



EMUL196-PC SDU

Preliminary User Guide

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

If you need assistance, please phone ICE Technology
www.icetech.com - email support@icetech.com
Tel: 650.375.0409 - 800.686.6428
Fax: 650.375.8666

Table of Contents

Chapter 1 : SDU Emulator Board	1
Known SDU196 Emulator Limitations	1
Comparison of SDU196 Emulator vs. Full Emulator system	
Detailed Installation Instructions	4
Setting the I/O address jumpers -- J1	
Setting the Target Communication Rate -- Header JP1 ..	5
The PWR Header -- JP2	5
Chapter 2: 196EC FLASH Programmer	7
Disclaimer	7
Introduction	7
SDU Overview	7
Setting Pins for SDU Testmode Entry	
Running the SDU_WIN.exe	
How it works	
Programming for use with Nohau SDU Emulator Software	
Chapter 3: Intel Evaluation Board (196EA)	13
Software Defects	14
Tutorial	14
Debugging with Your Own Target	15
Chapter 4: SDU Software User Interface	17
Detailed Installation Instructions	17
Initial Software Configuration	
Configuring the Software	19
Projects	
Setting the Paths ..	
Mapping memory	

Emulator Hardware Configuration	
Miscellaneous bits	
Miscellaneous Configuration	
Enable Code Space Limits	
Reset vs. Full Reset	
Window Colors	
Fast Break Write	
Memory Coverage	
Detailed Memory Coverage Report	
Performance Analysis	
Menus	45
File Menu	
View/Edit Menu	
Run Menu	
Breakpoints Menu	
Config Menu	
Program Menu	
Source Menu	
Data Menu	
ShadowRam Menu	
Register Menu	
Stack Menu	
Watch Menu	
Window Menu	
Help Menu	
Dialog Boxes	59
Child Windows	59
Register Windows	
Data and Shadow RAM Windows	
Program Windows	
In-line Assembler	
Source Windows	
Other Windows	
Inspect Window	
Tool Bar	68
Help Line	69

Appendix A	71
Reset Cut Required on Intel's 196EA Evaluation Board	71
Appendix B	73
INTEL Serial Debug Unit (SDU) Standardized Connector Specification	73

Chapter 1 : SDU Emulator Board

Known SDU196 Emulator Limitations

Comparison of SDU196 Emulator vs. Full Emulator system

- 1) No Trace Board Support.
- 2) No Shadow Ram support.
- 3) Currently you are limited to one hardware breakpoint.
- 4) No memory mapping capability.
- 5) No RESET functionality with the Intel Eval Board. However, designs with SDU rism code resident in the target will be able to support RESET.
- 6) No DDE capability.
- 7) The pull-down menu 'Config'/'Emulator Hardware' will be able to edit the CCB's, however, this will have no effect on the Intel Board due to the RESET not being functional. (see item #5 above)
- 8) No support for Fast Break Writes.
- 9) No support for Break on Internal Access.
- 10) The LED's on the Gray POD do not function with CBL-B-LC25/25.
- 11) The SDU196 Emulator hangs when application fails. (Requires functional code or efficient error recovery in target app. at all times.)
- 12) Reserved locations 0xE0-0xF7 and 0x400-0x407. (The SDU196 Emulator will fail if these locations are corrupted by users code.)

- 13) Requires that customer links in low level SDU rism code into application.

Note: We have supplied a version of SDU rism code for your benefit. Located in the \EMUL196\SDU\RISM directory, you will find test.a96 and sdurism.a96 required to run on the Intel Eval Board. Do not attempt to use this code in your application without making appropriate changes first. We cannot be responsible for damage caused if this code is used in a different application other than the Intel Board.

- 14) Requires Ram on Users target for loading code.
- 15) The SDU rism code is the property of Intel. Any questions on SDU rism should be directed to Intel.
- 17) The SDU emulator will poll the Target when attached. As a result target applications will not run at the same speed as with the emulator. If this is a problem, simply hit GO and then remove the SDU connector.

Please feel free to inform us of any other limitations or bugs you feel have been left off this list.

For Information on a Full Emulator System email sales at sales@icetech.com

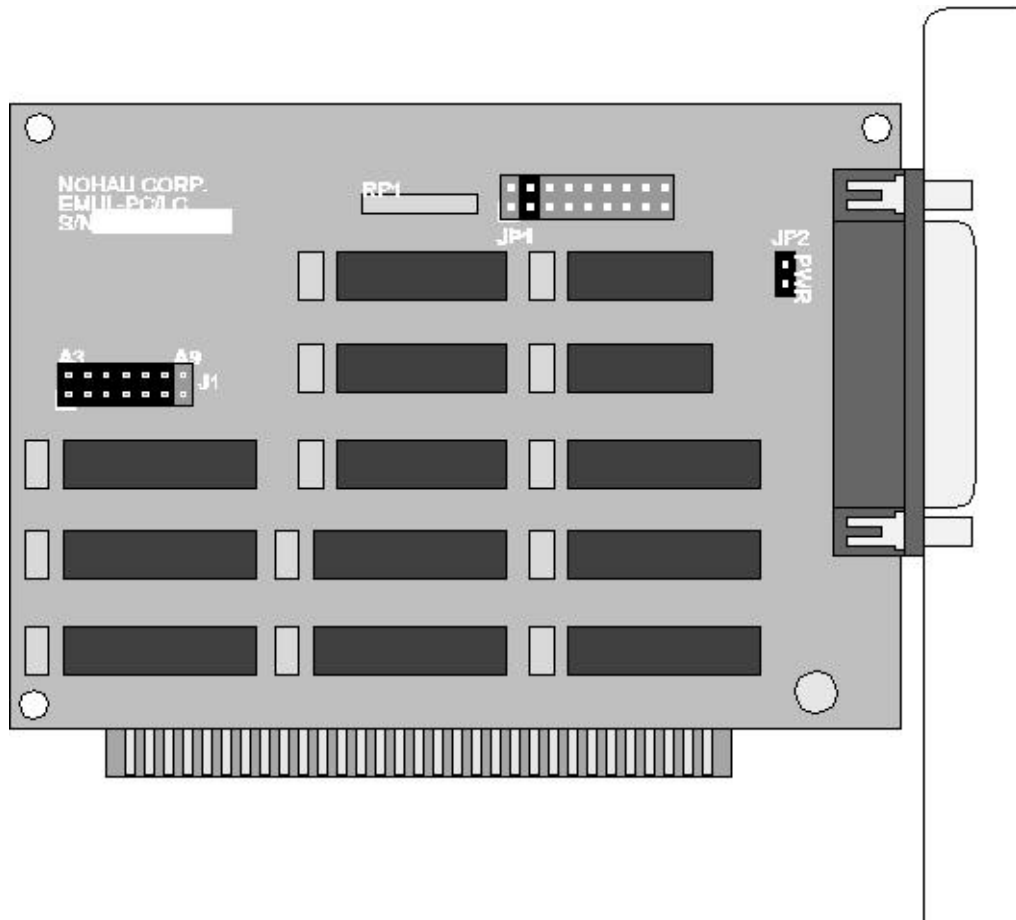


Figure 1: EMUL196-PC SDU Emulator Card

The EMUL196-PC SDU board supports target boards for the 80C196 Intel microcontroller family.

The EMUL196-PC SDU board is an 8 bit P.C. card that fits into any slot. The jumpers on the emulator board control three things: 1. the address used to communicate with the Host PC, 2. the

maximum communication rate of the target, and 3. whether or not power is provided to the target through the SDU connector. These are all described in more detail below.

Detailed Installation Instructions

Setting the I/O address jumpers -- J1

Note: The factory default is 0X200 for the software and hardware.

The EMUL196-PC SDU requires 8 consecutive I/O addresses from the P.C. 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 two pins represents a 0 in the address. A pair of pins with no jumper represents a 1. Below are 4 examples where the Least Significant Bit 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.

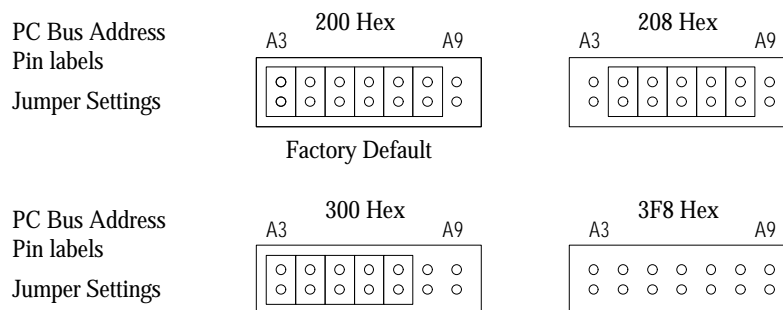


Figure 2: Emulator Header J1

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 3 to look up the clock rate in the lower row and place one jumper on header JP1 between the pins indicated in the upper row. There must be only ONE jumper on this header.

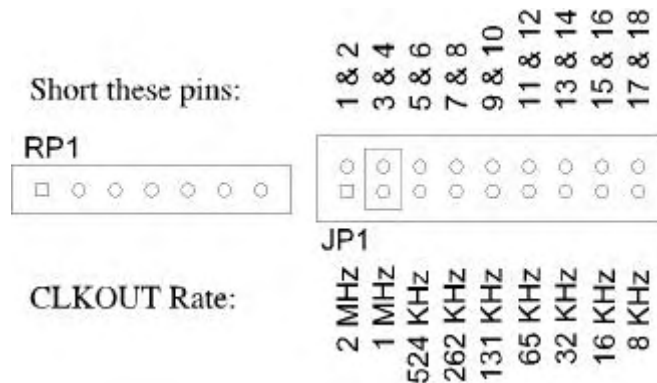


Figure 3: Header JP1

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

The PWR Header -- JP2

The third header on the low cost emulator board is the PWR header, which is also labeled JP2. With this jumper in place, +5 volts is supplied to the target from the P.C. power supply through the SDU connector. If the target has it's own power supply, remove this jumper. If you do not, the two supplies will be shorted together and because no two power supplies are identical, one supply may drive the other, drawing excessive current. The power supplies

themselves are not likely to be damaged, but the currents may exceed levels safe for the traces and components on the target or low cost emulator.

