resolution.

For both 256K and 1M pod: 1M mapping capability.

Trace

1) Can trigger/break on code fetch/code read (MOVC)/address/data, external data read/write/address/data.

2) You must use the trace if you want to break at external access (external data read/write address and/or read/write data).

Shadow Memory

Supports speeds up to 30MHz (requires IETR).

Wait State Input / WAITD Bit

The /EA/WAIT input is delayed by a 10ns gate.

The POD-51XA/G3/E does not support Wait State Disable, i.e the WAITD bit (bit 4) in BCR equal to 1 is not supported.

Speed Limit

POD	FREQ	BUS CYCLE	CODE FETCH/DATA READ/DATA WRITE		(16 bit)
		5 CLK W/ALE, 4 CLK W/O ALE	4 CLK W/ALE, 3 CLK W/O ALE	3 CLK W/ ALE, 2 CLK W/O ALE	2 CLK W/ ALE, 1 CLK W/O ALE
POD-51XAG3-256/E16	16MHz	WORKING	WORKING	WORKING	WORKING
POD-51XAG3-256/E20	20MHz	WORKING	WORKING	WORKING	WORKING
POD-51XAG3-256/E25	25MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-256/E30	30MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-1M/E16	16MHz	WORKING	WORKING	WORKING	WORKING
POD-51XAG3-1M/E20	20MHz	WORKING	WORKING	WORKING	WORKING
POD-51XAG3-1M/E25	25MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-1M/E30	30MHz	WORKING	WORKING	WORKING	NOT WORKING

POD	FREQ	BUS CYCLE	CODE FETCH/DATA READ/DATA WRITE		(8 bit)
		5 CLK W/ALE, 4 CLK W/O ALE	4 CLK W/ALE, 3 CLK W/O ALE	3 CLK W/ ALE, 2 CLK W/O ALE	2 CLK W/ ALE, 1 CLK W/O ALE
POD-51XAG3-256/E16	16MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-256/E20	20MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-256/E25	25MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-256/E30	30MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-1M/E16	16MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-1M/E20	20MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-1M/E25	25MHz	WORKING	WORKING	WORKING	NOT WORKING
POD-51XAG3-1M/E30	30MHz	WORKING	WORKING	WORKING	NOT WORKING

Chapter 8: POD-51XA/G3/IE

Introduction

The POD–51XA/G3/IE contains a Philips P51XAG3 bondout microcontroller chip, (suitable for emulating the Philips P51XAG3 up to 16/20/25/30 MHz) 256 Kbytes, 1 Mbyte or 2 Mbytes of emulation RAM, circuits for driving the bus cable, and CPLD chips. The pod supports low-voltage 3.3 VDC operation. You need to remove the JP15 (POD PWR) jumper and set JP16 (5 V/3 V) to the 3 V position in order for the pod to operate at 3.3 VDC.

The software configurations must match the hardware jumper configurations for the data buswidth and the number of address lines. If you use a 16-bit data bus, you must have 20 bits of address. The software setup must match this setting.

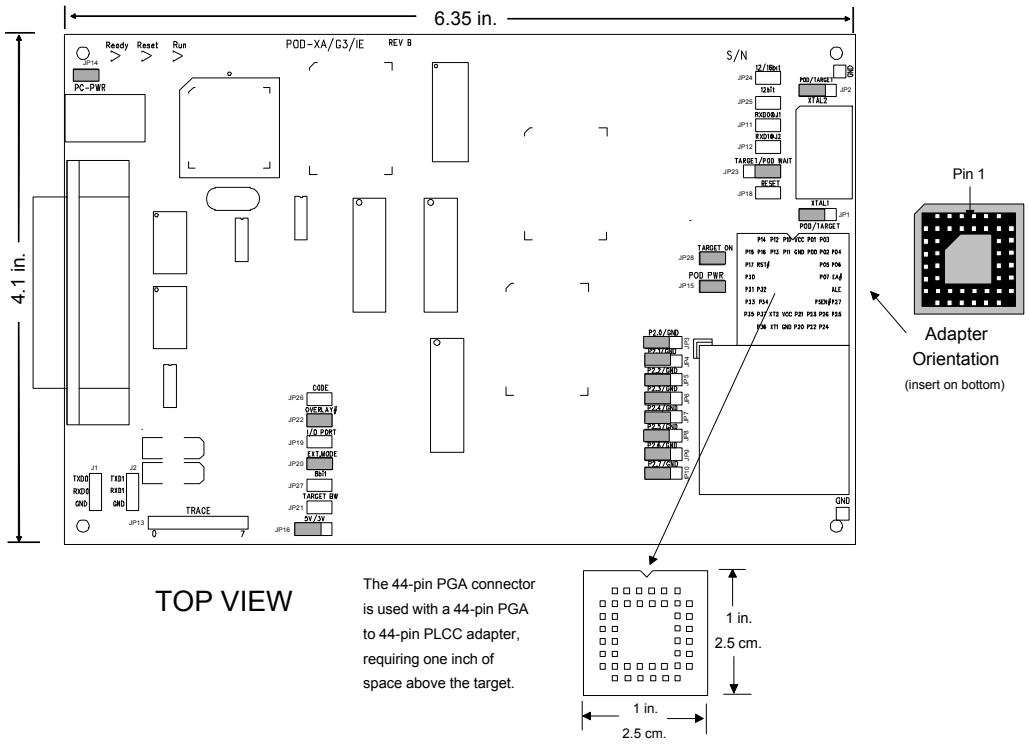


Figure 53: POD–51XA / G3 / IE

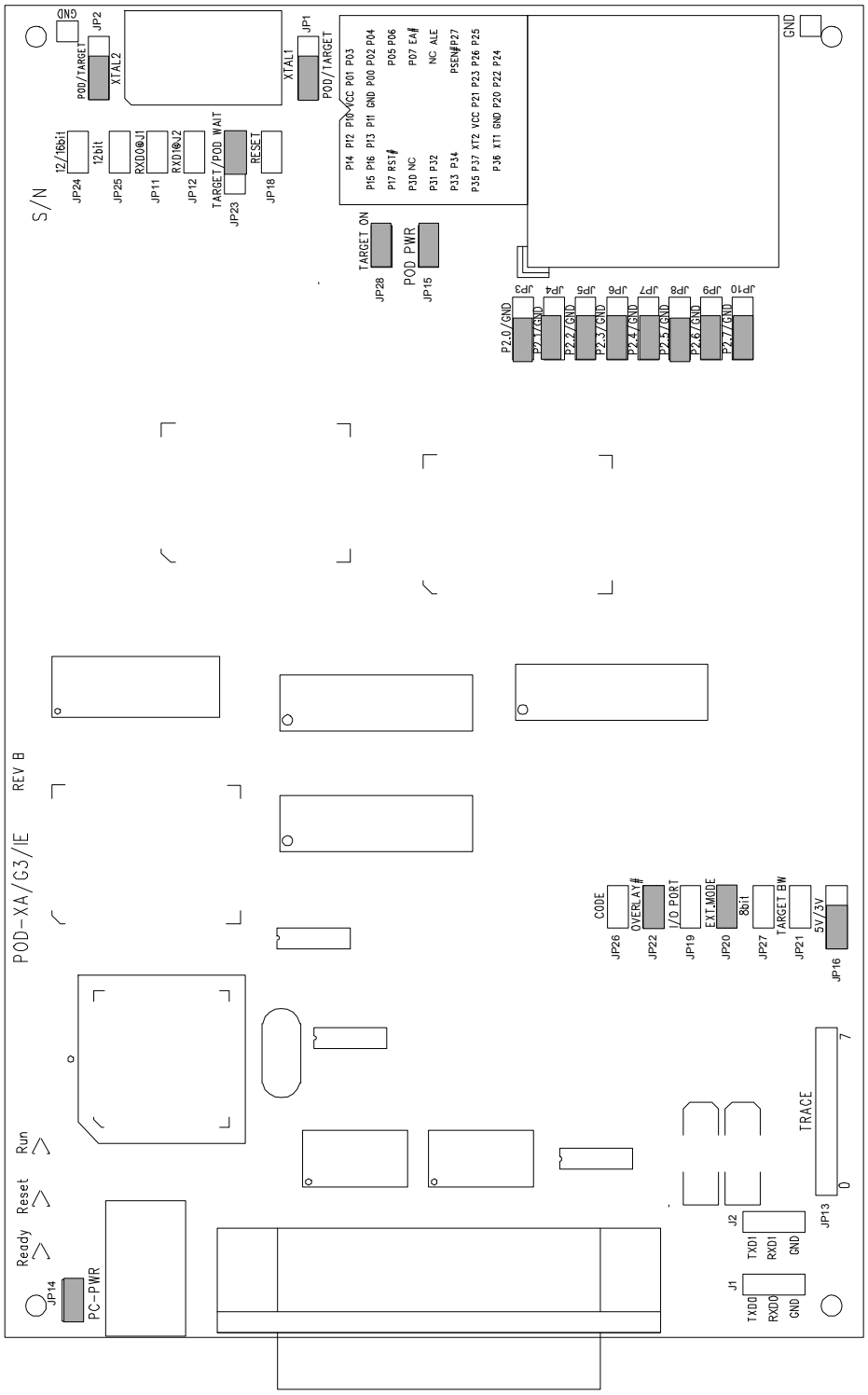


Figure 54. Enlargement of POD–51XA / G3 / IE

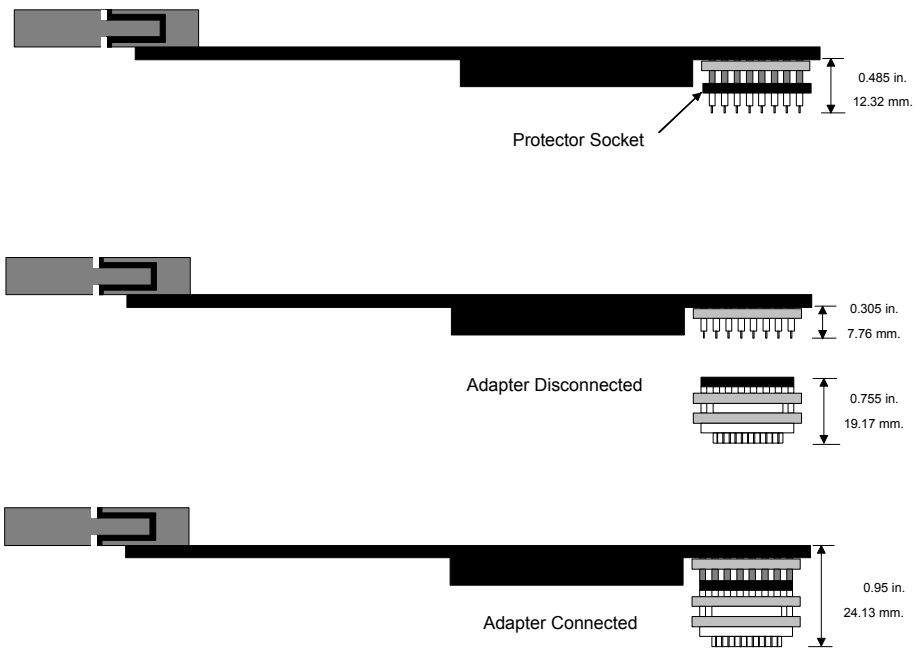


Figure 55: Adapter Dimensions

Dimensions

The pod board measures 6.35 inches by 4.1 inches and requires one inch (2.5 cm) of space above the target.

The location and dimensions of the 44-pin PGA connector is shown in Figure 53. This 44-pin PGA connector is attached to a 44-pin PGA to PLCC adapter. The dimensions of the 44-pin PGA to PLCC adapter are shown in Figure 55.

Internal / External Trace Board (IETR) Users

Figure 56, shows the items that you must remove before installing the trace board. Remove the foam, the protector socket, and the back panel from the pod. Do not remove the spacer from under the pod board.

