*Uni*ROM Support Files

for

SingleStep 68K

Introduction

*Uni*ROM provides advanced memory emulation and hardware support for Software debuggers. This manual documents the TechTools support files for the SingleStep Debugger on PC platforms.

*Uni*ROM enhances the SingleStep debugger by:

- Eliminating the need for a target Serial Port and support hardware.
- Eliminating the need for extra SRAM during debugging to hold the application.
- Allowing the application code to run "in-place" for more accurate emulation.
- Adding a high speed parallel interface for quick startup and application loading.
- Allowing vector and code patches in EPROM space.

System Overview

This manual documents the SPECIFIC issues involved in configuring *Uni*ROM, SingleStep and your target monitor to work together. It does NOT attempt to replace the manuals or procedure documentation included with each separate product. This manual will refer you to those documents for detailed instructions.

We understand that there is a considerable amount of data to absorb from all manuals combined, so we will use this document as the over-all guide to getting started. Before getting into the details, it would help to have a general understanding of HOW the over-all system works and what part each piece plays.

SingleStep is a target monitor based software debugger. This means that is executes a very small program on the TARGET called a MONITOR. This MONITOR communicates to the HOST PC through a serial port. The HOST PC executes the DEBUGGER which contains the bulk of the debugging code and all of the user interface code. All user commands (like setting a break-point) are converted to simpler commands (like write this byte to address xxx) and sent to the MONITOR program over the serial link.

The MONITOR program is reduced to the bare minimal code necessary to support the DEBUGGER. Essentially, the MONITOR reads and writes memory and registers, implements single stepping and traps errors and breakpoints.

*Uni*ROM is a memory emulator with many advanced features. Many of these features can help real-time debugging in a stand-alone configuration (without a software debugger). Refer to the *Uni*ROM Users' Manual for details on these features. In addition, *Uni*ROM provides a VIRTUAL UART (URCOM) and CONSOLE paths that support debuggers. These features provide a communications path between the target and the HOST without using any target hardware resources. *Uni*ROM also accepts target write cycles, allowing the target to write into its code space. This capability allows a target program (like a MONITOR) to load applications and to write breakpoints into the code space to affect debugging.

These support files document how to configure the SingleStep DEBUGGER and MONITOR programs to communicate through *Uni*ROM's VIRTUAL UART and CONSOLE path. They also provide basic script file templates for configuring *Uni*ROM and loading the MONITOR file(s).

Installation

Requirements

The SingleStep MONITOR must be assembled and linked to operate. All of the support files supplied by SDS and TechTools assume that you are using the SDS C compiler, assembler and support tools.

SingleStep requires:

- 640K of main memory and a minimum of 4MB of extended memory available
- Windows 3.1 or higher.
- 386SX or better
- A HARD DISK with at least 4 MB of free space

URLOAD and URTERM require:

- 286 or better
- 384K main memory, 0 extended memory
- DOS 3.1 or later

Installing SingleStep

Install SingleStep as instructed on the disk label. You should probably install the debugger files into the same directory as the SDS compiler and assembler tools.

If you installed into a new directory, add the "CMD" subdirectory to your path.

Installing and Testing Uni*ROM*

Refer to Chapters 1and 2 of the *Uni*ROM Users' Manual for details on installing and testing *Uni*ROM. In addition, refer to Chapter 14 if you are installing multiple *Uni*ROMs. The included support files and this documentation assume a SERIAL connection to *Uni*ROM(s) for control/configuration/loads and CONSOLE connections. We recommend using the SERIAL connection initially. You can add the PARALLEL connection later for ultra-fast file transfers.

You will probably want to place URTERM.exe and URLOAD.exe files in a directory on your path so that you can execute them from anywhere.

Before continuing, you may wish to review the SingleStep documentation to get a better idea of the over-all monitor build process.

Installing SingleStep Support files

The SDS development tools use several batch files to automate tasks. These batch files assume a specific directory structure for the SingleStep files. The SingleStep INSTALL program automatically creates this directory structure and places the correct files in the correct places. Specifically, SDS expects you to CD to a <BOARD> directory to build your monitor files. If the SingleStep install did not create a directory for your target board, you should create one under the "x:\sds65\boards\manuf\" directory. Refer to Chapter 2 in the "Monitor Preparation and Kernel Support" manual for details on selecting or creating a <board> directory and files. We HIGHLY recommend that you adhere to this structure.

The "*Uni*ROM support files for SingleStep" disk contains several support files to simplify building a SingleStep monitor with *Uni*ROM support. These files should be copied to specific directories as indicated below.

```
copy  a:\devices\unirom.s   <SDSROOT>\boards\devices\*.*
copy  a:\manuf\*.*          <SDSROOT>\boards\manuf\<board>\*.*
```

Substitute the actual path to your SDS tools root directory for "<SDSROOT>".
Substitute the name of the board directory for "<board>".

This disk contains two directories. The contents of each are discussed below.