Note: *EMUL196-PC SDU has been designed to be compatible with 3 volt target designs **IF** the PWR header has no jumper. With that jumper removed, when the target's power supply is providing 3 volts to the target, no voltage adapters or other electronics are necessary to emulate 3 volt designs.*

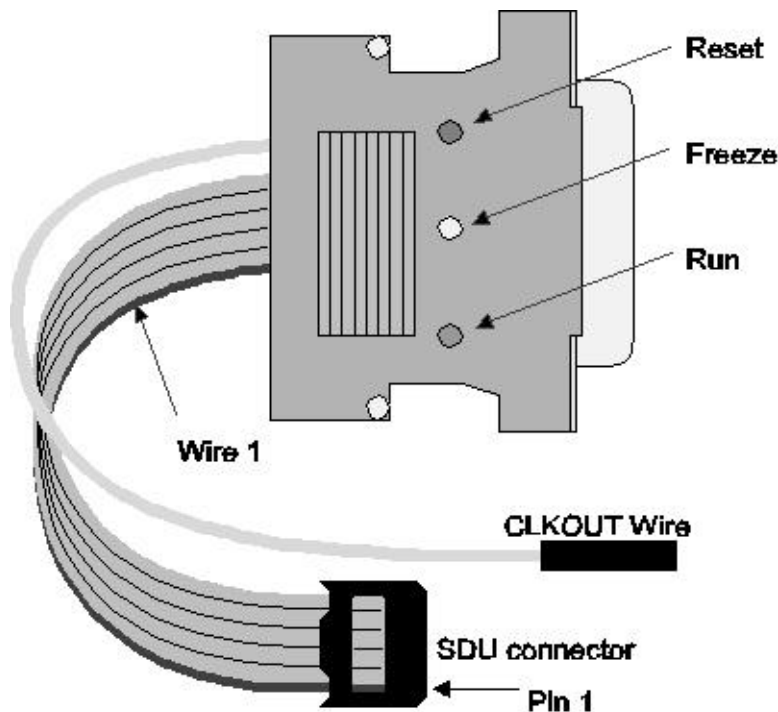


Figure 4: The POD

Chapter 2: 196EC FLASH Programmer

Disclaimer

The icon “196EC FLASH Programmer” uses an executable SDU_WIN.exe supplied by Intel. SDU_WIN.exe is not a supported product of Intel Corp. or ICE Technology. This program has never been tested and therefore the source files have been included. The source code for SDU_WIN is for free distribution and may be modified by users to their own liking. We cannot take any responsibility for losses or costs associated with code programmed by this software.

Introduction

This section describes how to use SDU_WIN.exe to program user code in the internal flash memory using the Serial Debug Unit (SDU) of the 88C196EC (EC) microcontroller. The software is designed to work with the SDU card made by ICE Technology. This card is the interface between the PC and the SDU port of the 88C196EC. The SDU card has the standard SDU connector and pin, making it targetable to the 88C196EC Evaluation board or to a user's own hardware. Hardware must pull the proper pins on the 88C196EC upon reset to enter SDU TROM mode.

A brief description of the Serial Debug Unit (SDU) is covered first. The next section discusses how to put the EC in the SDU Testmode required for operation with the SDU_WIN.exe. The remaining text covers how to invoke SDU_WIN.exe and an overview of how it works.

SDU Overview

The SDU provides high speed access to code RAM and on-chip RISM functions to provide program, debug, and application features. The SDU can directly read/write the code RAM of the

88C196EC without intruding on execution of the CPU.

Application code can therefore be running simultaneously while the SDU reads and writes code RAM. The SDU has four pins TDO (data out), TDI (data in), THS (handshaking), and TCLK (clock in). Commands and data are clocked into the SDU externally. The CPU, however, has priority over the SDU for accesses to code RAM. This means that if the CPU is using the code RAM, the SDU must wait.

The SDU signals when it is working by pulling the THS pin of the EC low. When the SDU has been able to complete the requested operation, the EC will drive the THS pin high. In addition to non-intrusive read/write operations on code RAM, the SDU also can make calls to RISM code by performing a series of writes to code RAM and triggering the SDU interrupt. This interrupt will stop the CPU and start executing the RISM interrupt service routine.

While RISM calls do intrude on the CPU (interrupt the CPU), they add several functions such as the ability to read/write any memory location (not just Code RAM) and the ability to start the CPU executing at a new location. When the EC is operating in a testmode (such as for SDU_WIN.exe), the on-chip RISM built into the EC is used.

Setting Pins for SDU Testmode Entry

Upon reset, certain pins on the EC are sampled to determine the mode of execution (normal, test mode, etc). SDU_WIN.exe requires the EC to be in the SDU trom mode. SDU TROM mode is entered if the following pins are pulled to the indicated logic level upon reset. After reset the testmode entry pins should be de-asserted by hardware to allow the EC to use RD#,WR#, etc. to function.

RD#	WR#	ALE	#BREQ	PLEN	TXD0	HLDA#	P12.2	P12.1	P12.0	P5.4	VPP
*1	*1	*0	*1	*1	*1	*1	*1	*1	*1	*1	+12V

- * *Denotes pin is weakly pulled to this state by the controller. Do not drive these pins during rising edge of reset or SDU testmode entry will fail. The only pin that has to be actively driven is: VPP = +12V*

Running the SDU_WIN.exe

Be sure to set the address of the SDU card to 200H. Attach the SDU connector with the red stripe connected to pin 1 of target header. Enter the SDU Trom mode by jumpering the VPP= +12V and press the reset button.

The simplest invocation of SDU_WIN.exe has the following command line:

Note: This should be done by modified properties of the “196EC FLASH Programmer” icon using the Windows “Files/Properties” command

SDU_WIN.exe filename.hex

where filename.hex is the file to be programmed (include the .hex extension in the name). You can include other command line options after the first specifier. For, example:

SDU_WIN.exe filename.hex -verify

will program the filename.hex and will, in addition to checking the flash status registers for program/erase errors, also physically verify each byte programmed to the flash. This is a bit slower but solidly confirms that your code got there.

The other command line options are listed below and can be added in any order (only filename.hex needs to be first). To see the table of command line options type: SDU_WIN.exe -h. When programming is done, remove +12V from VPP and reset your hardware to re-boot under normal execution.

Command line options:

- h: Print command line options and exit (no operation)
- v: Verbose mode. Prints each table to screen before programming
- verify: Physical verify each byte programmed to flash
- e: Skip erase
- eo: Erase only, no programming

Note: *hk32.hex and filename.hex should be located in the same directory.
(ie. C:\EMUL196\SDU)*

How it works

SDU_WIN.exe uses SDU commands to download the handler file (hk32.hex) to the code RAM of the EC. The SDU then calls the RISM GO command to start the handler code executing. The handler loops waiting for either of two tables to be marked FULL signaling data to be programmed or erased. Meanwhile, since the SDU also has access to the code RAM, SDU_WIN.exe reads in your code for programming. SDU_WIN.exe parses the code into tables and transmits them to the EA code RAM using the SDU.

When a table is in code RAM, SDU_WIN.exe marks it as FULL. The handler sees the FULL flag and programs the table the proper address in the flash. When it is done, it checks for program/erase errors and marks the table as EMPTY. While the handler is programming one table, SDU_WIN.exe is downloading another.

The exception is when the -verify command line specifier is used. When -verify is used, SDU_WIN.exe sends the table and waits until the table is programmed (a new table is not downloaded until the present table is programmed and verified). Then it uses SDU RISM calls to read each byte from the flash and compare them to the table.

Any program/erase errors are reported by SDU_WIN.exe including low Vpp, program error, erase error, failure to enter SDU TROM mode, THS handshake timeout, verify error, can't open to be programmed, can't open handler.hex, can't open config.cfg.

Programming for use with Nohau SDU Emulator Software

To use the Nohau SDU emulator in your hardware or on the eval board you must do three things:

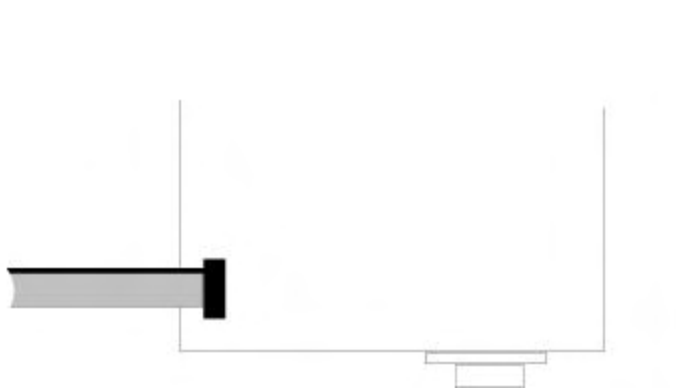
- a) enable the SDU interrupt and global interrupts (ei instruction) in your cstart.a96 file.
- b) alter the page of the risism code supplied by Nohau to match your memory map (i.e., if you use 180000H-1FFFFFFH you must change the CSEG's in the sdurism.a96 to: 080000H to 180000H etc. and recompile it).
- c) link the sdurism.obj from step b) in with cstart.boj and the rest of your code.

Use oh196 to convert the resulting file from object code to hex (say file.hex). Use SDU_WIN.exe to program this to the flash. Move PLLN=5v and hit reset. The part will boot cstart and jump to the RISM idle loop.

You should now be able to invoke the Nohau software on the PC and connect to the board over the SDU. In the Nohau software, set the PC to the beginning of your code.

Chapter 3: Intel Evaluation Board (196EA)

To get started with your Nohau SDU software, we suggest that after you have installed the emulator board and connected your SDU POD, you test the system with an Intel evaluation board as your target system. Please note that when you do your debugging, the Intel evaluation board will be replaced by your target.



To use the evaluation board, you must first do the Appendix A modification to prevent the Windows software reset from forcing the evaluation board to execute the RS 232 RISM code.

When using the Intel evaluation board, you will have to start the software (ecm96ea.exe) Intel delivered with the evaluation board and load the sdurism.omf file Nohau has supplied in the RISM directory. The code will be download through Intel's RS 232 cable.

1. Load SDU RISM.OMF
2. Set the PC = 0FF2080.
3. Type in the "GO" command at the prompt.

4. At this point, we suggest that you exit the Intel evaluation board software by using the “QUIT” command.

The SDU RISM code is now running in the evaluation board. Please review the first paragraph of Chapter 3 for the software installation.

You can now start the Nohau Windows based SDU software and be able to perform most of the commands that the Nohau debug environment provides.

Software Defects

The Stack Pointer must be initialized with two trailing zero's (i.e., for SP=400, you must enter 40000 into the Miscellaneous setup).

Please report any software defects in writing with a technical description of the problem to Nohau Technical Support for SDU196™: email: support@icetech.com.

Tutorial

Click in the Program Window. Type Control<A>, type in the value: 0FF20DF, and click “OK”.

Click on the “GO” button (the user code will run starting at address FF20DF).

Click on the “BREAK” button (the user code will stop running).

Click in the Program Window. The line will change color to indicate a breakpoint.

Hit the “GO” button and the user code will stop immediately. The cursor will now be pointing to the line where you put the breakpoint (the user code stopped at the correct location).

Debugging with Your Own Target

To run the Nohau SDU Windows based software with your own target connected, the following requirements will have to be met:

1. Your target must have some SDU RISM code resident.
2. Your target is constructed in such a way that you can download the code you want to debug into on-target RAM.
3. SDU interrupts are turned on.

Please see the included readme.txt file that you will find in the RISM subdirectory under the Nohau install directory. Please also review Appendix A and Appendix B.

Chapter 4: SDU 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 mastering MS *Windows*, please refer to the Microsoft *Windows* User's Guide.

The EMUL196™-PC / SDU floppy disk includes an MS Windows compatible SETUP.EXE program. To install this software, run SETUP.EXE by typing "WIN A:SETUP" at the DOS prompt before entering Windows or, from within MS Windows, by selecting the **RUN** item in the Program Manager **File** menu and typing "A:SETUP" as the file to run. A dialog box will ask for a directory for the EMUL196™-PC / SDU 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 change the various MS *Windows* ".ini" files as needed. When installed, there will be a **NOHAU** program group with an **EMUL196** icon.

Note: *The Confidence Test is not supported for the SDU. Before starting the EMUL196, first run the ini generator.*

Double-clicking on the emulator icon will start the EMUL196™-PC / SDU application. If you wish to move the icon to another group, you may do so by using the **Move...** menu item in the **Program Manager's File** menu or by dragging the icon to the new group.