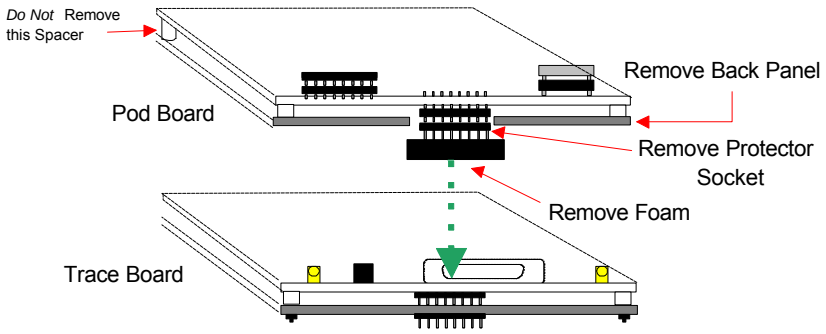


Figure 56: Internal / External Trace Board Installation

Emulation Memory

The standard emulator pod has 128 Kbytes of code and 128 Kbytes of data memory. If you need more memory, pods with 512 Kbytes and 1 Mbyte of code and data memory are available. Email sales@icetech.com for information about ordering a one- or two-megabyte pod.

Memory can also be configured as

- Kbytes of code memory without data memory (256K pod)
- 256 Kbytes of code and data overlay (256K pod)
- 512 Kbytes of code and 512 Kbytes of data memory
- 1 Mbyte of code without data memory (1-Mbyte pod)
- 1 Mbyte of code and data overlay (1-Mbyte pod)

See the “Code Header—JP26 and Overlay #Header—JP22” section on page 114 for details on the jumper settings.

Headers

Figure 54 on page 106 shows the headers with their jumpers in the default positions. When shipped from the factory, all jumpers are in place for stand-alone operation (without a target). This stand-alone operation runs code from external code memory after reset, 20 bits of address, and 16 bits of data.

When you connect this pod to a target, be sure to examine all the jumpers for correct placement. The following sections describe the correct placement for these jumpers.

Clock Headers—JP1 and JP2

These two headers each have two jumper positions: TARGET and POD. They must be moved as a pair. With both headers set in the TARGET position, the on-pod XA bondout chip receives the clock signal from the target crystal or oscillator. It is important to use a 50 percent duty-cycle oscillator.With both headers set in the POD position, the XA bondout chip uses the oscillator on the pod.

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

EXT Mode Header—JP20

If you operate the XA in Internal mode (see Figure 57) then you need to *remove* JP20. If you operate the XA in External mode (see Figure 58) then you need to install JP20. Internal mode means /EA = VCC (5.0 VCC for XAG49, 3.3 or 5.0 for G3), and code memory starts from on-chip ROM/FLASH. External mode means code memory starts from off-chip EPROM or FLASH and /EA = 0 VDC. The Pod Selection menu also follows these guidelines – INT if /EA = VCC, EXT if /EA = 0.

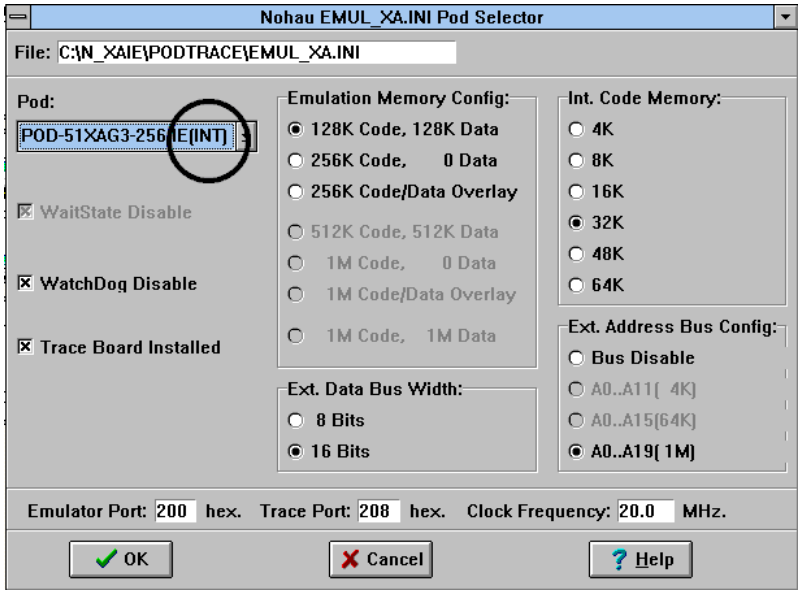


Figure 17. Selecting the Internal Mode Operation

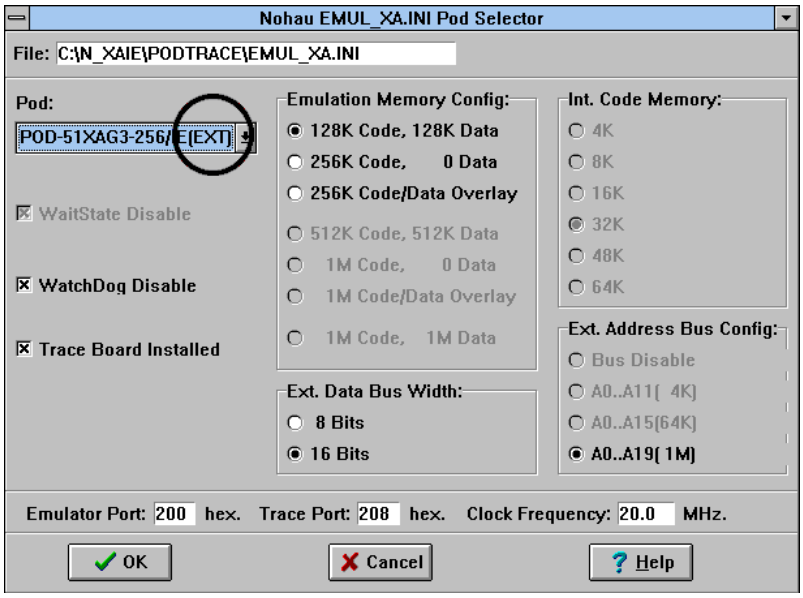


Figure 58: Selecting the External Mode Operation

PC–PWR Header—JP14

If you run the pod with the EPC interface, install JP14 and use the external power supply. When using the LC–ISA, we recommend using the external power supply and removing JP14. To run the pod with PC power, install JP14 (the external power supply is not used).

POD–PWR Header—JP15

Remove JP15 if you are using power from the target for the CPU. If JP15 is not removed in this circumstance, the target VCC is connected to 5 V from the pod. If the target requires less than 0.5 amps current, the pod can be used to power the target with JP15 installed. Higher currents cause a significant voltage drop along the current path. This drop in voltage can damage the pod. You need to remove JP15 in order for the pod to operate at 3.3 VDC supplied by the target.

Target On Header—JP28

If you connect the pod to a target that could be affected by the pod outputting 1.8 volts at the XA VCC pin and the I/O pins, remove JP28 before applying power to the target. Reinstall JP28 after applying power to the target. Similarly, remove JP28 before you turn off the target power. JP28 has this off-and-on capability to avoid voltage problems. By removing JP28, all the pins of the XA are tristated. However, after applying power to the target, you must reinstall JP28.

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

5 V and 3 V Header—JP16

If you operate the XA at 5 V, set JP16 at the 5 V position. If you operate the XA at 3.3 V, set JP16 at the 3 V position.

RXD Headers—JP11 and JP12

If your target outputs debugging information on the serial port, you might want to connect an RS232 device (a terminal or a PC). This pod includes a MAX232 chip that converts the signal levels from RS232 to TTL levels. If you place a jumper on either RXD header, the MAX232 chip drives the serial port input pin on the XA bondout chip. To keep the MAX232 chip from driving the serial input pin on the XA bondout chip, remove the jumper on the RXD header.

RS232 Headers—J1 and J2

Header J1 connects to serial port 0. Header J2 connects to serial port 1.

Trace Header—JP13

With the optional trace, these eight pins monitor any eight logic signals on your target board. The Trace menu displays these pins as TR0 – TR7. TR0 is closest to the JP13 label and TR7 is closest to the 5 V/3 V jumper, JP16.

Reset Header—JP18

Occasionally, a target contains an external device designed to reset the XA chip by pulling the RST pin low. During debugging, this reset might be inconvenient. The signal from the target RST pin passes through the RST jumper. Removing the RST jumper prevents the external device from resetting the XA bondout chip.

TARGET / POD Wait Header—JP23

This header has two jumper positions: POD and TARGET. When the pod operates in stand-alone mode (without a target), set JP23 in the POD position. The pod provides the WAIT signal to the on-pod XA bondout chip. When the pod operates with a target, setting JP23 in the TARGET position connects the XA to the target WAIT signal. The target WAIT signal passes through JP23.

I/O Port Header—JP19

JP19 enables the pod to recognize the signals on P3.6 and P3.7 as I/O signals, instead of WR and RD signals. Install JP19 if the XA operates in internal mode, and uses all pins on P0, P1, P2 and P3 as I/O. If the XA *does not* operate in internal mode, remove JP19.

Target BW Header—JP21 and 8-Bit Header—JP27

JP21 should always be off.

When stand-alone, the 8-bit jumper, JP27, enables the on-pod XA boundout chip to run with an 8-bit wide data bus. Removing JP27 enables the on-pod XA bondout chip to run with a 16-bit wide data bus.

Note: *This pod does not support user programs that can override the bus width setting by writing to the Bus Configuration Register (BCR). The bus width is determined by the value of the BUSW pin when Reset is released.*

Code Header—JP26 and Overlay #Header—JP22

Table 1. Jumper Settings for the 256K Pod

256K Pod	Code Header—JP26	Overlay #Header—JP22
128K code, 128K data	Off	On
256K code, 0K data	On	On
256K code, 256K data overlay	Off	Off

Table 2. Jumper Settings for the 1-Mbyte Pod

1-Mbyte Pod	Code Header—JP26	Overlay #Header—JP22
512K code, 512K data	Off	On
1 Mbyte code, 0 Mbyte data	On	On
1 Mbyte code, 1 Mbyte data overlay	Off	Off

Table 3. Jumper Settings for the 2-Mbyte Pod

2-Mbyte Pod	Code Header—JP26	Overlay #Header—JP22
1 Mbyte code, 1 Mbyte data	Off	Off

12 / 16-Bit and 12-Bit Headers—JP24 and JP25

JP24 and JP25 determine the number of address lines that are used by the on-pod XA boundout chip.

Table 4. JP24 and JP25 Settings

JP 25 (12-/16-Bit)	JP 24 (12-Bit)	Number of Address Lines
On	On	12
On	Off	16
Off	Off	20

Note: This pod does not support user Programs that override the number of address lines setting by writing to the Bus Configuration Register (BCR).

A12 – A19 Headers—JP3 – JP10

The A12 – A19 headers each have two positions: P2.x and GND. Set these headers according to the number of address lines that were set by the JP24 and JP 25 jumpers (see Table 5).

Table 5. A12 – A19 Headers—JP3 – JP10 Settings

Number of Address Lines	A12	A13	A14	A16	A17	A18	A19
12	GND	GND	GND	GND	GND	GND	GND
16	P2.0	P2.1	P2.2	GND	GND	GND	GND
20	P2.0	P2.1	P2.2	P2.4	P2.5	P2.6	P2.7

Features and Limitations

The emulator uses six bytes of stack space in a large memory model and four bytes of stack space in a small memory model. You need to add six (or four) bytes to your stack size calculation to avoid a stack overflow exception at 0080H.

Emulation Memory

One of the following for the 256K pod

- 128K code and 128K data memory
- 256K code and 0K data memory
- 256K code and 256K data memory overlay

One of the following for the 1-Mbyte pod

- 512K code and 512K data memory
- 1 Mbyte code and 0 Mbyte data memory
- 1 Mbyte code and 1 Mbyte data memory overlay

The 2-Mbyte pod has 1 Mbyte of code and 1 Mbyte of data memory.