Initial Software Configuration

The Windows software is used for all EMUL196™-PC / SDU 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 **Send Command** error message. To ensure that you do not get this error we include a utility that you probably want to run when you first install the emulator, and possibly again when you

change your pod type. This utility is called **INI196**. (You can also run this utility any time you want to check the values in the initialization file.)

To invoke **INI196**, double click on the icon in the **NOHAU** program group labeled INI196. If the EMUL196.INI settings are not self-consistent, you will see a warning message, otherwise you will see the window shown in Figure 5.

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

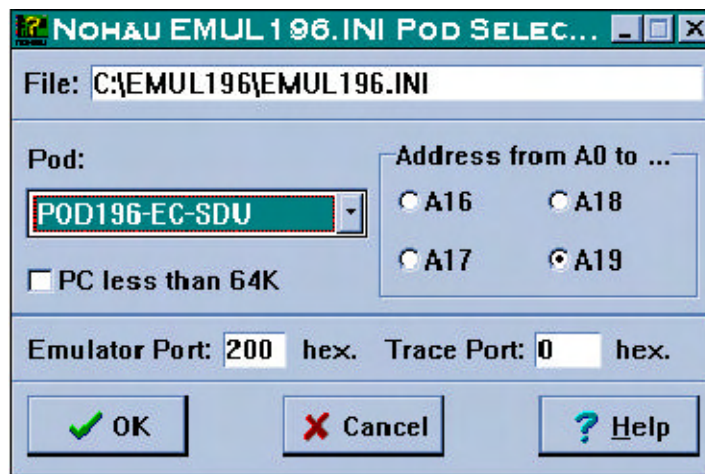


Figure 5: Choosing a target processor with INI196

After you have set the **Emulator Port:** and **Trace Port:** fields, click on the OK button or press <Enter>. The **Emulator Port:** field must agree with the values you set in the J2 jumper on the emulator board, as described in the section, “Setting the I/O address jumpers -- J1” on page 12. Likewise, the **Trace Port:** field should be set to zero.

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

Configuring the Software

If the Quick Installation instructions do not work, you will most likely need to adjust either the hardware jumpers, the software configuration, or possibly both. Please refer to the appropriate chapters for setting the jumpers on 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 if needed.

Projects

Config
Project name ..
Paths ..
Memory map ..
Emulator Hardware ..
Miscellaneous ..
Full Reset
Color ..
Trace ..
Fast_Br_W ..
Memory Coverage ..
PP Analyzer

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

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

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

Creating a Project

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

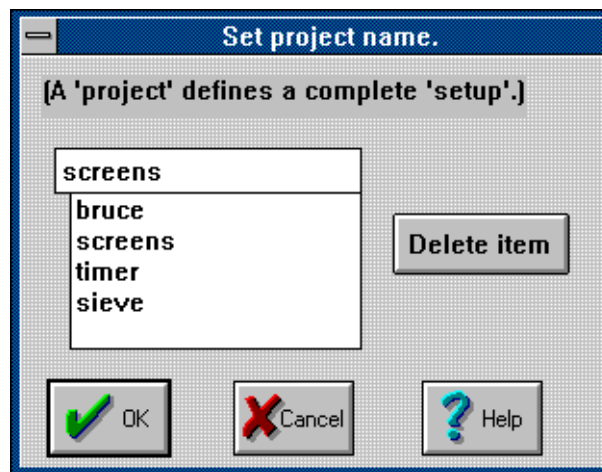


Figure 6: Set Project Name Dialog Box

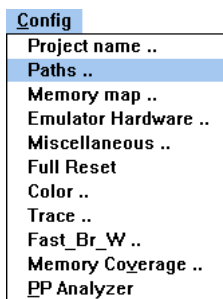
To add a project, type the new name over the current name. Because the project name is used as the body of a DOS file name, do not use characters in the name that cannot be used in a file name (like a space character). The new project will inherit all the settings from the old project. Projects are deleted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



Figure 7 : EMUL196™-PC / SDU Title Bar

The name of the current project appears in the emulator software title bar. Figure 7 is an example of the **EMUL196** title bar for the project named **SCREENS** (used to create the screen shots for this manual).

Setting the Paths ..



The next item in the **Config** menu is **Paths ..** which opens the dialog box shown in “Figure 8: Paths Dialog Box,” on page 22. The emulator uses these directories to find the files it needs.

Each of these fields can hold up to 1024 characters. Each directory in the path must be separated by a semi-colon (;) just like MS DOS path names. By default, The **User load modules:** field will contain the directory from the last loaded object file, and the **Emulator internal files:** field will contain the directory where the emulator files were installed.

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

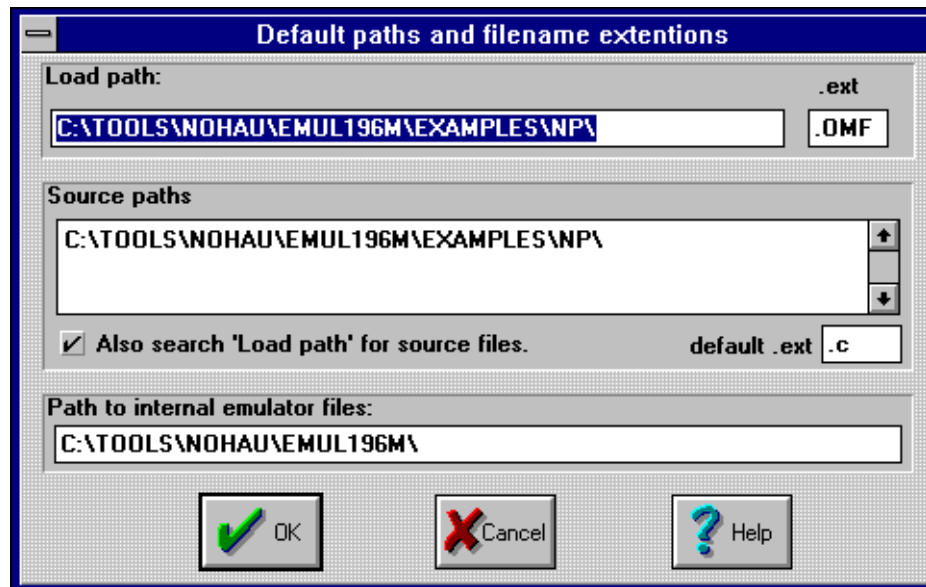


Figure 8: Paths Dialog Box

With many compilers, the full path name of the source file is contained within the object file. Linked object files consisting of several linked objects will, correspondingly, have several source file names and paths. If that source file name exists in the object file that EMUL196™-PC / SDU is loading, the debugger will look for that source file when updating the **Source** window.

The second field, **Source paths**: identifies other directories to search for missing source files not identified in the object file or files moved since the compile. The directories in this field must be entered by the user. Once entered, directories will stay here until removed by the user. The small check box, when checked, will tell EMUL196™-PC / SDU to look for source files in the **Load path**: directory as well. Simple projects may have all the source and object files in the same directory (the **Load path**:) and may not need any directories in the **Source paths**: field.

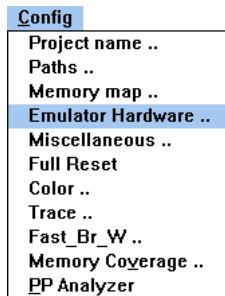
Note: The “.ext” field specifies the source file extension. If your C modules have the extension “.c”, enter that. To see assembler source (.asm) in the Source window, enter “.asm”.

The **Emulator internal files:** field will be set during the installation and probably will not need to be changed. **Emulator internal files:** is the directory the application uses to find the various support files that are part of the EMUL196™-PC / SDU 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 EMUL196™-PC / SDU to a new directory or disk drive, remember to change this field also.

Mapping memory

This feature does not apply to SDU.

Emulator Hardware Configuration



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

The **Emulator Port** value must agree with the jumper settings on the emulator board header J2. (See “Setting the I/O address jumpers -- J1” on page 12). If they do not agree, the EMUL196™-PC / SDU software will not be able to communicate with the hardware, and the dialog box in Figure 9 will automatically be displayed, as a reminder that communication has failed and some change is needed.

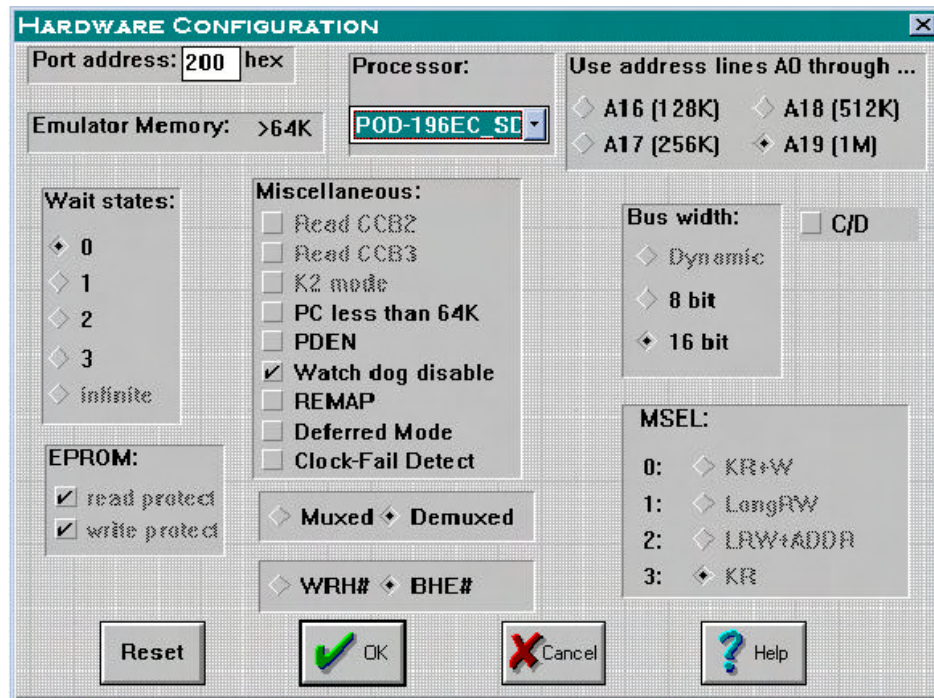


Figure 9: Hardware Configuration Dialog Box

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, EMUL196™-PC / SDU will not work properly.

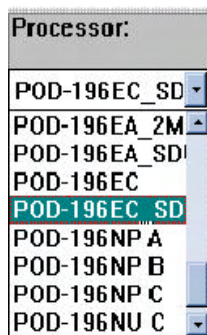


Figure 10: Processor List Box

The processor list box lets you choose the kind of processor being emulated. This setting must agree with the kind of processor you are using in the target or the emulator will not work correctly. If you must change the processor type, exit the program and run ini196.exe.



Figure 11: Wait States field

The box labeled **Wait States:** will let you select the number of wait states the chip will use when accessing external memory. Emulation RAM is fast enough to respond with 0 wait states. Target RAM may or may not be fast enough, depending upon the speed of

the chips on your target. Selecting the **infinite** button requires that the target hardware correctly assert the **READY** signal when the data on the bus is valid.

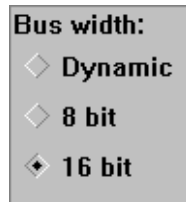


Figure 12: Setting the Bus Width

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

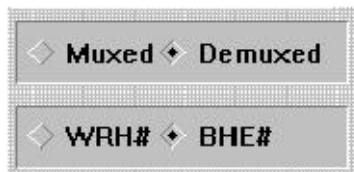


Figure 13: Bus Control Settings

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

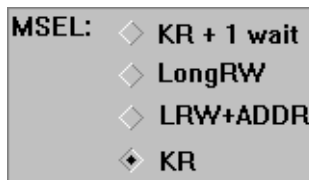


Figure 14: Mode Select Field

This field is not functional on the SDU.

Miscellaneous bits

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

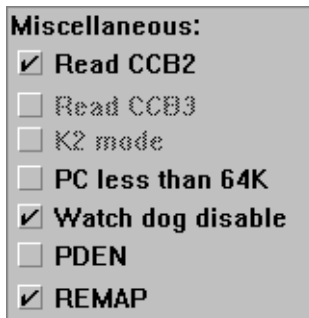


Figure 15: Miscellaneous CCB bits

The **K2 mode** check box is not used at this time.

When the **REMAP** box is checked, the on-chip program memory¹ (in the OTP/EPROM) will appear both in page 0 and page FF.

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

The check box marked **Watch dog disable** controls when the watchdog timer starts. If this bit is cleared, the watchdog timer will start immediately after the controller is reset. If this bit is set, the watchdog timer will not start until the watchdog is reset. To reset the watchdog timer, write the values 1EH followed by E1H to the watchdog register at address 0AH. Once the watchdog timer is running, the only way to stop it is to reset the controller.

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

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



Figure 16: Exiting the Hardware Config dialog box

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

1. Note that this memory may be physically on the controller or it may be in emulation RAM, depending upon how the memory is mapped.

Note: When the EMUL196™-PC / SDU software is started, the controller will be released from a reset state and it will read the CCB values from locations in the POD EPROM and execute code that writes the .ini file CCB values to the CCB RAM locations. Then the controller is reset again and this time it will read the CCB values from RAM.

Miscellaneous Configuration

Config

Project name ..

Paths ..

Memory map ..

Emulator Hardware ..

Miscellaneous ..

Full Reset

Color ..

Trace ..

Fast_Br_W ..

Memory Coverage ..

PP Analyzer

The **Miscellaneous** item in the **Config** menu opens a dialog box that controls special features of EMUL196™-PC / SDU:

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

By default, the emulator resets the controller when the EMUL196™-PC / SDU software is started and after an object file is loaded. The **Reset chip at start up:** and the **Reset chip after load file:** radio buttons can disable either of those resets which may be helpful during particularly difficult or unusual debugging circumstances.



Figure 17: Miscellaneous Setup Dialog Box

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

Even though the next two fields appear to work, they are not implemented yet. Watchpoints are not yet saved and writes to memory are always read back. For information about the status of these features, email support@icetech.com, customer support.




Figure 18: Two Features Not Yet Implemented

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

There is a lower limit to how often the screen and the DDE link can be updated. This limit depends upon the speed of your machine, how much RAM your machine has, and how many applications are running. Due to delays from inter-application messaging in *MS*

Windows and possible problems caused by a message backlog, the lowest setting allowed is 100 milliseconds. The upper limit is 32767 milliseconds.



The screenshot shows a configuration window with a grey background and blue borders. It contains two input fields. The first field is labeled "DDE sampling interval (ms):" and has the value "250" entered. The second field is preceded by a checkbox and is labeled "DDE poke enable. Flag at address:". The checkbox is currently unchecked, and the value "0" is entered in the field.

DDE sampling interval (ms):	250
<input type="checkbox"/> DDE poke enable. Flag at address:	0

Figure 19: Controlling the DDE Sample Interval

Occasionally, while the emulation is running, writing a value to a RAM location can help debugging. For example, it might be useful to simulate a memory mapped input device this way. The EMUL196™-PC / SDU DDE interface supports "poking" a value into an address. The DDE link can be a two-way connection and some applications may send a value to a particular address in emulation or target RAM. However, updating RAM at pseudo-random times can be dangerous for the program and possibly the hardware you are controlling while you are emulating. With the next field, the **DDE poke flag address**, you specify an address of a two byte flag that, when polled and found to be set to hexadecimal 1234, indicates that it is safe to write the DDE value to RAM.

The check box to the left turns poking on or off: a check turns it on. When EMUL196™-PC / SDU has a value to "poke" into RAM (every **DDE sampling interval**), it first polls this box. If it is checked, the emulator application will read the contents of the poke flag address. If the contents are set to 1234H, then EMUL196™-PC / SDU will suspend the emulation for approximately 200-250 microseconds while it updates RAM. Finally, EMUL196™-PC / SDU will clear the poke flag (set to 0) and restart the application. The software must reset the flag to 1234 when it is again safe to update the poke addresses.

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

Enable Code Space Limits

In some symbols files, symbols are not identified as Program space variables or as Data space variables. If this is true of your file, you may see the EMUL196™-PC / SDU software try to set breakpoints in your variable address range. To prevent this, check the **Enable code space limits** box and set the **Low** and **High** addresses to encompass just the instructions. Configured this way, EMUL196™-PC / SDU will not put breakpoints outside that address range.

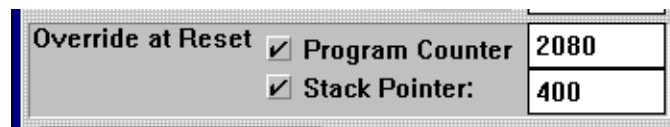


Figure 20: Forcing Reset Vectors

When the **Override at Reset** boxes are checked and the fields contain addresses (in hexadecimal notation), those values will be written to the controller's program counter and stack pointer every time EMUL196™-PC / SDU resets the controller. If you have some test code at an address other than 2080H that you want to execute, filling in this field will force the program counter to the specified value each time you reset the controller. Similarly, to run the test code right after a reset, the stack pointer register must have a legitimate value. This field will conveniently force the stack pointer to a specified value after reset without having to run your start-up code.

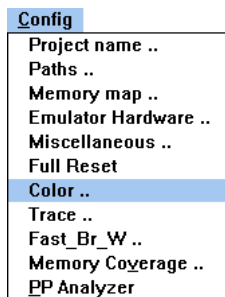
In the lower left corner of the **Miscellaneous Setup** dialog box is a group of radio buttons that control the **Memory scroll range**. These buttons specify the highest address displayed in **Program** and **Data** windows when the elevator in the scroll bar is dragged to the bottom of the scroll bar. If the scroll range is set to 256K, the

bottom of the scroll bar represents the address 40000 Hex.
Similarly, halfway down the scroll bar represents 20000 Hex. If the scroll range is set to 1 Meg. the bottom represents 100000 Hex, and so on. .

Reset vs. Full Reset

Full Reset is not functional on the SDU.

Window Colors



Under the **Config** menu is the **Color ..** menu item. Open this dialog box to set the colors of different kinds of EMUL196™-PC / SDU child windows. For example, all **Program** windows can be set to have a dark blue background with white text to differentiate them from other kinds of windows. At the same time, all **Data** windows can be green with Black text, and all **Source** windows set to have white background and red text. It is possible to make the screen quite attractive.

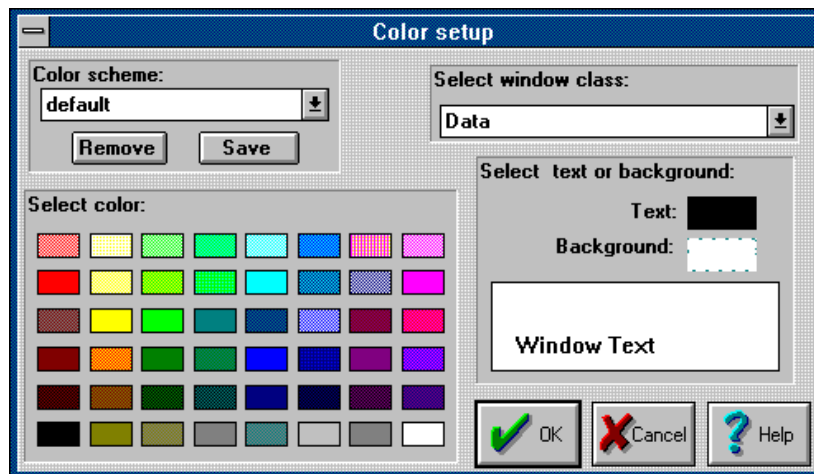


Figure 21: The Color Setup Dialog Box

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

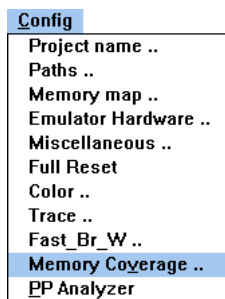
After you have set all the colors the way you want them, you can name a color scheme by typing the name in the **Color scheme** field and then click on the **Save** button. This color scheme can then be recalled by selecting it from the drop list of color schemes.

***Note:** Not all combinations of background and foreground colors are possible. EMUL196™-PC / SDU is constrained by the same limits as MS Windows itself, and is affected by the color palette chosen in the Windows Control Panel program. No matter what colors you select from this palette, the example text pane will show you the colors that will actually be used by EMUL196™-PC / SDU. Trace Config Menu*

Fast Break Write

Fast Break Write is not functional on the SDU.

Memory Coverage



The EMUL196™-PC / SDU trace option includes the hardware necessary to monitor memory and correlate its use with your C source code. If instructions are fetched, the trace board will mark those addresses.

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

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

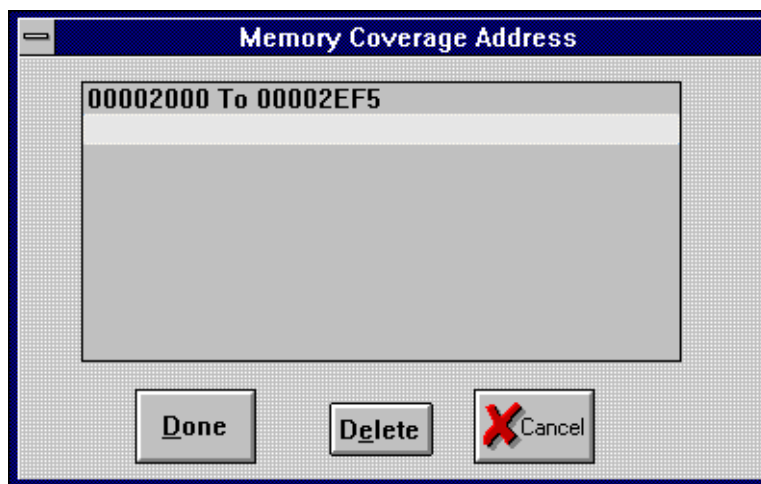


Figure 22: Editing the Coverage Address Range

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

EMUL196™-PC / SDU supports any number of address ranges. The ranges do not need to be next to each other. They may be located anywhere in the address space. The only practical limit is that the sum of all ranges must be less than 256 kilobytes for the 32K and 128K trace boards, and less than 1 megabyte for the 512K trace board.

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

Each square represents a memory word: two bytes starting on an even byte. Squares are grouped into segments 8 squares across. For each address there are 4 rows of segments. Figure 23 shows a **Memory Coverage** window with 7 segments in each row. In this case, each row of dots represents 70 hex bytes of memory. Each row of blocks represents 1C0 hex bytes. As the window gets wider, each row contains more blocks of squares and the the addresses will get farther apart.