Software Breakpoints

You can set software breakpoints wherever there is emulation code memory.

Hardware Breakpoints

- All code address, one instruction skid
  - ➤ 256K pod—256K hardware breakpoints
  - ➤ 1-Mbyte pod—1 Mbyte hardware breakpoints
  - ➤ 2-Mbyte pod—1 Mbyte hardware breakpoints
- External data read/write address with word resolution

Fast Break Write

Fast Break Write is available when the pod is operating in the external or internal mode.

Data / Address Bus Configurations

The configurations of an 8-bit data bus and a 12-bit address bus in external mode are not supported.

Operating Frequency

1 MHz to 25 MHz in 16-Bit Mode

WM0 must equal 1 in BTRL. Table 6 shows the external bus signal timing configurations.

Table 6. Configurations for 1 MHz to 25 MHz in 16-Bit Mode

	CR1, CR0	CRA1, CRA0	DW1, DW0	DWA1, DWA0	DR1, DR0	DRA1, DRA0
00	Supported	Supported	N/A	Not supported	N/A	Supported
01	Supported	Supported	N/A	Supported	N/A	Supported
10	Supported	Supported	N/A	Supported	N/A	Supported
11	Supported	Supported	N/A	Supported	N/A	Supported

25 MHz to 30 MHz in 16-Bit Mode

WM0 must equal 1 in BTRL. Table 7 shows the external bus signal timing configurations.

Table 7. Configurations for 25 MHz to 30 MHz in 16-Bit Mode

	CR1, CR0	CRA1, CRA0	DW1, DW0	DWA1, DWA0	DR1, DR0	DRA1, DRA0
00	Not supported	Not supported	N/A	Not supported	N/A	Not supported
01	Supported	Supported	N/A	Supported	N/A	Supported
10	Supported	Supported	N/A	Supported	N/A	Supported
11	Supported	Supported	N/A	Supported	N/A	Supported

1 MHz to 20 MHz in 8-Bit Mode

WM0 must equal 1 in BTRL. Table 8 shows the external bus signal timing configurations.

Table 8. Configurations for 1 MHz to 20 MHz in 8-Bit Mode

	CR1, CR0	CRA1, CRA0	DW1, DW0	DWA1, DWA0	DR1, DR0	DRA1, DRA0
00	Supported	Supported	Not supported	Not supported	Supported	Supported
01	Supported	Supported	Supported	Supported	Supported	Supported
10	Supported	Supported	Supported	Supported	Supported	Supported
11	Supported	Supported	Supported	Supported	Supported	Supported

20 MHz to 30 MHz in 8-Bit Mode

WM0 must equal 1 in BTRL. Table 9 shows the external bus signal timing configurations.

Table 9. Configurations for 20 MHz to 30 MHz in 8-Bit Mode

	CR1, CR0	CRA1, CRA0	DW1, DW0	DWA1, DWA0	DR1, DR0	DRA1, DRA0
00	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
01	Supported	Supported	Supported	Supported	Supported	Supported
10	Supported	Supported	Supported	Supported	Supported	Supported
11	Supported	Supported	Supported	Supported	Supported	Supported

Mapping Capabilities

The mapping capabilities map code and data with 16 bytes of resolution. The mapping capability covers the entire address range of the POD–51XA/G3/IE (1 Mbyte).

Trace

Trace can trigger/break on code fetch/code read (MOVC)/address/data, external data read/write/address/data.

Shadow Memory

Shadow memory supports speeds up to 30 MHz, and is available when you use an IETR.

Chapter 9: POD-51XA/S3/IE

Introduction

The POD–51XA/S3/IE contains a Philips P51XAS3 bondout microcontroller chip, (suitable for emulating the Philips P51XAS3 up to 16/20/25/30 MHz) 256 Kbytes, 1 Mbyte or 2 Mbytes of emulation RAM, circuits for driving the bus cable, and CPLD chips. The pod supports low-voltage 3.3 VDC operation. You need to remove the JP15 (POD PWR) jumper and set JP16 (5 V/3 V) to the 3 V position in order for the pod to operate at 3.3 VDC.

The software configurations must match the hardware jumper configurations for the data buswidth and the number of address lines. If you use a 16-bit data bus, you must have 20 or 24 bits of address. The software setup *must match* this setting.

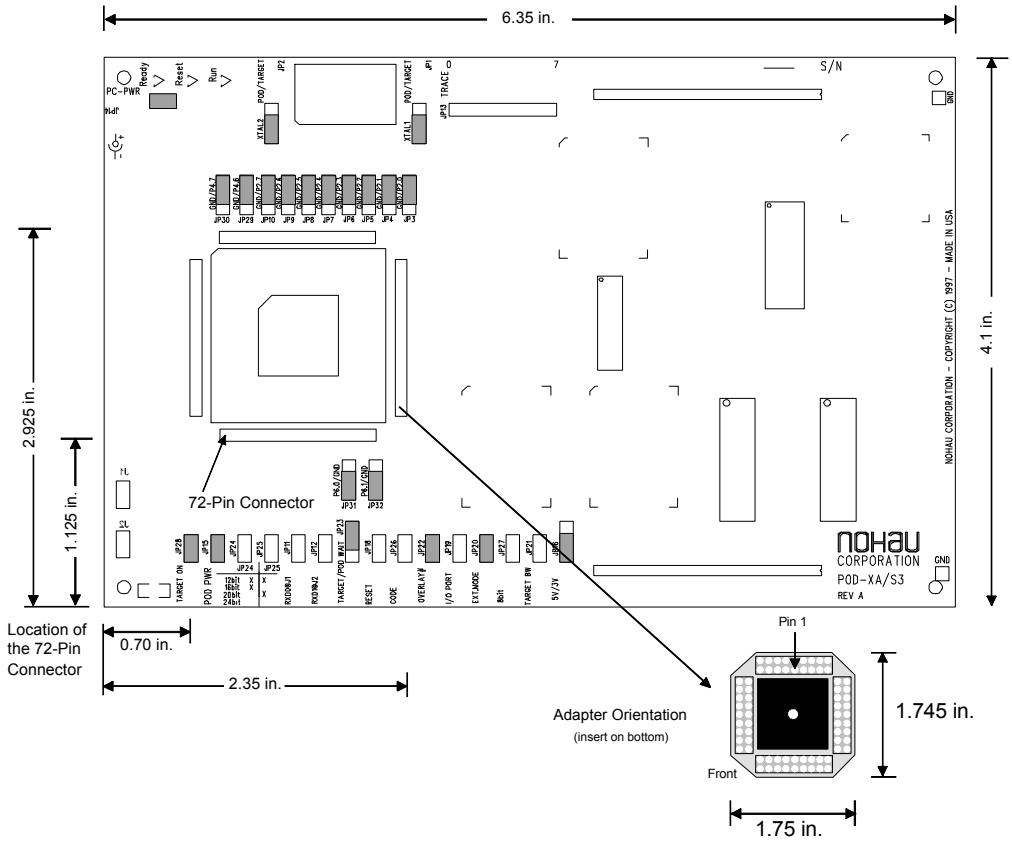


Figure 59: POD-51XA/S3/IE

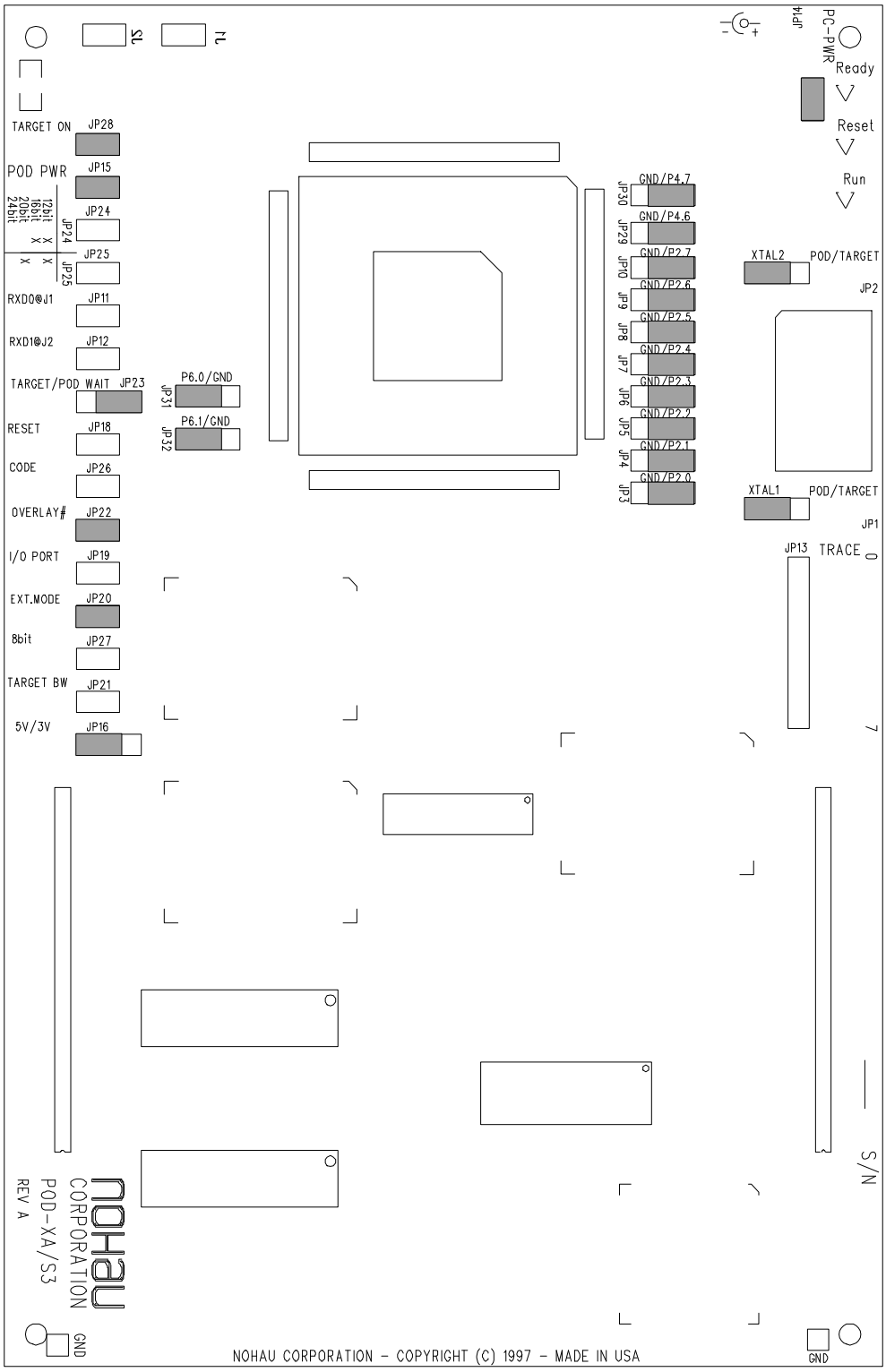


Figure 60: Enlargement of POD-51XA/S3/IE

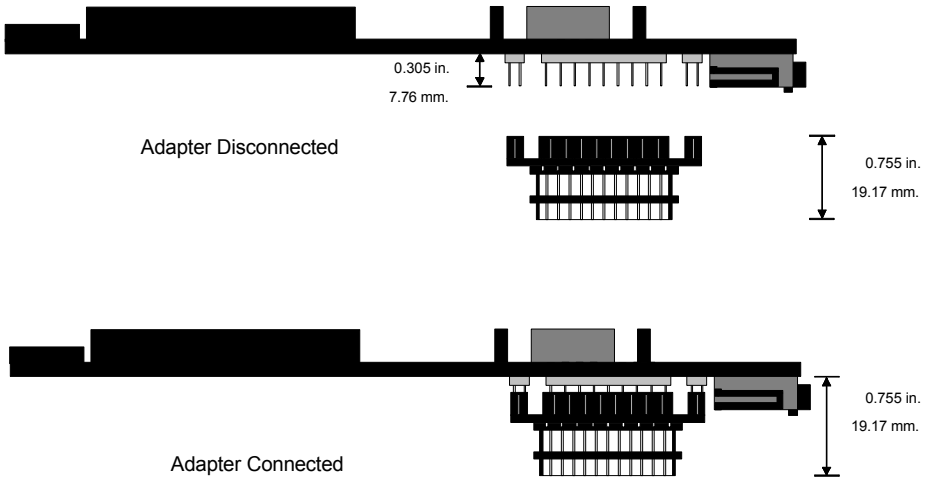


Figure 61: Adapter Dimensions

Dimensions

The pod board measures 6.35 inches by 4.1 inches and requires one inch of space above the target.