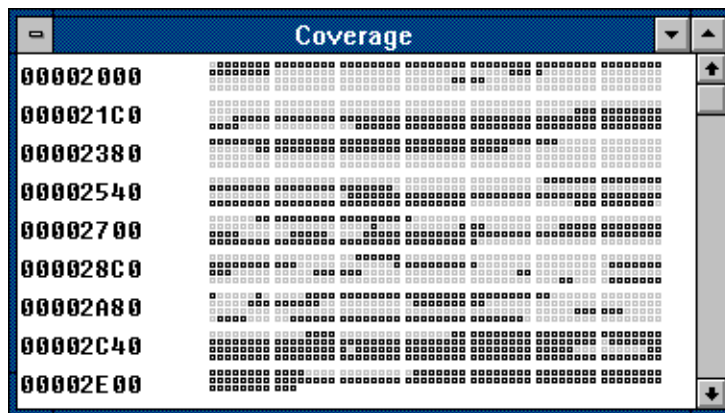


Figure 23: Typical Memory Coverage Window

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

As shown in Figure 24, there are new symbols in the **Source** and **Program** windows. In the **Program** window, you will find either a **:** or an **x** between the address and the rest of hexadecimal value at that address. The **:** means that the opcode has not been executed. The **x** indicates that it was either executed or prefetched.

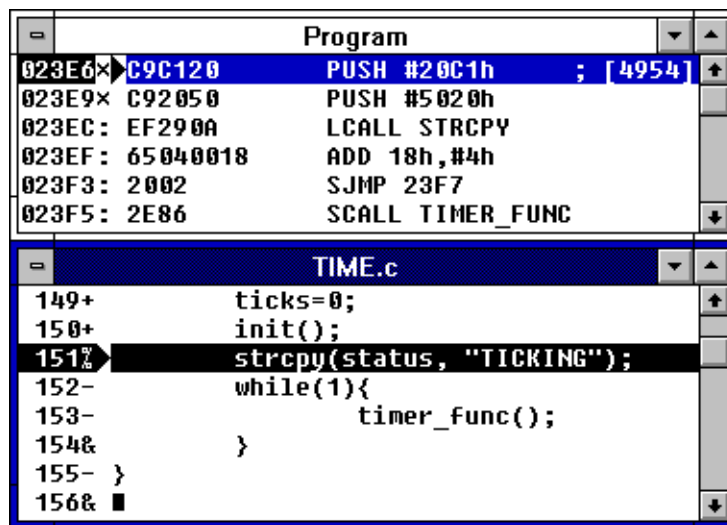


Figure 24: **Program** and **Source** Windows in Coverage Mode

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

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

Please note two important things in Figure 24. The **strcpy()** call is not completely covered, so in the **Source** window, it is marked with a percent sign. The other important thing to notice is that address **23E6** has a breakpoint. The instruction at address **23E9** has been marked as fetched (because it has been fetched) but it has NOT been executed. Instructions right after executed jumps will be shown as fetched. They may or may not have been executed.

Summary Memory Coverage Report

A screen full of coverage information may be helpful, but it won't satisfy the FDA. or the FAA. They both want written evidence they can hold in their hands that show that your tests actually tested your code. EMUL196™-PC / SDU can help there.

When you select **Report** from the **Coverage** menu, you will see a dialog box similar to Figure 25.

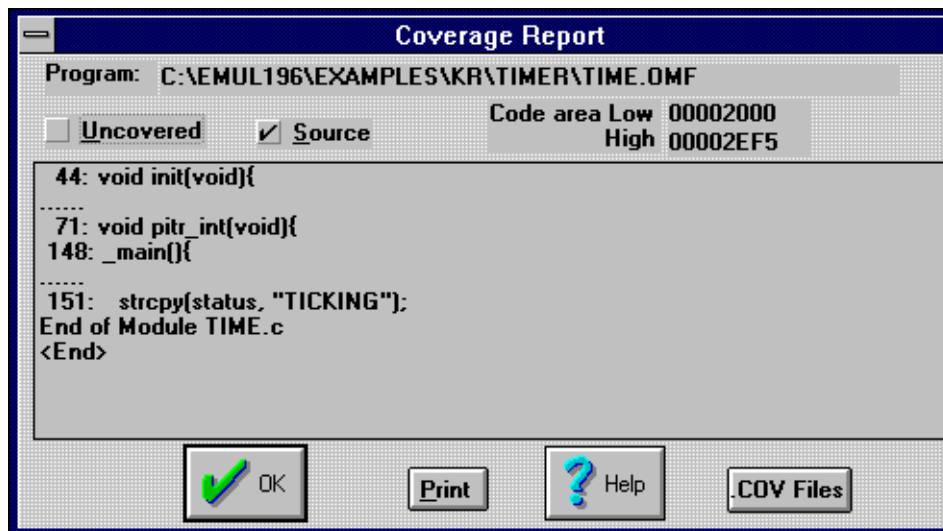


Figure 25: Summary Coverage Report

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

To obtain a paper copy of the report, click on the **Print** button. This will send the summary to the current default printer. A different kind of report, a more detailed report, is available by clicking on the **.COV Files** button. Figure 26 is the dialog box that configures these reports.

The summary report is good for small test runs that can be completed without turning off your P.C.; without exiting the EMUL196™-PC / SDU software. However, your tests may be very large; so large that running all of them without exiting the emulator software may not be possible. The detailed coverage reports let you combine multiple test runs in a single document (for each source file).

Detailed Memory Coverage Report

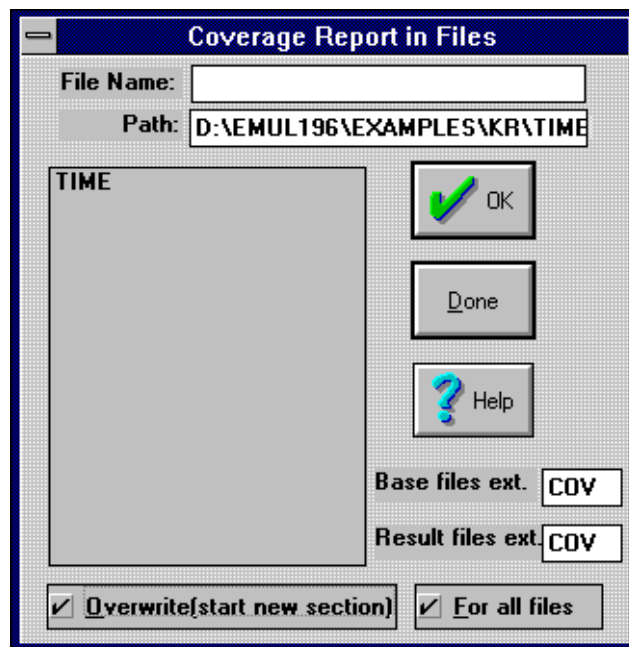


Figure 26: Producing Detailed Coverage Reports

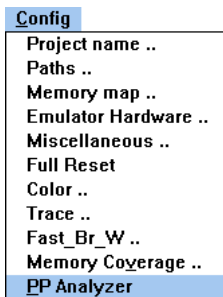
The detailed coverage report produces a text file that looks like the **Source** window while in coverage mode. It reads in the source file (or a previous coverage file), and puts the line number and one of the four status characters at the beginning of each line. The output text file is written to the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

If the source file scanned is actually the output from a previous coverage report, it will combine the two reports so that lines covered by either report will be marked as covered in the new report. This is the feature that allows you to run your tests over

several days and still generate a single set of files that accurately reflects how well all of the tests together have covered the generated instructions.

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

Performance Analysis¹



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

Performance Analysis is a statistical analysis of execution behavior. Once every second, a percentage of the bus cycles are collected, sorted into their respective bins, and the results are displayed on the screen. As you might guess, the percentage of the cycles that are collected depends upon the speed of your P.C., what other tasks are running, etc.

1. Later software versions will have the Performance Analysis item under the Trace Menu.

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.

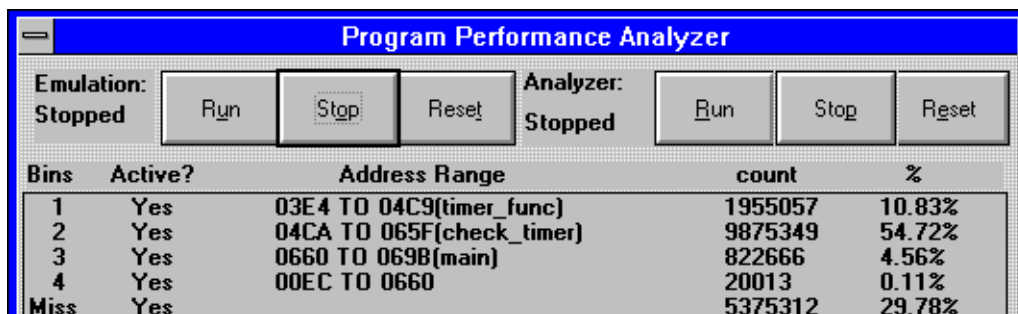


Figure 27: Performance Analysis Control Window

When you select **PPA Analyzer** from the **Config** menu, you will see a control window that looks like Figure 27. The application and the data collection will automatically be started every time you open the PPA control window. The six buttons at the top of the window control the application and the data collection separately.

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

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

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

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

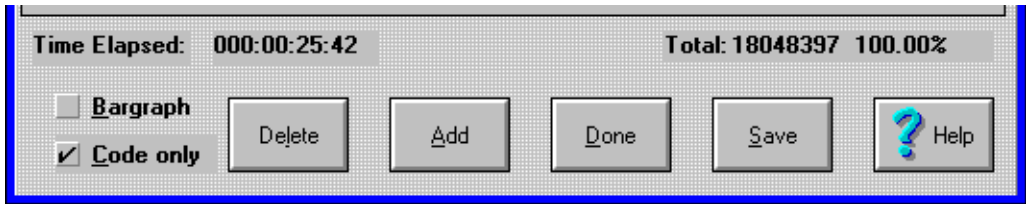


Figure 28: Performance Analysis Control Options

Figure 28 shows the rest of the statistics from Figure 27, plus a few options. This data set took 25 minutes and 42 seconds to collect. A total of 18,048,397 frames were recorded. Figure 28 shows that data read and write cycles were ignored; the **Code only** option is checked. Only instruction fetches are counted in the statistics.

Figure 28 also shows that the **Bargraph** option is turned off. If you prefer a graphic display, you may turn on the **Bargraph** option is turned offbar graph and see the data displayed in a form similar to that shown in Figure 29.

Bins	Active?	Address Range	count	%
1	Yes		798964	4.68%
2	Yes	[check_timer]	11325917	66.34%
3	Yes	[timer_func]	4156092	24.34%
Miss	Yes		792162	4.64%

Figure 29: Bargraph Display Option

To add bins of your own, click on the **Add** button to open the list of functions shown in Figure 30. You can add any of the functions as an address range, or you may create an address range not on the list.