The dimensions of the 68-pin PLCC adaptor (ET/AP4-68-SUB1) are shown in Figure 59. This 68-pin PLCC adaptor (ET/AP4-68-SUB1) is attached to a 72-pin connector on the bottom of the pod board. The location of the 72-pin connector is shown in Figure 59. The dimensions of the 68-pin PLCC adaptor (ET/AP4-68-SUB1) are shown in Figure 61.

Internal / External Trace Board (IETR) Installation

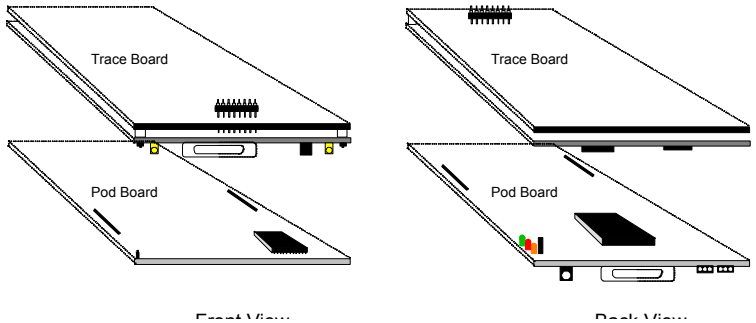


Figure 62: Internal / External Trace Board (IETR) Installation

Emulation Memory

The microcontroller directly addresses 128 Kbytes of code and 128 Kbytes of data memory. If you need more memory, pods with 1 Mbyte and 2 Mbytes of emulation memory are available. Email sales@icetech.com for information about ordering a one- or two-megabyte pod.

Memory can also be configured as

- Kbytes of code memory without data memory (256K pod)
- Kbytes of code and data overlay (256K pod)
- Kbytes of code and 512 Kbytes of data memory
- Mbyte of code without data memory (1-Mbyte pod)
- Mbyte of code and data overlay (1-Mbyte pod)
- Mbytes of code without data memory (2-Mbyte pod)
- Mbytes of code and data overlay (2-Mbyte pod)

See the “Code Header—JP26 and Overlay #Header—JP22” section on page 7 for details on the jumper settings.

Headers

Figure 60 shows the headers with their jumpers in the default positions. When shipped from the factory, all jumpers are in place for stand-alone operation (without a target). This stand-alone operation runs code from external code memory after reset, 24 bits of address, and 16 bits of data.

When you connect this pod to a target, be sure to examine all the jumpers for correct placement. The following sections describe the correct placement for these jumpers.

Clock Headers—JP1 and JP2

These two headers each have two jumper positions: TARGET and POD. They must be moved as a pair. With both headers set in the TARGET position, the on-pod XA bondout chip receives the clock signal from the target crystal or oscillator. It is important to use a 50 percent duty-cycle oscillator.With both headers set in the POD position, the XA bond-out chip uses the oscillator on the pod.

Note: The XTAL1 and XTAL2 signals from the pod are disconnected from the target when the clock jumpers are in the POD position.

EXT Mode Header—JP20

If you operate the XA in Internal mode (see Figure 63), then you need to remove JP20. If you operate the XA in External mode (see Figure 64), then you need to install JP20.

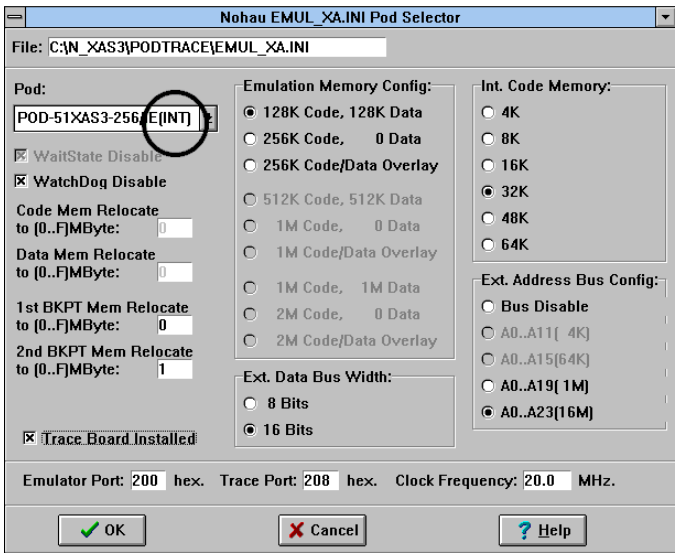


Figure 63: Selecting the Internal Mode Operation

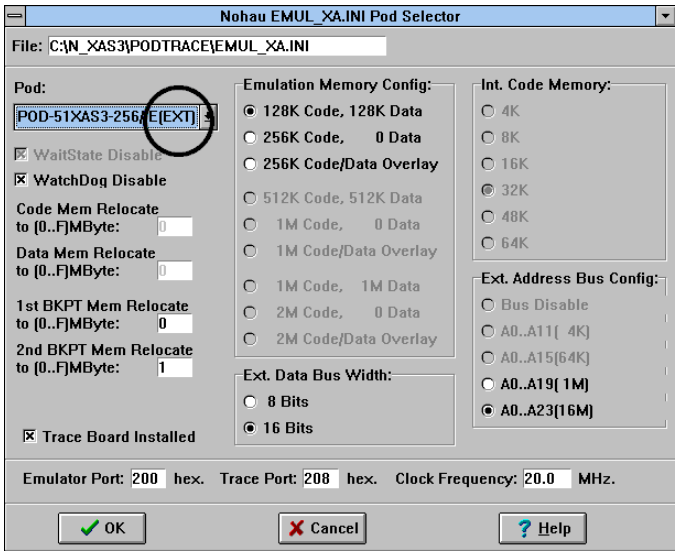


Figure 64: Selecting the External Mode Operation

PC–PWR Header—JP14

If you run the pod with the EPC interface, install JP14 and use the external power supply. When using the LC–ISA, we recommend using the external power supply and removing JP14. To run the pod with PC power, install JP14 (the external power supply is not used).

POD–PWR Header—JP15

Remove JP15 if you are using power from the target for the CPU. If JP15 is not removed in this circumstance, the target VCC is connected to 5 V from the pod. If the target requires less than 0.5 amps current, the pod can be used to power the target with JP15 installed. Higher currents cause a significant voltage drop along the current path. This drop in voltage can damage the pod. You need to remove JP15 in order for the pod to operate at 3.3 VDC supplied by the target.

Target On Header—JP28

If you connect the pod to a target that could be affected by the pod outputting 1.8 volts at the XA VCC pin and the I/O pins, remove JP28 before applying power to the target. Reinstall JP28 after applying power to the target. Similarly, remove JP28 before you turn off the target power. JP28 has this off-and-on capability to avoid voltage problems. By removing JP28, all the pins of the XA are tristated. However, after applying power to the target, you must reinstall JP28.

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

5 V and 3 V Header—JP16

If you operate the XA at 5 V, set JP16 at the 5 V position. If you operate the XA at 3.3 V, set JP16 at the 3 V position.

RXD Headers—JP11 and JP12

If your target outputs debugging information on the serial port, you might want to connect an RS232 device (a terminal or a PC). This pod includes a MAX232 chip that converts the signal levels from RS232 to TTL levels. If you place a jumper on either RXD header, the MAX232 chip drives the serial port input pin on the XA bondout chip. To keep the MAX232 chip from driving the serial input pin on the XA bondout chip, remove the jumper on the RXD header.

RS232 Headers—J1 and J2

Header J1 connects to serial port 0. Header J2 connects to serial port 1.

Trace Header—JP13

With the optional trace, these eight pins monitor any eight logic signals on your target board. The Trace menu displays these pins as TR0 – TR7. TR0 is closest to the JP13 label and TR7 is closest to the 5 V/3 V jumper, JP16.

Reset Header—JP18

Occasionally, a target contains an external device designed to reset the XA chip by pulling the RST pin low. During debugging, this reset might be inconvenient. The signal from the target RST pin passes through the RST jumper. Removing the RST jumper prevents the external device from resetting the XA bondout chip.

TARGET / POD Wait Header—JP23

This header has two jumper positions: POD and TARGET. When the pod operates in stand-alone mode (without a target), set JP23 in the POD position. The pod provides the WAIT signal to the on-pod XA bondout chip. When the pod operates with a target, setting JP23 in the TARGET position connects the XA to the target WAIT signal. The target WAIT signal passes through JP23.

I/O Port Header—JP19

JP19 enables the pod to recognize the signals on P3.6 and P3.7 as I/O signals, instead of WR and RD signals. Install JP19 if the XA operates in internal mode, and uses all pins on P0, P1, P2 and P3 as I/O. If the XA does not operate in internal mode, remove JP19.

Target BW Header—JP21 and 8-Bit Header—JP27

The Target BW jumper, JP21, must be installed when the pod is connected to a target. The target BUSW signal passes to the on-pod XA bondout chip. When the pod operates in stand-alone mode (without a target), remove JP21.

When stand-alone, the 8-bit jumper, JP27, enables the on-pod XA boundout chip to run with an 8-bit wide data bus. Removing JP27 enables the on-pod XA bondout chip to run with a 16-bit wide data bus.

Note: This pod does not support user programs that can override the bus width setting by writing to the Bus Configuration Register (BCR). The bus width is determined by the value of the BUSW pin when Reset is released.

Code Header—JP26 and Overlay #Header—JP22

Table 10. Jumper Settings for the 256K Pod

256K Pod	Code Header—JP26	Overlay #Header—JP22
128K code, 128K data	Off	On
256K code, 0K data	On	On
256K code, 256K data overlay	Off	Off

Table 11. Jumper Settings for the 1-Mbyte Pod

1-Mbyte Pod	Code Header—JP26	Overlay #Header—JP22
512K code, 512K data	Off	On
1 Mbyte code, 0 Mbyte data	On	On
1 Mbyte code, 1 Mbyte data overlay	Off	Off

Table 12. Jumper Settings for the 2-Mbyte Pod

2-Mbyte Pod	Code Header—JP26	Overlay #Header—JP22
1 Mbyte code, 1 Mbyte data	Off	Off
2 Mbyte code, 0 Mbyte data	On	On
2 Mbyte code, 2 Mbyte data overlay	Off	Off

12 / 16-Bit and 12-Bit Headers—JP24 and JP25

JP24 and JP25 determine the number of address lines that are used by the on-pod XA boundout chip.

Table 13. JP24 and JP25 Settings

JP 25 (12-/16-Bit)	JP 24 (12-Bit)	Number of Address Lines
On	On	12
On	Off	16
Off	On	20
Off	Off	24

Note: This pod does not support user Programs that override the number of address lines setting by writing to the Bus Configuration Register (BCR).

A12 – A23 Headers—JP3 – JP10 and JP29 – JP32

The A12 – A19 headers each have two positions: P2.x and GND. Set these headers according to the number of address lines that were set by the JP24 and JP 25 jumpers The positions for A20 and A21 are P4.x and GND. The positions for A22 and A23 are P6.x and GND (see Table 5).

Table 14. A12 – A19 Headers—JP3 – JP10 Settings

	Number of Address Lines			
Header	**12**	**16**	**20**	**24**
A12	GND	P2.0	P2.0	P2.0
A13	GND	P2.1	P2.1	P2.1
A14	GND	P2.2	P2.2	P2.2
A15	GND	P2.3	P2.3	P2.3
A16	GND	GND	P2.4	P2.4
A17	GND	GND	P2.5	P2.5
A18	GND	GND	P2.6	P2.6
A19	GND	GND	P2.7	P2.7
A20	GND	GND	GND	P4.6
A21	GND	GND	GND	P4.7
A22	GND	GND	GND	P6.0
A23	GND	GND	GND	P6.1

Features and Limitations

The emulator uses six bytes of stack space in a large memory model and four bytes of stack space in a small memory model. You need to add six (or four) bytes to your stack size calculation to avoid a stack overflow exception at 0080H.

Emulation Memory

One of the following for the 256K pod

- 128K code and 128K data memory (non-relocatable)
- 256K code and 0K data memory (non-relocatable)
- 256K code and 256K data memory overlay (non-relocatable)

One of the following for the 1-Mbyte pod

- 512K code and 512K data memory (non-relocatable)
- 1 Mbyte code and 0 Mbyte data memory (non-relocatable)
- 1 Mbyte code and 1 Mbyte data memory overlay (non-relocatable)

One of the following for the 2-Mbyte pod

- 1-Mbyte code and 1-Mbyte data memory
  (Both 1 Mbyte memory is relocatable throughout the 16-Mbyte address space in one of the sixteen 1-Mbyte blocks)
- 2-Mbyte code and 0-Mbyte data
  (Both 1 Mbyte memory is relocatable throughout the 16-Mbyte address space in one of the sixteen 1-Mbyte blocks)
- 2-Mbyte code and data memory overlay
  (Both 1 Mbyte memory is relocatable throughout the 16-Mbyte address space in one of the sixteen 1-Mbyte blocks)

Software Breakpoints

You can set software breakpoints wherever there is emulation code memory.

Hardware Breakpoints

The POD–51XA/S3/IE has two 1-Mbyte hardware breakpoint blocks that are relocatable throughout the 16-Mbyte address space of the pod.

- All code address, one instruction skid
- External data read/write address with word resolution

Fast Break Write

Fast Break Write is available when the pod is operating in the external or internal mode.

Data / Address Bus Configurations

The configurations of an 8-bit data bus and a 12-bit address bus in external mode are not supported.

Operating Frequency

1 MHz to 20 MHz in 16-Bit Mode

WM0 must equal 1 in BTRL. Table 6 shows the external bus signal timing configurations.

Table 15. Configurations for 1 MHz to 20 MHz in 16-Bit Mode

	CR1, CR0	CRA1, CRA0	DW1, DW0	DWA1, DWA0	DR1, DR0	DRA1, DRA0
00	Supported	Supported	N/A	Not supported	N/A	Supported
01	Supported	Supported	N/A	Supported	N/A	Supported
10	Supported	Supported	N/A	Supported	N/A	Supported
11	Supported	Supported	N/A	Supported	N/A	Supported

20 MHz to 30 MHz in 16-Bit Mode

WM0 must equal 1 in BTRL. Table 7 shows the external bus signal timing configurations.

Table 16. Configurations for 20 MHz to 30 MHz in 16-Bit Mode

	CR1, CR0	CRA1, CRA0	DW1, DW0	DWA1, DWA0	DR1, DR0	DRA1, DRA0
00	Not supported	Not supported	N/A	Not supported	N/A	Not supported
01	Supported	Supported	N/A	Supported	N/A	Supported
10	Supported	Supported	N/A	Supported	N/A	Supported
11	Supported	Supported	N/A	Supported	N/A	Supported

1 MHz to 20 MHz in 8-Bit Mode

WM0 must equal 1 in BTRL. Table 8 shows the external bus signal timing configurations.

Table 17. Configurations for 1 MHz to 20 MHz in 8-Bit Mode

	CR1, CR0	CRA1, CRA0	DW1, DW0	DWA1, DWA0	DR1, DR0	DRA1, DRA0
00	Supported	Supported	Not supported	Not supported	Supported	Supported
01	Supported	Supported	Supported	Supported	Supported	Supported
10	Supported	Supported	Supported	Supported	Supported	Supported
11	Supported	Supported	Supported	Supported	Supported	Supported

20 MHz to 30 MHz in 8-Bit Mode

WM0 must equal 1 in BTRL. Table 9 shows the external bus signal timing configurations.

Table 18. Configurations for 20 MHz to 30 MHz in 8-Bit Mode

	CR1, CR0	CRA1, CRA0	DW1, DW0	DWA1, DWA0	DR1, DR0	DRA1, DRA0
00	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
01	Supported	Supported	Supported	Supported	Supported	Supported
10	Supported	Supported	Supported	Supported	Supported	Supported
11	Supported	Supported	Supported	Supported	Supported	Supported

Mapping Capabilities

The mapping capabilities map code and data with 128 bytes of resolution. The mapping capability covers the entire address range of the POD–51XA/S3/IE (16 Mbytes).

Trace

Trace can trigger/break on code fetch/code read (MOVC)/address/data, external data read/write/address/data.

Shadow Memory

Shadow memory supports speeds up to 30 MHz, and is available when you use an IETR.

Chapter 10: Software Support

Compilers

HI-TECH HPDXA

This integrated development environment is quite intuitive, although still a DOS-windowed application. All linker directives are done through pull-down window option lists, such as ROM, RAM, and NVRam start locations that are target specific. The G3 part has 32KB of on-chip EPROM/ROM and 512 bytes of on-chip data RAM starting a 0x0000. The POD-51XA/G3/I can support future derivatives with up to 64KB of on-chip EPROM/ROM.

Note: The emulator software version referred to is 1.0H.

Starting a project using the HPDXA

1) From the **Make** menu select **New project…** and enter a project name. You will then be asked to select processor and memory model. Currently the processor type cannot be changed, but you can select the appropriate memory model for your project.
2) For the output file format, select Intel HEX.
3) Make any necessary changes in ROM & RAM addresses.
4) For compiler optimization, select No optimization (there is a problem using optimization with the current emulator software which will be fixed in a future release).

Enter your source filenames and make sure the **Pathnames** option is set to **relative**.

1) From the Options menu select **Source level debug menu**. If there is already a check in front of it, leave as is.
2) From the **Make** menu select **Symbol file name** (it should have the same name as the project). For example, if the project is test.prj, then the symbol file would be test.sym.

Using XAC Command Line Compiler

1) Use -Gfile option. The `file' in `-Gfile' should have the same name as the final output. For example, if the final output is test.hex, then the -Gfile option would be -Gtest.sym.
2) Because the default output type of the compiler is Intel Hex, it is not necessary to use the option to specify the output file type.

Example 1: Single source file, test.c

```
XAC -A0,20,1E0 -GTEST.SYM TEST.C
```

Example 2: Multiple source files, test.c, test1.c, test2.c

```
XAC -C -G TEST.C TEST1.C TEST2.C

XAC -A0,20,1E0 -GTEST.SYM TEST.OBJ TEST1.OBJ TEST2.OBJ
```

3) For both cases, the compiler will generate one '.hex' file, one '.sym' file and for each source file, the compiler will generate one ".sdb' file.
4) All the files mentioned above (.hex, .sym, .c and .sdb) should be in one directory.

HIWARE / Archimedes

Nohau emulators support the HIWARE debug format.

Tasking

Nohau supports the Tasking debug format. See the readme.txt file under ..\emul51xa\tasking and the make file for required compiler and linker options.

Chapter 11: Troubleshooting

Overview

If you have trouble with your emulator, email support@icetech.com. If they call the engineer will likely lead you through the following steps.

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 found elsewhere in this manual.

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

Pod Problems

Step 1: Board I/O Addresses

Take the POD out of the target and set XTAL (JP1, JP2), PWR (JP16) and EA/WAIT (JP3) jumpers to POD. Confirm that the I/O address set in the jumpers on the emulator board agrees with the software settings found in their respective configuration dialog boxes. Run the DOS CONF_XA.EXE Program in the EMUL51XA directory. This tests the POD, emulator and trace boards using POD crystal, and PC power. Use port address 208 for the piggyback IETR.

Step 2: PWR and XTAL jumpers

Now we know the emulator, POD and trace operate correctly standalone. You might also try running the supplied demo Programs, such as TIME.HEX. Reset and single step with F7.

The POD-51XA/G3/I must see the /EA pulled to VDD on reset; in other words it must start up in single chip mode. If your target is a ROM-less design, use the POD-51XA/G3/E with /EA pulled low on RESET.

Now reposition the XTAL (JP1, JP2) and PWR (JP16) and EA/WAIT (JP3) jumpers to target position and connect the pod to your target board. Pull Buswidth (JP4). Open a Data window to address 0 and poke on-chip RAM. If not successful, check the XTAL1 TARGET POD crystal jumper pin for square wave if using an external clock, or XTAL1 and XTAL2 for square wave using the on-chip oscillator. If nothing is present, either the target crystal circuit is inoperative or the PLCC connector is open, shorted, or not seated properly into the target socket. Continuity checks can ring out these pin connections.

The other possibility is no VDD to the POD. This would point to openings in the PLCC adapter.

Step 3: Check Hardware Configuration

Double-check that the POD jumpers agree with the **External Address Bus Configuration** in the **Hardware Configuration** screen. This would most likely show up as incorrect trace synchronization.

Step 4: NOP Tests

Note: See Chapter 12 for further examples.

With emulation RAM mapped to the emulator (**Configuration/Memory Map**), open a **Data** window to address 0x000, with address space set to INT CODE. Enter a valid PSW, such as 8f00, and at address 2 a reset PC value of 0120. **<CTRL> F2 Reset** should now open your **Program** window to address 0x120. Now select the **Program** window and in-line assemble some NOPs and a return, JMP0120. **Step F7** should move through the NOPs and take the return branch. **F9 Go** should light up the green POD run light. **F9** again will break the program.

The trace, if installed, will show the NOP loop over and over again by moving the elevator box up the trace display screen. Clicking on an address in the **Program** window will set a breakpoint here.

Step 5: Single Steps, Resets, or Runs Briefly Then Halts

Should the **Program** window single-step but not run, check that the **Trace Setup** screen shows that is installed (if present). If you have a hardware watchdog circuit separate from the XA chip itself, this must be disabled (might be indicated by a near-solid red **Reset** LED). One way is to disable the **Reset** line to the POD by removing jumper J20/Reset. Another is to cut a trace or use a DIP Isolator socket.

Check the **Hardware Configuration** screen to verify you have checked **Watchdog disable**. If not, then your program will continue to reset when stepping. Final production code can enable the watchdog; however it must have a routine to “feed the watchdog.”

Look at your stack pointer. If close to 0080H, you risk a stack overflow exception and your program will vector off to the address at word 000EH.

Step 6: Individually open lines

If the above NOP test still fails, and you have tested the PLCC adapter for pin to pin shorts, then purchase a 44 pin isolator adapter and individually open lines until you gain control of the POD (start with the XTAL pins and jumper these back to the POD).

Trace Problems

Run the CONF_XA.EXE confidence test described in the previous section, POD Problems. Passing this but failing to trigger correctly would probably indicate incorrect trace setup and trigger options. Working through Chapter 12: Tutorial can answer most questions. Additionally, Chapter 4: Internal/External Data Trace Board, goes into the details of each trace setup menu item.

Triggers don't work

Check the setup screen and be sure **Ext trig line: Trig Inhib** is checked. The **External Trigger Input** at JP4 is always pulled high. Triggers are inhibited only if some external signal holds JB4 low. If you had selected **Ext trig line: Trig**, triggers would work only when the line was held low. Further discussion may be found on page 81, "External trig line" in Chapter 4.