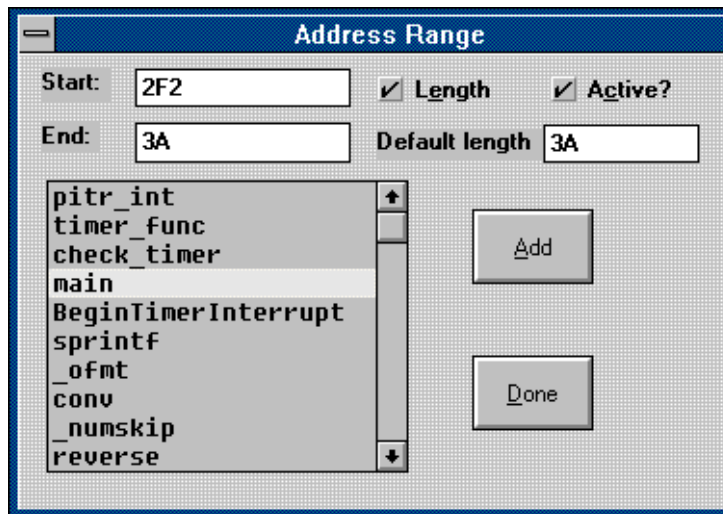


Figure 30: Adding a Bin

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

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

Using a very similar screen, any bin can be edited by double clicking on that line in the PPA Control window (Figure 27). This is how you activate and deactivate bins.

Once you have collected the data you want, EMUL196™-PC / SDU allows you to either save or discard the changes you just made to the list of bins. Only one bin configuration can be saved, not one per project like most configuration settings. This bin configuration will be automatically restored the next time you use performance analysis.

Menus

The primary means of controlling the debugger, thus the emulation, is through menus. The EMUL196™-PC / SDU menus conform completely with the Microsoft MDI standard. Only those menu items that have meaning or can be used with the current selection will highlight when the mouse is pointing to them. Menus are organized to hide items that are out of context.

Most menu items have "Hot Key" equivalents. That is, there is some combination of function keys, character keys, and modifier keys (Control, Shift, or Alt keys) to select most menu items. The Hot Key for each menu item is shown in that menu to the right of the item name, and are also shown below. Where you see "<Alt>FS" as the keyboard shortcut, you should type <Alt>F (hold the Alt key down while you then press the F key) to open the **File** menu, then press the S key (without the Alt key) to activate the portion of EMUL196™-PC / SDU 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 ..

Menu Item	Hot Key	Function
Load code ..	F3	Load an absolute file.

EMUL196™-PC / SDU supports many popular compiler object file formats. Load default symbols ..

Menu Item	Hot Key	Function
Load default symbols ..	<Alt>FL	Load symbols defined by the MCU manufacturer.

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

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

Save code as ..

Menu Item	Hot Key	Function
Save code as ..	<Alt>FS	Write the contents of RAM or ROM to a HEX record file.

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

Remove Symbols

Menu Item	Hot Key	Function
Remove Symbols	<Alt>FR	Delete all line number and symbolic information.

Show load info ..

Menu Item	Hot Key	Function
Show Load info	<Alt>FS	Display a window describing the object file last loaded including number of variables, address range loaded, etc.

This menu item opens an information box like the one in Figure 31.

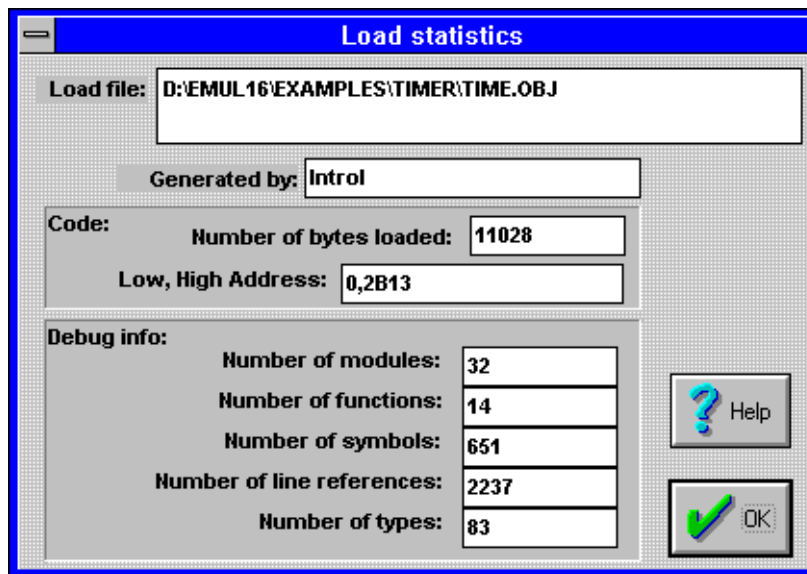


Figure 31: Example Load Statistics

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

Menu Item	Hot Key	Function
Preferences	<Alt>FP	Controls the way the emulator loads object files.

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

Exit

Menu Item	Hot Key	Function
Exit	<Alt>X	Quit the EMUL196™-PC / SDU application.

Exiting the EMUL196™-PC / SDU software will update the current debugger configuration to either the .ini file or to the current .pro file, if one is selected.

View/Edit Menu

Copy to clipboard

<Ctrl><Ins>

Copy the text (without formatting or font information) of the entire active window to the clipboard.

User defined symbols	<Alt>VS	This item opens a dialog box that lets you select the module from which you can view symbols.
Default CPU symbols	<Alt>VC	View and edit memory-mapped registers by name and by the bit.
DDE Status	<Alt>VD	Open a window displaying the DDE interface status.

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

Please email support@icetech.com technical support engineer for assistance using the **DDE Status** window.

C call stack ..	<Ctrl>SS	Opens a child window that displays the C call stack and passed parameters needed to reach the current Program Counter.
Evaluate ..	<Ctrl>E	Open a dialog box that evaluates C expressions. Expressions may contain variables. Assignment expressions may change the values of variables.

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

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

Search..	<Ctrl>S	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. In all other windows that support searching, the search is for a hex pattern.
Search next	<Ctrl>X	The last search defined will be performed again, from the cursor forward.
Search previous	<Ctrl>P	The last search defined will be performed again from the cursor backwards.

Run Menu

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

Go	F9	Begin executing instructions from the current PC at full speed until the next breakpoint.
Go to cursor	F4	Execute the instructions from the PC to the current cursor position.
Go to ..	<Ctrl>F9	Execute the instructions from the PC to the specified address.
Go to return address	<Alt>F9	Execute the instructions from the PC to the next found function return. Due to certain optimizations, this feature may not always be available.
Go FOREVER	<Alt>RF	Execute instructions from the current PC after disabling all breakpoints.
Break Emulation	F9	Suspend execution as if a breakpoint was encountered.
Reset Chip and Break	<Ctrl>F2	Reset CPU without executing any instructions.
Reset Chip and Go	<Alt>RRR	Reset CPU and begin execution from reset vector.

Breakpoints Menu

Toggle	F2	Disable or enable existing breakpoints.
Hardware breakpoints ..	<Alt>BH	Opens a dialog box that lets you set up address ranges for hardware breakpoints (that don't use the trace board).
Break on internal access ..	<Alt>BB	Not functional on the SDU.

This page left deliberately blank.

Hardware breaks only	<Alt>BHH	If this menu item is checked, all program/source window breakpoints will result in hardware breakpoints..
At ..	<Alt>F2	Set a breakpoint by address, line, or line in module.
Setup ..	<Alt>BS	Open a breakpoint editing dialog box.
Disable all	<Alt>BI	Disable all breakpoints from being active while remaining in the list.
Delete All	<Alt>BD	Clear all existing breakpoints.
Break now!	<Ctrl>C	Immediately halt the emulation.

Config Menu

Project name ..		Choose a configuration or project from a list of existing projects, or create a new one.
Paths ..	<Alt>CP	Sets the default directories for finding load files, source files, and emulator files.
Memory map ..	<Alt>CM	Assign memory to either emulation RAM or the target.
Emulator Hardware ..	<Alt>CE	Sets the emulator board address, controller type, and Chip Select registers reset values.
Miscellaneous ..	<Alt>CM	Sets automatic PC & SP reset value, DDE sampling interval, and memory scroll range values.
Full Reset		Reloads on-pod logic & performs reset.
Color ..	<Alt>CC	Assign colors to windows.
Trace ..	<Alt>CT	
Fast_Br_W ..	<Alt>CFF	Setup and execute a “fast break write” to memory.
Memory Coverage ..	<Alt>CV	Open the dialog box that controls Memory or Code coverage.
PP Analyzer	<Alt>CPP	Open a Performance Analysis control window and start recording addresses.

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

Program Menu

Address..	<Ctrl>A	Scroll the selected Program window to the specified address.
------------------	---------	---

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

Source Menu

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

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

Data Menu

Address..	<Ctrl>A	Scroll the selected Data window to the specified address.
Original Address	<Ctrl>O	Scroll the selected Data window to the last address used in an Address.. menu command.
Edit ..	<Enter>	Alter the contents of the highlighted location.
Block move..	<Ctrl>B	Move a segment of RAM to another location (in RAM).
Fill..	<Ctrl>F	Fill RAM with the specified value or pattern.
DataDisplay as..	<Ctrl>D	Set the data display mode (ASCII, hexadecimal bytes, long integers, etc.).
Address space ..	<Alt>DA	Set the address space for the selected Data window.

ShadowRam Menu

Address..	<Ctrl>A	Scroll the selected Shadow-Ram window to the specified address.
Original Address	<Ctrl>O	Scroll the selected Shadow Ram window to the last address used in an Address.. menu command.
Original address Display as..	<Ctrl-O> <Ctrl>D	Set the data display mode (ASCII, hexadecimal bytes, long integers, etc.).

Register Menu

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

Stack Menu

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

Watch Menu

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

Window Menu

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

Open a new program window	<Alt>IO	Open a new Program window.
Open a new source code window	<Alt>IO	Open a Source window.
Open a new data window	<Ctrl>M	Open a Data window.
Open a new register window	<Alt>IO	If one is open, it will ask "Are you sure?"
Open a new shadow ram window	<Alt>IO	Open a new ShadowRam window.
Open a window for RTXC		Open a window that displays the debugging information from RTXC.
Toggle help line	<Alt>IT	Turn on or off the text at the bottom of the EMUL196™-PC / SDU window.
Refresh	<Ctrl>R	Repaints the screen.
Tile windows	<Alt>IT	Resize and arrange the windows within the EMUL196™-PC / SDU application.
Cascade windows	<Alt>IC	Resize and overlap the windows within the EMUL196™-PC / SDU application.
Arrange Icons	<Alt>IA	Line up any closed EMUL196™-PC / SDU icons at the bottom of the main window.
Zoom	F5	Expand the selected window to fill the EMUL196™-PC / SDU window.
Next window	<Ctrl>F6	Change the currently selected (highlighted) window.
Close	<Alt>F4	Close the currently selected window.

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

Help Menu

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

Dialog Boxes

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

Child Windows

There are nine primary child windows created by EMUL196™-PC / SDU: **Program** windows, **Data** or Memory windows, **Inspect** windows, **ShadowRam** windows, **Source** windows, a **Registers** window, a **SpecialRegs** window, **Call Stack** window, **Watch** windows, and **Trace** windows (even if you have no Trace board). All of these windows are opened by selecting the corresponding item in the **Window** menu.

Any number of child windows may be open at the same time. Any number of child windows can overlap but only one child window is active (has the focus) at a time. They may be scrolled and resized to view any address desired. Their locations and sizes are saved to the current project file when EMUL196™-PC / SDU exits, and will be restored when the software restarts.

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

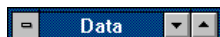
Register Windows



The **Registers** window displays the CPU registers. All registers are displayed in hexadecimal notation. Clicking anywhere in the **Registers** window will select that window (make it the active window) and right-clicking brings up the **Registers** menu. The only operation supported in the **Registers** window is editing register contents. The menu also lets you turn on or off, a display mode that, whenever the window is updated, compares the current values with the last values displayed and highlights (displays in a different color) the registers that have changed.

Note: *Editing the memory locations that represent the Stack Pointer, Imask, and WSR in a Data window will not update the Register Window display or the target CPU registers. Always use the Register window to edit these registers. During 1996, the software will be revised to handle the Register and Data windows transparently.*

Data and Shadow RAM Windows



Use **Data** windows to examine or modify emulation or target memory directly. EMUL196™-PC / SDU uses the controller to read and write RAM, so the **Data** window cannot be updated while the emulation is running. Instead, asterisks will be displayed until the next time the controller starts executing monitor code.

Data can be displayed or modified in various formats:

Note: 32 and 64 bit IEEE_754 floating point numbers must be word aligned. Some compilers support packed structures that can have floating point fields that start on an odd address. These fields will not be displayed properly in a Data window.

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

Changing a value at any memory location is as easy as selecting the byte, word, or long word to change and then typing the new value.

The first character you type will open a small data entry window, shown in Figure 32.¹

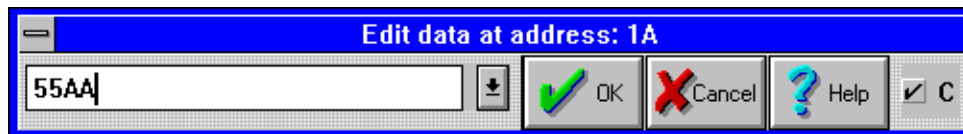


Figure 32: Editing Memory with a Data Window

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

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

1. Editing in a window displaying memory as ASCII characters will not open the Edit data dialog box. The new character will simply replace the one highlighted.

On the far right side of the Edit data dialog box is a small check box labeled with the letter **C**. This check box impacts how the emulator interprets the data you enter. If you have a symbol named “abcdef” and you are displaying in 16 bit hex, it is not clear whether to interpret “abcdef” as a symbol name or as a hex number. With the box checked, the emulator uses C syntax first, so it will be treated as a symbol name. Without the **C** box checked, assembler rules apply first, and it will be interpreted as a hex number (see far right edge of Figure 32).

Data windows can be assigned to read from and write to either code space or data space. Data space/code space are the same until the INST pin is used. This feature is only partially implemented at this time. Below 2000H, a **Data** window assigned to data space will display and modify on-chip memory and a data window assigned to code space will interact with the off-chip memory. Above 2000H, all windows will read from the same memory, no matter how they are assigned.

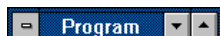
Custom Display Format

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

ShadowRAM Windows.

ShadowRam is not functional on the SDU.

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 92 for

information about how to change the color settings.) Use the cursor to set and disable breakpoints, set the program counter, and invoke the in-line assembler.

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

A comment will sometimes appear to the right of the highlighted instruction. The comment displayed is a function of the kind of instruction and is a hint about what will happen when the instruction is executed. For example, if the highlighted instruction will change the contents of memory, the hint will contain the value about to be overwritten.

Program windows can control the emulation. To set a breakpoint, click once on the address portion of the instruction where you want the break. Or, you may click once on the desired instruction (to highlight that instruction) and then click on it again to highlight the address. A breakpoint is indicated by displaying the address with white letters on a black or dark background¹. This second mouse click (not a double click) creates the breakpoint. To deactivate (not delete) that breakpoint, click again on the same instruction. The address will no longer be highlighted and the breakpoint will be inactive. To delete the breakpoint, use the **Setup ..** dialog box from the **Breakpoints** menu. Any highlighted instruction can be a temporary breakpoint. The **Run** menu item **Go until cursor** will use the cursor as a temporary breakpoint.

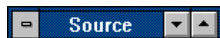
1. The colors used to indicate a breakpoint may be different if you have changed the color scheme as described in the next section.

In-line Assembler

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

The in-line assembler will translate the input line according to the syntax described in the 80C196 data books and replace the former opcode(s) and data with the new opcode(s) and data. Note that the assembler will write as many bytes as required for the new instruction. This may overwrite part or all of subsequent instructions. Be sure to examine the subsequent instructions as well as the new instructions for correctness.

Source Windows



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

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

Displaying and toggling breakpoints in **Source** windows is different than in **Program** windows. In **Source** windows, breakpoints are displayed by inverting (or highlighting) the entire source line. In **Program** windows, only the address is highlighted. In **Source** windows, a single click on any line number (or address

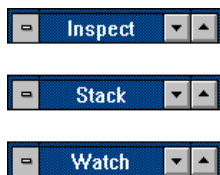
in the Program window) will toggle the breakpoint. In both kinds of windows, pressing F2 will toggle a breakpoint on the highlighted instruction.

When a **Source** window appears blank with the window title "Source", it usually means that the program counter is pointing to instructions derived from a module with no debugging information. As soon as the PC points to an instruction from a C module or assembly module with line number symbols, the **Source** window will show that text, and the title on the window will change from "Source" to the name of the source file being displayed.

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

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

Other Windows



Three more child windows used for high level debugging in C are available: the **C call stack** window, the **Evaluate** window, the **Inspect** window, and the **Watch** window. These windows are opened by selecting their respective items in the **View/Edit** menu.

Like the other child windows, selecting one of these open windows will bring a corresponding menu up between the **Config** and **Window** menus.

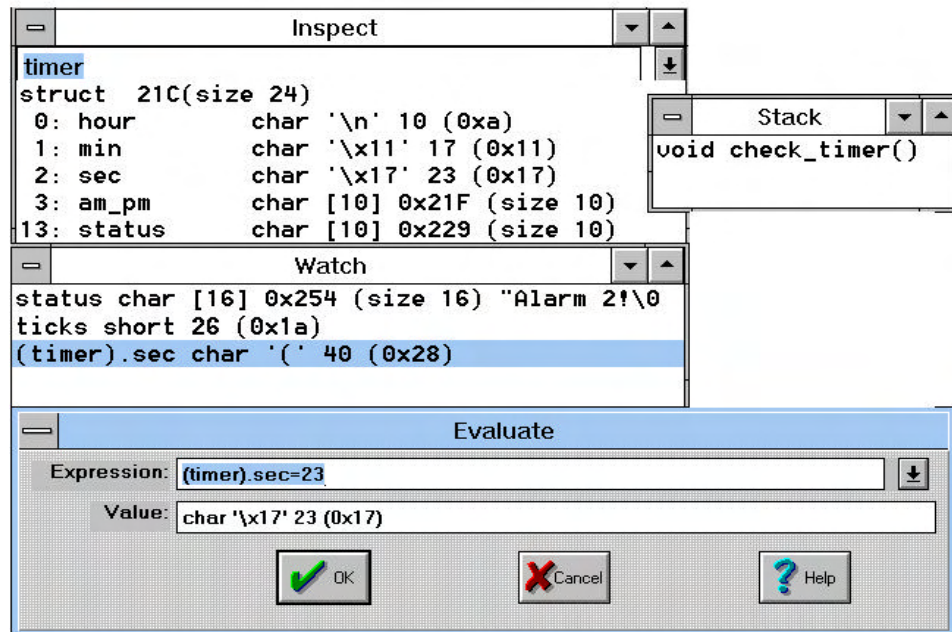


Figure 33: C call stack, Evaluate, Inspect & Watch Windows

Inspect Window

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

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

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

Watch Window

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

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

Evaluate Window

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

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

Stack Window

The **Stack** window displays the "call stack," or the list of functions called to reach the current point in the application, and the current value of parameters passed to them.

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

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

RTXC Window

EMUL196™-PC / SDU has built-in support for the RTXC multitasking kernel from Embedded Systems Products.

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

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

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 34: The Tool Bar

Help Line

At the bottom of the EMUL196™-PC / SDU window is a line of text that, depending upon the context, explains what the selected item is or what it does. This kind of context-sensitive help is turned on and off with the **Toggle help line** item in the **Windows** menu.Dynamic Data Exchange

file named “TIME.OMF”. The TIME program writes the value of a timer at the location 5010H through 5030H (look at the Shadow RAM in Figure 35). In the **Shadow RAM** Window you should choose **Display as... ASCII**. You will see the seconds and tenths of seconds being updated in your *Excel* cell, if you do the following:

Select the destination cell in *Excel* and enter the following formula (see Figure 35, cell A1):

=emul196 | shadow! '5016'

The cell will be updated and displayed as words in decimal representation. If you would like to have it displayed as ASCII, you must write your own function as a Macro under *Excel*. In Figure 35 we make a call for function *AsciConv* in cell A2. The function might look like:

Function AsciiConv(word)

AsciiConv = Chr(word And 255) + Chr((word / 256) And 255)