Data Bus Triggering not working

Triggering amounts to a match of comparators of address, data, and bus cycle type. Triggering on data reads and writes can be particularly tricky when variables are byte wide, because the trace always collects 16 bits of information. Using 16-bit memory, byte writes to odd addresses - for example, requiring masking off the low-order byte using FF00 (see page 78, Data Bus Qualifier , in Chapter 4). Using 8-bit memory, you can mask off the upper byte of captured trace data with 00FF. You may need to look at the captured trace data to understand the correct data mask qualifier. You can set the filter to capture all read/write cycles to a particular address, then look at the captured data display.

Internal and/or External Bus MUST be selected

The IETR piggyback has two check boxes, **Internal Bus** and **External Bus**. Checking **Internal Bus** means that, after valid trigger match, internal bus cycles are used to count down the **Post-trigger samples:** entry before stopping trace capture. The same applies to External Bus countdown.

If neither box is selected, you will get notification of trigger found, but the trace will continue to collect cycles and overwrite the original trigger point. The trace would have to be stopped manually and have no useful data.

Understanding Sequential Nature of Trig1 then Trig2 then Trig3

The three trigger conditions are sequential, i.e., without an active, Trig1, trig2 and trig3 will not happen. If you wanted a condition where three or four different events would be "**OR**ed" together, use only Trig1 and keep selecting **New**. Some developers have tried to use Trigs 1, 2, and 3 for three commonly used trigger conditions, assuming an **OR** rather than a **THEN** condition. Trigs 1, 2, and 3 are implemented as a State Machine - the higher numbered triggers are NEVER considered unless Trig1 is active and fires.

Last trig event repeat count

This should not cause confusion, but it needs emphasizing. If you had both Trig1 and Trig2 active with a repeat count of 3, the logic: If Trig1 then (Trig2 for three times) occurs, start counting down **Post Trigger Samples**, then stop trace capture.

Check both Address and Data Bus conditions

Each trigger event contains **Address** and **Data Bus** qualifiers. Leave the **Data Bus** blank for opcode trigger addresses and use, if desired, for read and/or write cycles. Both fields are not displayed simultaneously, thus you may have left qualifiers in the **Data Bus Edit** window, even though you have gone on to a new trigger **Address** location. You would catch this by clicking in the **Data Bus** diamond for that Trig event number. Use the Delete key to clear the qualifier, if present.

Summary

ICE Technology has helpful technical support engineers, as well as knowledgeable sales representatives and distributors. Don't hesitate to call them for help.

Chapter 12: Tutorial

This chapter is designed to cover the basics of EMUL51XA-PC operation. You can then go back later to review the more detailed descriptions on memory coverage, Program performance analysis, and advanced trace features such as **Window** filter mode.

The first section is directed at the hardware engineer, with in-line hand assembly of a simple Program. We will then discuss C source level debugging using the time.hex file located in the ..\hitech subdirectory. If your organization divides hardware and software functions, ideally the hardware designer will use the ICE to validate his design and then spend time with the software engineer explaining the memory map and hardware configuration screen settings.

Note: References to "target" or "target board" mean your circuit board which actually contains the XA-G3 part. "POD" refers to the in-circuit emulator pod with the bondout chip.

Hardware Issues

Is the Target Board On?

1) As soon as the equipment is installed, run the confidence test described in Chapter 1: Software User Interface. If this fails call or email your local sales representative. You can call support at 408.626.7893 or email technical support at support@icetech.com.

2) POWER EVERYTHING DOWN, then:

- Plug the POD into your target board. Note pin 1 (with the ---v---- notch) orientation: looking down, the POD cable will exit to the left when pin 1 is pointing to the on-pod crystal.
- The PLCC adapter should have a snug fit.
- The black ground wire isn't important (although designers recommend connecting it before POD plug-in to prevent any static discharge).
- Pull JP4, BUSWIDTH, so target sets BUSWIDTH
- Pull the PWR jumper, JP16, because your target should have it's own VDD.

*Note: If running 3.3 VDC, in addition to removing the PWR jumper, make changes to the IETR setup screen under **Target VCC:** .*

- Move the 2 crystal jumpers, JP1 and JP2, from POD to TARGET position to use your on-board target oscillator.
- Move JP20, /EA/WAIT, from POD to TARGET.
- All the P2 jumpers (next to crystal) will be on the left position for 16 bit data bus/ 20 bit address bus operation. This setup can be complicated for other data bus/address bus combinations, so you should review the Headers section of the POD-51XA/G3 in Chapter 6: POD-51XA/G3/I or Chapter 7: POD-51XA/G3/E for other size combinations.

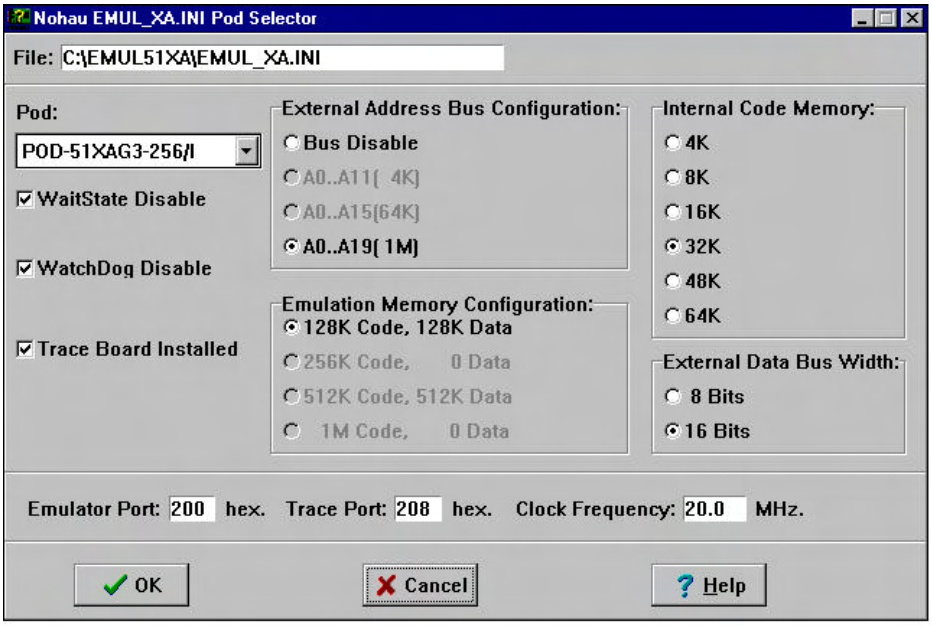


Figure 65: POD-51XA/GE/I Hardware Configuration Screen

3) Power up the PC and then the target board. Select the XA INI Generator icon and you will see the **Hardware Configuration** screen above. This screen must match your target design (see Chapter 1, "Quickstart Installation"). A **Send Command** box usually means the POD is not seeing a target oscillator or VDD (see Chapter 11, or call for help if needed).

Exercising Memory

By default, all memory, data and code is mapped to emulator SRAM. Try opening three new data windows from the **Window** menu. These will land on top of each other, so reposition them where you wish for proper viewing.

By default, these windows look at internal data RAM space. Change one to Shadow space and the other to Internal Code space by clicking on the **Data** window with the right mouse button, then select **Address space** with the left mouse button.

Code space is self-explanatory. Display this as 16-bit hex for the moment by again using the right mouse button and selecting **Display as..**

The **Internal Data** window may stay as is. Enter a few values here by selecting a word and simply typing over the existing value. Internal data RAM on the G3 runs from 0 to 1FF.

Note: After 1FF, this window defaults to EXTERNAL data memory.

The **Shadow** window lets you see external memory writes while the target system is running (updates 4x sec - requires IETR).

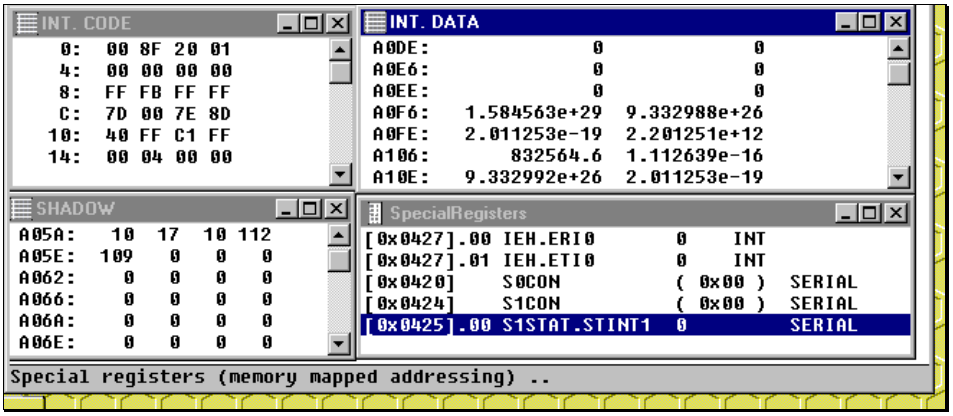


Figure 66: Data and Special Register Windows

Open the **Special Register** window. With the right-click mouse, add the SFR registers or bits of interest.

Another way to look at SFR space together with a detailed bit usage description is from the **View/Edit** menu (**Default CPU Symbols**). This .reg file is loaded manually under the **File** menu by choosing **Load Default Symbols..**, or automated at emulator startup by checking the **Load Default Symbols** box in the **Config/Miscellaneous** screen.

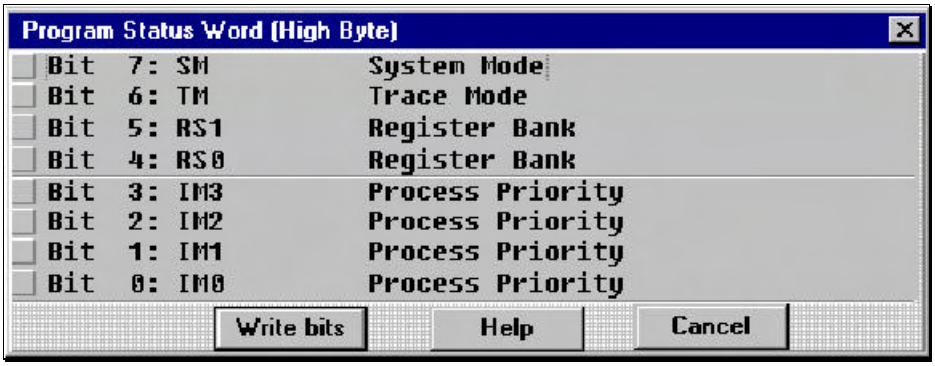


Figure 67: View Bits

Warning:	***You can set PCON.0 IDL, but should not set PCON.1PD from these windows. Power Down turns off the bondout clock and the emulator, then fails to communicate with the POD. The user Program can enter Power Down, but the CPU must be out of Power Down to break emulation (through reset or external interrupts)***

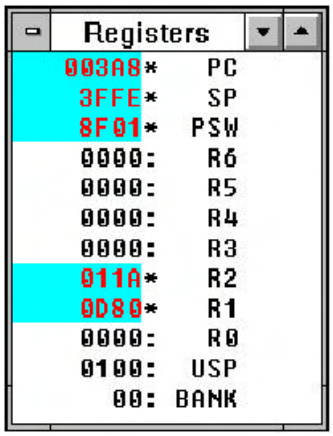


Figure 68: Registers Window

Open the CPU **Register** window from the **Window** menu, sizing it to cover all the registers, and position it on the screen above. Change from bank 0 to bank 1 by typing over the BANK register with 1. Then enter data values in any of register R0 to R6. Observe that changing back to bank 0 also changes these Rx values. Generally you don't want to change PC, PSW, SSP, USP, and Rx values here, but use this window for observation purposes only. The Program will manipulate these values.