End Function

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

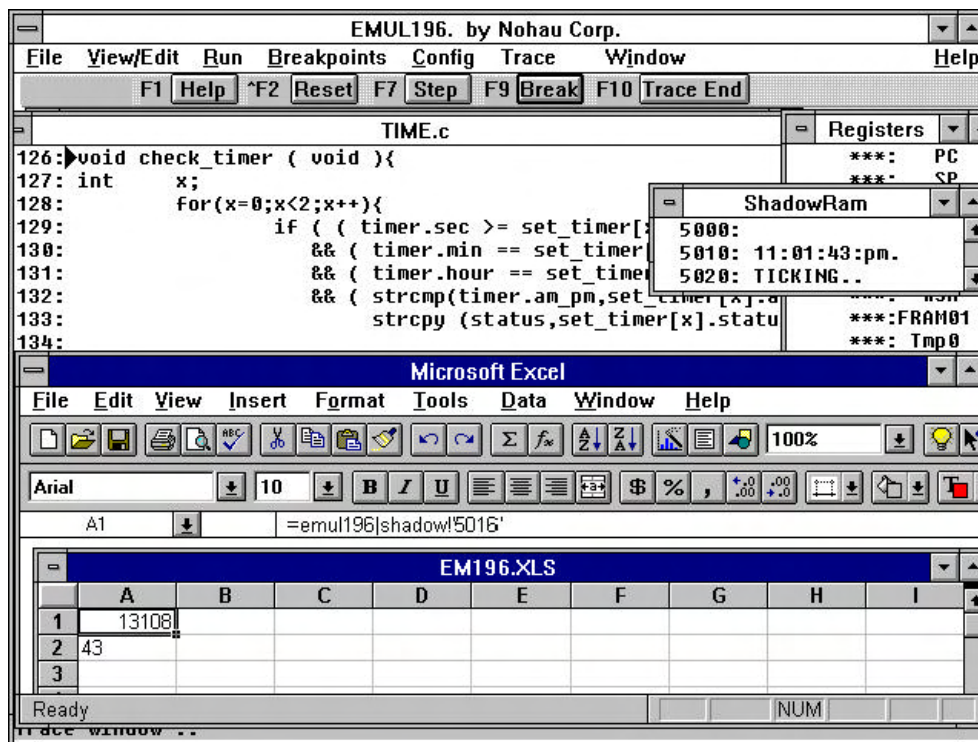


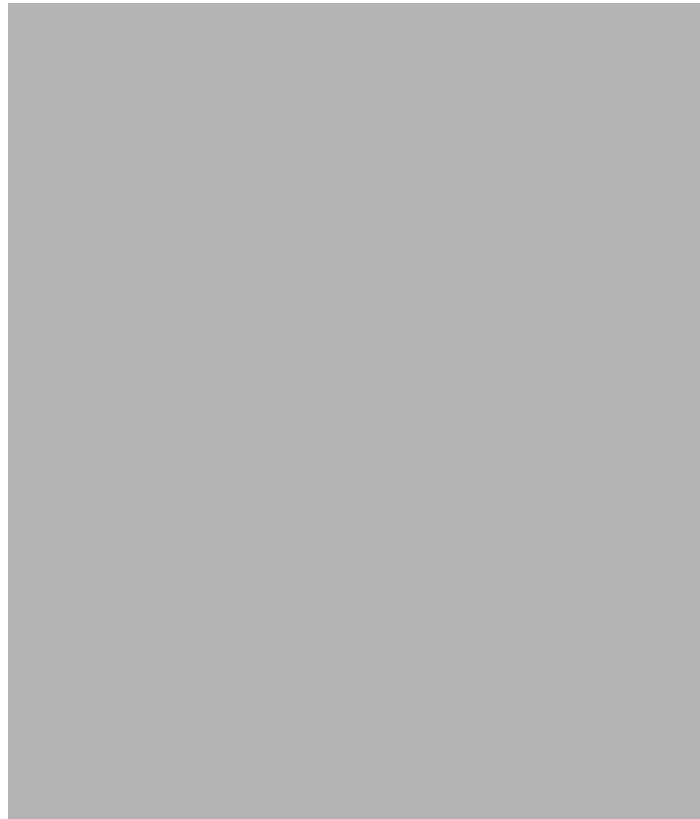
Figure 35: EMUL196™-PC / SDU with DDE to Excel

Appendix A

Reset Cut Required on Intel's 196EA Evaluation Board

The following steps are required to work around the evaluation board's on-board flash RS232 RISM monitor.

1. Perform cut at shown below.



Back
Of
Board

2. Solder resistor of 10K from pal U11 pin 4 to pin 24.

Appendix A

Reset Cut Required on Intel's 196EA Evaluation Board EMUL196™-PC/

Appendix B

INTEL Serial Debug Unit (SDU) Standardized Connector Specification

Rev. 2.0 10-17-95

In order to standardize the connector interface to the SDU port on next-generation products, Intel is providing this specification to both development tool vendors and customers.

Benefits include:

- same SDU interface port between all development tool vendors
- low-cost, readily available connector specified
- connector footprint allows for PCB to be layed out to easily support connector addition after manufacturing of the module.
- "keyed" connector eliminates mis-socketing problems

Connector Specification

The specified standard connector is a readily available 10-pin, 2-row header with 0.25" square-pin headers on 0.100" centers. The pin-out is as follows when looking down at the PCB board. On the PCB board (male) side, the square-pin at position #1 (KEY) will be removed. Likewise, on the cable (female) side, the hole at position #1 (KEY) will have an insert that will prevent mis-socketing of the connector.

2	4	6	8	10
N.C.	TCLK (SDLD	THS (SDU)	TDI (SDU)	TDO (SDLD)
KEY (N.C.)	GND	GND	RESET"	VCC
1	3	5	7	9

(TOP VIEW LOOKTNG AT PCB BOARD)

Appendix B

INTEL Serial Debug Unit (SDU) Standardized Connector Specification

Signal Descriptions

General pin descriptions given below, consult device Target Specification for details.

Pin	Signal Name	Signal Description
1	KEY - N.C.	no signal - reserved for reverse socketing prevention
2	N. C.	No Connect - do not use
3	GND	target digital signal ground
4	TCLK (SDM	Clock, external clock input to target device SDU
5	GND	target digital signal ground
6	THS (SDU)	Handshake, target device pulls low to signal busy condition
7	RESET#	RESET# signal to target board
8	TDI (SDU)	SDU data input, serial instructions and data into the target device
9	VCC	target device digital signal supply, +5.0V
10	TDO (SDU)	SDU data output. serial data transferred out of the target device

Symbols

.COV Files button 39
.ext field 21

A

Add .. 57
Add a watch point 49
Address space .. 56
Address.. 54, 55, 56, 57
Animation 65
Arrange Icons 58
At .. 53

B

Bargraph display 43
Base files ext, Memory Coverage Report
41
Block move.. 56
Break
 Button 69
 Emulation 51
 Now! 53
Break on internal access .. 51
Breakpoints 32, 63
 Clearing 53
 Menu 51, 63
 Setting 63, 64

C

C call stack 49
C check box 62
Call stack .. 56
Cascade windows 58
CCB Registers

CCB0, CCB1, CCB2, CCB3 27
Child windows 59, 65
 Data 33, 59
 Program 33, 59
 Registers 59
 Source 33, 59
 Watch 59
Clipboard, copy to 48
Close 58, 59
Code Coverage 34, 54
Code window 56
Color
 Config 33
 Menu 33
 Scheme field 34
 Setup dialog box 33
 Window 33
Color .. 54
Config
 Color 33
 Fast_Br_W .. 34
 Hardware 23
 Memory Coverage 34
 Menu 19, 20, 21, 22, 23, 26, 29, 30,
 31, 32, 33, 34, 41, 42, 43, 44,
 54
 Miscellaneous 29
 PPA 41
 Trace 34
Copy
 To clipboard 48
Coverage
 For all files 41
 Menu
 Edit 35
 Load 36
 Save 36
 Report 40

Detailed 40
File name 41
Summary 38
Reports 38
Window 34, 35, 36
Current program counter 63

D

Data Window 32, 56, 58, 59, 60, 61, 62
Menu 56, 61

DDE

Poke flag address 31
Sampling Interval 30, 31
Status 49
Value 31

Default symbols 49
Load 46

Delete

All 53
Breakpoint 63
Item 20
Projects 20
Watchpoint 57

Detailed Coverage Reports 40

Disable

All breakpoints 53
Disassembled instructions 63
Display as.. 56, 57
Dynamic Data Exchange 69

E

Edit

Coverage Ranges 35
Registers 57
Edit .. 56, 57
Emulator

Hardware .. 54
Hardware Configuration 23
Internal files 23
Emulator Hardware ..
Menu 23
Enable
Code space limits box 32
Evaluate 49
Exit 48

F

Fast Break Write 34
Fast Breaks 54
Fast_Br_W .. 54
File
Menu 17, 45, 46
Name 41
Fill.. 56
Full Reset 33
Function 55, 56

G

GO 51
Button 68
FOREVER 51
To cursor 51
Until cursor 63
Go to .. 51

H

Hardware
Config 23
Configuration Dialog Box 24
Hardware breakpoints .. 51
Help

Button 68
Line 68
Menu 59

Hex

Record files 21
Hexadecimal address 63
High address 32
Hot Keys 45

I

Icons, Arrange 58
Info .. 59
INI196 18
IniEdit 17
Initial Configuration 17
In-line assembler 64
Inspect 49
Window 66, 68
Intel Hex files 21

J

J2 header
Emulator 23

K

K2 mode 27

L

Linked Object files 22
Load
Coverage ranges 36
Default symbols 46
Path 21, 22
User Modules 21

Load code .. 21, 46
Low address 32

M

Mapping
Emulation Memory 23
Memory
Mapped devices 31
Scroll range 29, 32
Memory Coverage
Config 34
Detailed Report 40
Summary Report 38
Window 34, 35, 36
Memory Coverage .. 54
Memory map .. 54
Menus 45
Breakpoints 51
Color 33
Config 19, 20, 21, 22, 23, 26, 29, 30,
31, 32, 33, 34, 41, 42, 43, 44,
54
Data 56, 61
File 17, 45, 46
Help 59
Program 54
Registers 57, 60
Run 63
Shadow RAM 57
Source 55
View/Edit 48, 65
Watch 57
Windows 58, 69
Microsoft Windows 17, 31
Miscellaneous 54
Config 29
Item 29

- Menu 30
- Setup 32
- Miss bin 43
- Module 55
- Move .. 17
- MS Windows 17, 30

N

- Next window 58

O

- Object files, linked 22
- Odd
 - Address floating point 62
 - Address integers 62
- Open a new
 - Child window 58
 - Source code window 58
- Origin (at program counter) 55
- Original Address 56, 57
- Override at Reset 32

P

- Parameters in Hex 57
- Path
 - Dialog box 22
 - Internal files 23
 - Load 21
 - Setting 21
 - Source 54
- Paths .. 21, 54
- PDEN 28
- Performance Analysis 41, 43, 54
 - Add button 43
 - Adding a Bin 44

- Bargraph option 43
- Miss bin 42
- Poking a value 31
- Port
 - Address 23
- Power Down Mode 28
- PPA 41, 43
- PPAnalyzer .. 54
- Preferences 48
- Program
 - Menu 54
 - Window 32, 33, 46, 54, 55, 59, 62, 63, 64, 65
- Project name .. 54
- Projects
 - Creating 20

R

- Registers
 - Menu 57, 60
 - Window 59, 60
- REMAP 28
- Remove .. 57
- Remove Symbols 47
- Reset
 - Button 68
 - Chip after load file 29
 - Chip and Break 51
 - Chip and Go 51
 - Chip at start up 29
 - Override at 32
- Result files ext. field 40
- RTXC Window 68
 - Open 58
- Run
 - Menu 63
 - Program Manager menu item 17

S

- Save
 - Button 34
 - Coverage menu 36
- Save code as . 46
- Search 50
 - Next 50
 - Previous 50
- Select window class 34
- Send Command error 17
- Set
 - New PC value at cursor 55
 - Project Name Dialog box 20
- Setting the Paths .. 21
- Setup
 - Dialog box 63
- Setup .. 53
- SETUP.EXE 17
- Shadow RAM 57, 59, 62
 - Menu 57
- Show
 - Function 57
 - Load info 47
- Single step 65
- Software
 - Configuration 19
 - Configuring 19
- Source
 - Menu 55
 - Paths 22, 54
 - Window 33, 50, 55, 56, 58, 59, 64, 65
- Special Conditions .. 53
- Stack
 - Window 67
- Step
 - Button 65, 68
 - Into 50

Over 50

Summary Coverage Report 38

Symbol Files 32

T

- Tile windows 58
- Toggle 51
 - Breakpoint 55, 56
 - Help line 58, 69
- Trace
 - Beg 69
 - Config 34
 - Setup 69
 - Window 59, 65
- Trace .. 54

U

- User
 - Defined symbols 49
 - Load modules 21

V

- View
 - Assembly code 56
 - Source window 55
- View/Edit Menu 48, 65

W

- Warnings
 - Installation 24
- Watch
 - Menu 57
 - Window 57, 59, 67
- Watch dog disable 28

Watch point 30

 Add 49

 Delete 57

Windows

 Code 56

 Colors 33

 Data 32, 56, 58, 60, 61, 62

 Menu 58, 69

 Program 32, 33, 54, 59, 62, 63, 64, 65

 Registers 59, 60

 Source 50, 55, 56, 58, 59, 64, 65

 Watch 57, 59, 67

Z

Zoom 58