Hand-Assemble Some Simple Program

Go into the **INT CODE DATA** window at location 0 (<CTRL>A), and enter the reset PSW of 8F00. The **RESET** Program counter for the EMUL51XA-PC is at location 2. Enter 120 here. Now open the **Program** window (**Window** menu) and click on the **Reset** toolbar, or press <CTRL>F2. This should set the cursor to 120. Select the middle of this line with the mouse and start entering assembler mnemonics:

```
NOP

NOP

NOP

NOP

JMP 0120
```

Step the Program with F7. Click on a NOP and use F4 Go to Cursor for temporary software breakpoints. If you develop some useful test routines this way, save them under the **File** menu as .HEX files.

Clicking on the left address sets a permanent software breakpoint.

Note: Customers setting a breakpoint normally use a software breakpoint. We suggest using a hardware breakpoint only when your code is in PROM or EPROM.

Explore the **Run** and **Breakpoint** menus on your own.

---

*Note: The **Reset and Go** option is not implemented.*

Hardware breaks only! permits breaking even when code memory has been mapped to the target Eprom/Flash. This results in a skid of one opcode.

Break on external access allows breakpoints on data reads and writes, however only to 16-byte resolution (for example, an address of 200H would break on 200 through 20F, or 20XH where X is Don't Care). Due to the instruction pipeline, the break address will skid by the prefetched word count.

*Note: The IETR 128/512 also allows breaking at a unique external address and data bus value using the trace trigger setup and choosing **Break Emulation? Yes, on trig** or **Yes, on trace stop**.*

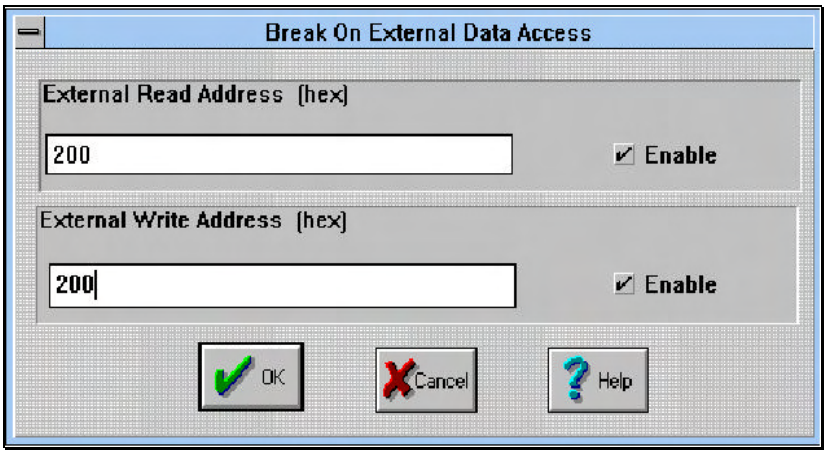


Figure 69: Break on External Data Access

Testing Target Data RAM

Assuming you are using the external data bus, open another Data window and **select Address space.. EXT DATA**. By default, this looks at Emulator SRAM. Change this to target under **Config/Memory map..** by checking the box under **Map to Target**, then in the blank space on the immediate right Click and enter an appropriate address range (for example, 0-FFFF).

*Note: If you need to change your setup, uncheck the box, select **OK**, then recheck the box and enter the corrected address range.*

You might use the **Fill** command to write 00 to the entire memory and observe any memory read back error messages.

If you see any error messages, start looking for shorts / opens / missing strobe lines, etc. Do not exclude checking the PGA to PLCC adapter.

Software Issues

High-Level C-Source Debugging

You should now be able to peek and poke into all the various data spaces, SFR, internal code, and on-chip and external data memory. The rest is relatively easy, and no different than running Borland or Microsoft C debuggers on a PC.

First verify that you have opened a **Source** window from the **Windows** menu. The Nohau HLD interface uses separate **Program/Disassembly** windows and **Source** windows; only one of each need be open at any time.

Note: Actual addresses may change with later versions of time.hex.

Now load the time.hex Program, found under ..\hitech. If you have selected the **Source** window as active window (CTRL F6 or mouse-click in this screen), F7, F8, and other **Run** menu options will operate on the C source line level. Selecting the **Program** window, however, will show the Step at the assembler opcode level. No C source statements? Check your **Config/Paths** setup. The debugger looks for .C files from the default load directory but also multiple source directories separated by semicolons as shown in Figure 70.

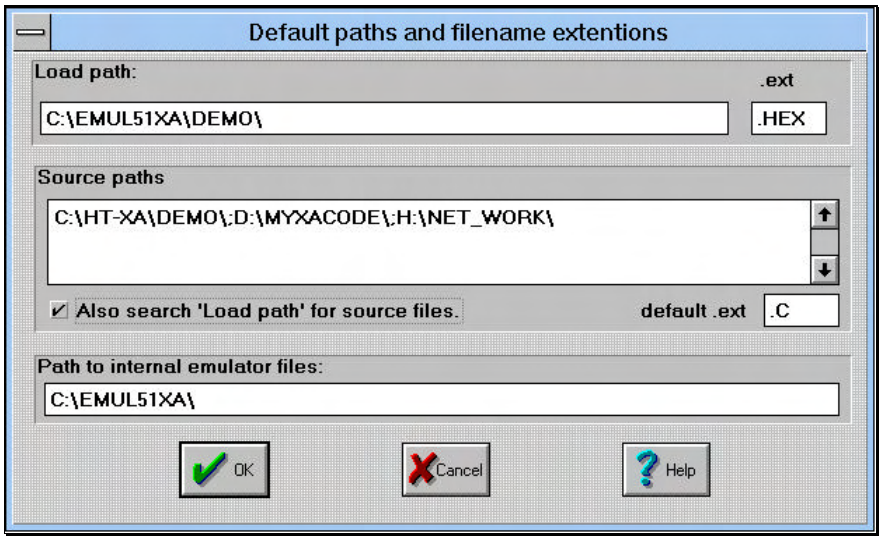


Figure 70: Paths Configuration

WATCH, INSPECT, and EVALUATE C Source Windows

Step the C source with F7, using F8 to Step Over C function calls. Temporary breakpoints are set by clicking on an executable C statement and using F4, **Go to Cursor**.

Use the temporary breakpoint to stop in the timer0 interrupt function called **pitr_int().** In the **Source** window, use your right mouse button to select **Functions**, then click on **pitr_int().** Now select a statement such as **ticks**++ with the mouse and simply use F4 to execute to this point.

Similarly, this function contains a structure called timer. Under the **View / Edit** menu you will notice CTRL-E, I, and W hot keys being used **to Evaluate a C expression, Inspect arrays and structures**, and **Add a Watch** point. Selecting **ticks** in the **Source** window with the mouse and CTRL W adds this to a **Watch** window. Individual timer structure members may be selected by pointing the mouse on the member side of the structure name. Try adding the seconds member **timer**.**sec** to the **Watch** window.

Now step through ticks++ with F7. The **Watch** window should show ticks incremented by 1.

Another useful window is **Inspect**. This is the default window type if you double click on a variable. Open **Inspect** windows for the structure "timer", and the string "show" by double-clicking.

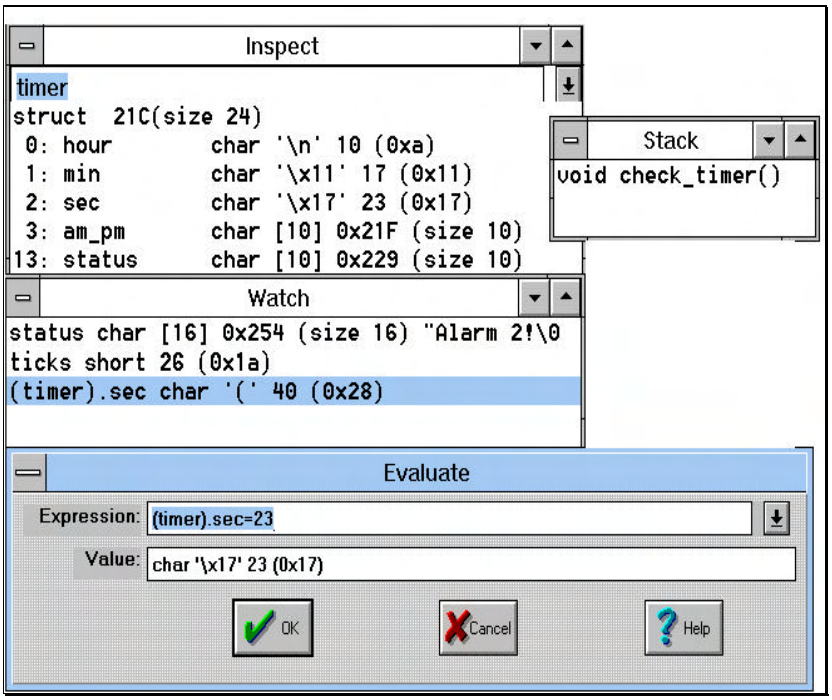


Figure 71: HLD Windows

Editing a C Variable

Use the **CTRL E Evaluate** option after highlighting a variable to change, for example, **timer.sec**. Change this expression to timer.sec = 23. Note this window accepts any number, such as 1000 for example, returning 0x3E8 just as with a HEX calculator.

C Call Stack

Lastly, with function calls you might want to see the nesting order. This window, found under **View/Edit**, shows all nested function calls and associated parameters. Opening this in the **check_timer( )** function will show the function **timer_func ( )** and **main ( )**.

Real-time Debugging with the Trace (IETR 128/512)

Hardware engineers who have used logic analyzers should have little trouble with our **Trace Setup** screen. Software engineers, however, may be new to viewing real-time instruction history. Using the time.hex Program, lets look at a few examples of how the trace might be used with the POD-51XA/G3/I.

First, by default the trace always records all bus address activity when you run your Program. After 131000 (or 524000) records, the circular buffer overflows and new instructions overwrite the old. Lets look at this.

After loading the ..\emul51xa\hitech\time.hex, set a software breakpoint on C source line 102, the start of the timer_func function, by clicking once with the mouse. Use F9 to start the Program. It immediately breaks, and the trace displays some 2860 CPU clock cycles: -2860 to -1 (the most recent opcode fetch). As you move into negative (-) trace frame you return all the way to the startup instruction at -2857, JMP start.

Note that with multiple word opcodes, you see word fetch cycles oo2 until completed. Read cycles R2 and write cycles W2 are also displayed. Hardware engineers may be interested in the internal code bus and seeing the start of instruction cycles that use “s.” This will require showing the internal bus in the trace submenu. Software engineers will prefer the compress option showing actual execute code and read/write activity.

Another useful exercise is to select **Synchronize Program window**. This generates a cursor that moves in the **Source** and **Program** windows as you scroll backwards in the trace buffer. Using the trace in this very limited mode can save man weeks of debugging time, where you run real time to a breakpoint and then at your leisure go back over the previous 131000 (or 524000) records that preceded it.

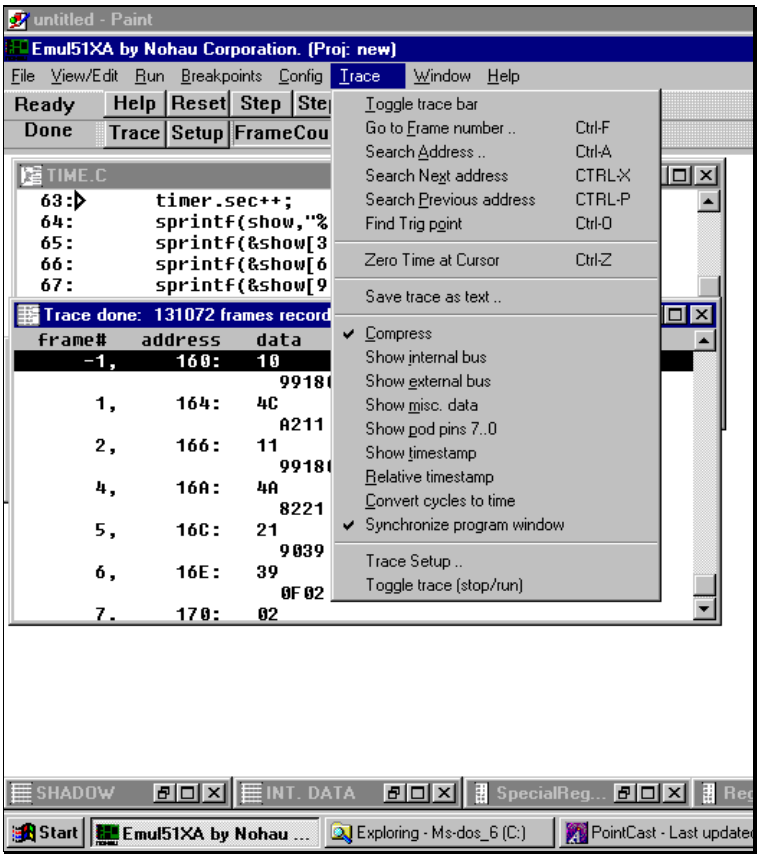


Figure 72: Synchronize to Source Window

Filtering

*Note: Line numbers and addresses may change with later versions of **time.hex**..*

Even 131000 (or 524000) records can disappear in a hurry if you are in a loop. Filtering lets us see only the routines or bus cycles of interest. Try filtering out everything except the pitr_int routine by entering the **Trace / Setup** menu, checking **Filter, Address**, and then **New**. For the starting address, use the function name pitr_int and for the end address use #71 or #TIME#71 indicating C line number 71 in module TIME (note compiler generated labels often default to upper case).

Now restart the trace using using **F10** while the target is still running, or use **F9 GO** if the Program is stopped. Right away you will notice the trace frame count growing only when running through this interrupt. If nothing gets captured make

sure you are running (the green POD led is on). To stop trace capture manually, use **F10**. Observe that only the pitr_int( ) code is present in the buffer.

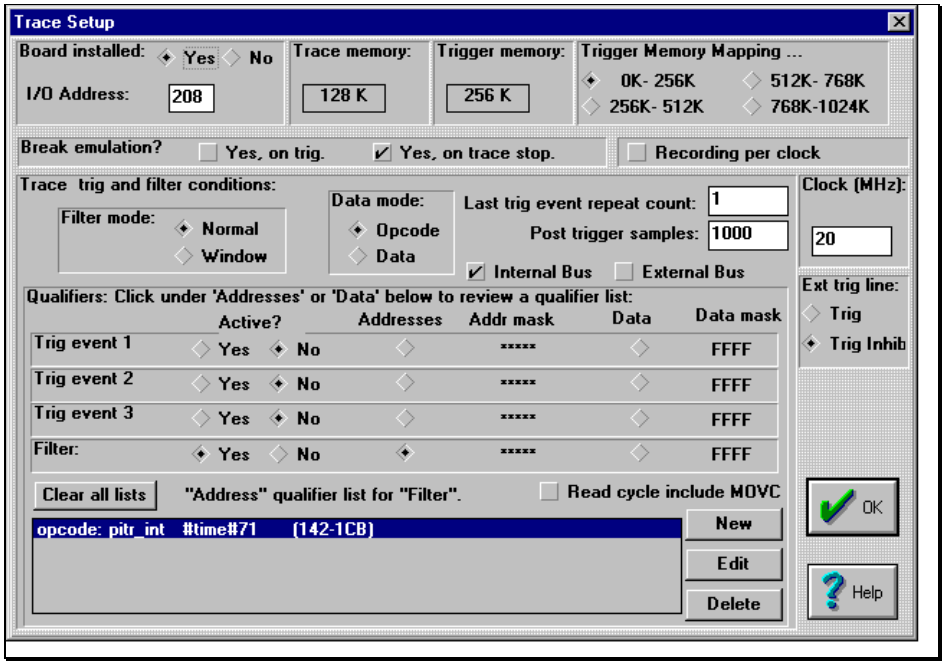


Figure 73: Filtering

Timing

Filtering can also be useful in timing events. In the **TIME** Program, a 1KHZ interrupt is generated with timer0 overflow, calling pitr_int( ) and incrementing the **TICKS** counter. After 1000 counts the timer.sec word is incremented, tested for exceeding 59, and so on.

Go back into the **Trace Setup** screen and change the filter end address to pitr_int() also, so we only capture the initial fetch instruction of the interrupt. Restart the trace capture with **F10** or hit **F9 Go** if you have stopped the Program. Stopping the trace with **F10**, you will notice the only bus activity captured is at address 142, the start of **pitr_int( ).** This is exactly what we want. Now reenter the **Trace** menu and select **Show timestamp**, **Convert cycles**, and **Relative timestamp**.

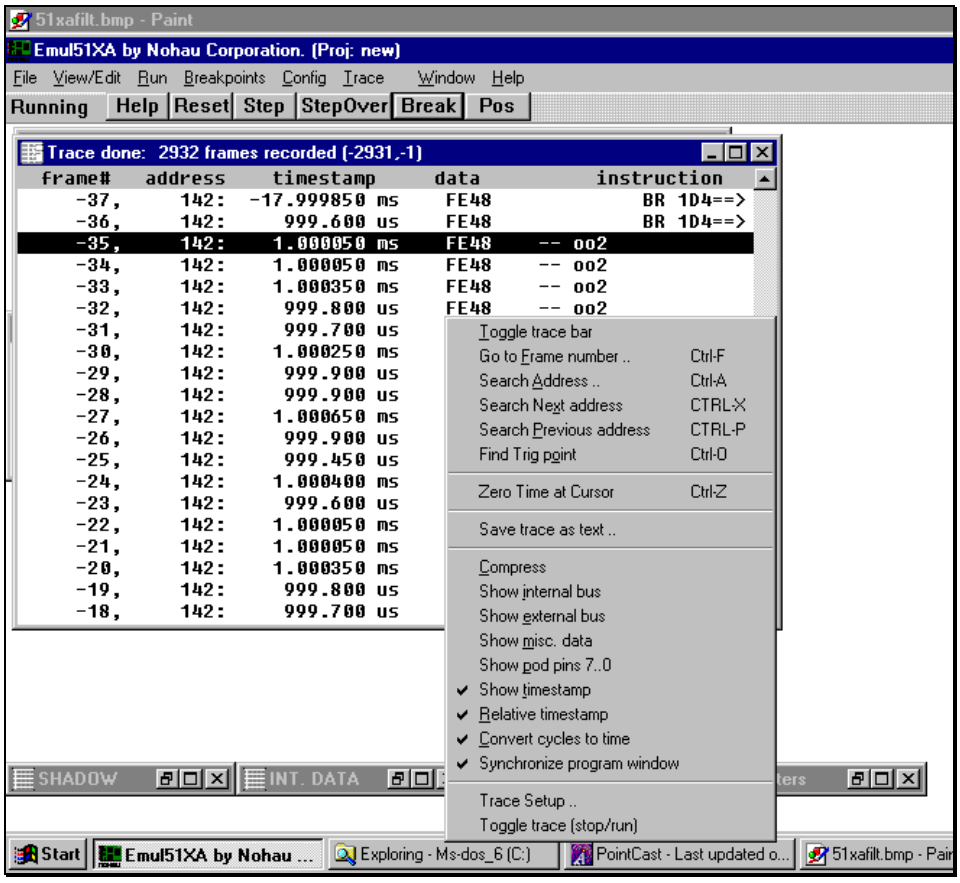


Figure 74: Relative Timestamping

Observe the relative interval between pitr_int( ) calls is about 1 ms.

Note: A 40-bit counter of CPU clocks uses the crystal clock entered on the trace setup screen to convert cycles to seconds. In this instance, we assume you are using the 20 MHz POD crystal.

You may find it easier to measure the interval between writes to **ticks** in external SRAM. Here, you would set **Data mode: Data** and enter **ticks** as **Start** and **End** address, selecting **Cycles: Data write**.

Trace Trigger

If your debug style is to go to breakpoints, then you can scroll back through the trace to see the previous execution thread. Suppose you want to set a breakpoint "on the fly" or examine code in a particular routine without stopping your realtime application. Let's see how this is done with the time.hex Program.

Start the application using F9. Now enter the **Trace/Setup** screen, disable the filter **<No>,** and set Trig event 1 to **<Yes> New** with **check_timer( )** in both **Start** and **End** addresses (use tab key to duplicate start address) and **Opode fetch** cycle type (Note Data mode: Opcode setting in Figure 75.)

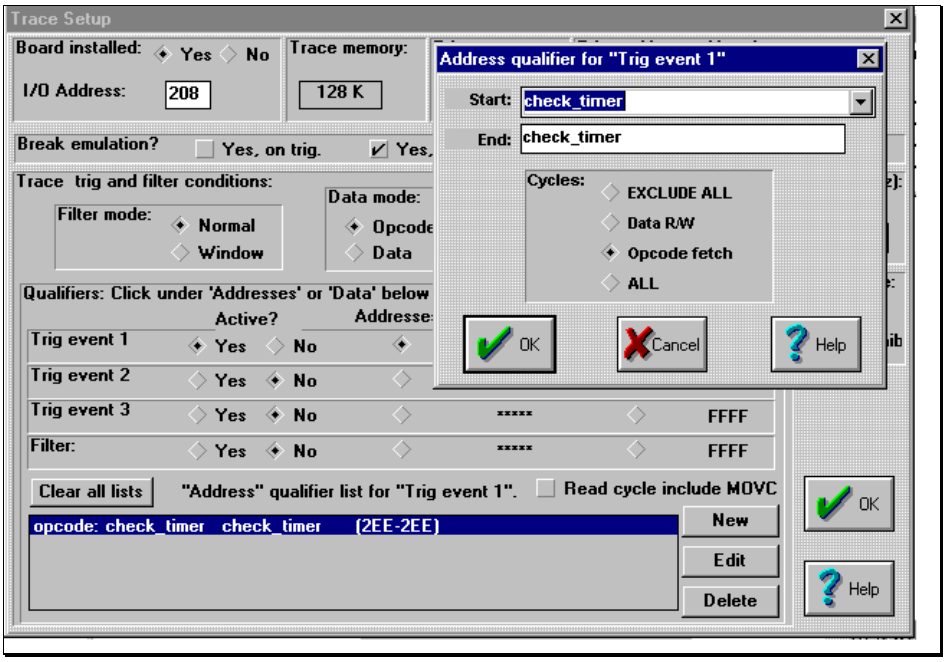


Figure 75: Trace Triggering

Re-initialize the trace logic by stopping trace (F10) and then restarting (F10). If the captured frame count displays -67070,64000, this tells you 67070 address bus cycles before the **check_timer( )** function were captured and 64000 frame after starting this routine were captured (the post trigger frames will differ in your setup, matching the number in the **Post trigger samples** entry).

Setting the **Last trig repeat count** to 10 would mean 10 iterations through this routine before triggering. You can add multiple ORed address ranges to each trigger by continuing to select **New**. For example, on Trig event 1 select **New** and add another function such as **pitr_int( ).** Now, either **check_timer** or **pitr_int** function execution would cause the trace to trigger and stop capturing data when the **Post trigger samples** entry has reached 0.

Trig event 2 and Trig event 3 function just like Trig event 1, *EXCEPT* you must specify previous triggers for them to activate. This then implements a state machine, i.e., Trig 1 THEN trig 2 THEN trig 3, or Trig 1 THEN Trig 2. The **Last trig event repeat count** means you have one repeat counter, and it applies to the last of the defined triggers.

On-the-Fly Breakpoints

After specifying the trigger condition, select **Break emulation**, **Yes, on trig** or **trace stop** (both require an active trigger event). If trace is busy, stop it with **F10**, then restart with **F10** to reinitialize its logic for the new setup. The Program will now halt either right after the trigger or when the post trigger count decrements to 0.

CONGRATULATIONS on successfully completing this tutorial! Less frequently used features can be found in Chapters 1 and 4.

Index

F

G

H

I

J

L

M

N

O

P

R

S

T