\DEVICES

This directory contains UNIROM.S, a routine which allows SingleStep to communicate through the URCOM option board in *Uni*ROM.

\MANUF

This directory contains a batch file and several template script files.

FILE	Description
mkroms.bat	A batch file to automate the creation of ROM images for loading into *Uni*ROM.
ur08x1.cfg	A *Uni*ROM script template for a single 8 bit UniROM for targets with an 8 bit EPROM data path.
ur16x1.cfg	A *Uni*ROM script template for a single 16 bit UniROM for targets with a 16 bit EPROM data path.
ur08x2.cfg	A *Uni*ROM script template for two 8 bit UniROMs for targets with a 16 bit EPROM data path.
ur08x4.cfg	A *Uni*ROM script template for four 8 bit UniROMs for targets with a 32 bit EPROM data path.
ur16x2.cfg	A *Uni*ROM script template for two 16 bit UniROMs for targets with a 32 bit EPROM data path.

Adding UniROM *to the Device Library*

Before compiling the monitor or test programs, we need to add *Uni*ROM to the Devices Library. Change to the "\boards\devices" subdirectory under your SDS tools. FROM THIS DIRECTORY, type "makedev". This will re-compile the entire devices library and add to it the *Uni*ROM device.

Getting Started

We will refer to several basic configuration parameters throughout the remainder of this document. These should be determined now to eliminate any confusion or misinterpretation later. We discuss each key configuration item below and provide a summary sheet at the end of this section. You may find it useful to photocopy the Summary Table and fill it in as you go along.

Memory Map

Of course you need to know how your target's memory is located. This information is used to locate the monitor code. The <board>.def file defines the RAMADDR, ROMADDR and RAMSIZE symbols. In addition, the ROMADDR is usually also the start of the emulation memory space and therefore used to generate the monitor ROM images.

INTERVAL Determination

The INTERVAL is the EPROM DATA PATH WIDTH in bytes. This is simply the number of 8 bit EPROMs (or double the number of 16 bit EPROMs) used for code storage. Put another way, it is the EPROM DATA WIDTH (in bits) divided by 8. This is used to translate target address references to actual emulator offsets.

URCOM Address Selection

*Uni*ROM allows considerable flexibility in the selection of the URCOM address. However, the selected address must meet the following criteria:

It MUST reside within the address space emulated by *Uni*ROM.

URCOM is a memory mapped device within *Uni*ROM. Its address must fall within the address space being emulated by *Uni*ROM.

It must be properly ALIGNED, based on the INTERVAL:

INTERVAL	Alignment	Valids Addresses end in:	Examples
1	16 Byte	0x0	0xXXX10, 0xfff0
2	32 Byte	0x00, 0x20, 0x40, 0x60, 0x80, 0xa0, 0xc0 or 0xe0	0xXXX20, 0xffe0
4	64 Byte	0x00,0x40, 0x80, or 0xc0	0xXXX40, 0xffc0

This address must be a multiple of 16 for 8 bit EPROM paths, 32 for 16 bit data paths and 64 for 32bit data paths.

Example1: IF you are emulating 2-8bit EPROMs or 1-16bit EPROM, you have a 16 bit EPROM data path (INTERVAL=2). In this case, the URCOM address would have to be 32 Byte aligned.

Example2: IF you are emulating 4-8bit devices or 2-16 bit devices, you have a 32bit data path (INTERVAL=4). In this case, the URCOM address would have to be 64 Byte aligned

Another sanity check: The selected address must be evenly divisible by (INTERVAL*16).

In C nomenclature, URCOM_ADDR % (INTERVAL*16) == 0;

The selected address must be available.

The selected address must NOT be in use by the monitor, or any applications that will be loaded later. Since this address is manually selected, space is NOT reserved by the linker. It is your responsibility to insure that you have selected an available location.

A popular spot is the very top of ROM (less 16,32, or 64 bytes). Another popular spot is at (16Kbytes - 16, 32 or 64bytes) above the start of the monitor. This allows a total of 16KBytes for the monitor. The monitor normally uses just under 12KBytes. Any applications could be consistently loaded at (START-OF-ROM + 16K).

The first option makes it very simple to set the URCOM offset in the URLOAD script files. Our script file templates use the first approach. The following table summarizes the correct settings if we assume that URCOM is placed at the top of the memory emulated by *Uni*ROM.

INTERVAL	URCOM_OFFSET (used in UniROM.cfg)	URCOM_BASE (defined in <board>.h)
1	Device SIZE - 0x10	Top of Emulation Memory - 0x10
2	Device SIZE - 0x10	Top of Emulation Memory - 0x20
4	Device SIZE - 0x10	Top of Emulation Memory - 0x40

Summary Sheet

Target Description: _____

HINTS/NOTES	PARAMETER	VALUE
Stating address of the Memory devices being emulated by *Uni*ROM. This is usually the same as ROMADDR in the <board>.def file	Emulation Start ADDRESS:	<1>
TOTAL capacity in KBYTES. NOTE: a 27210 is a 64Kx16 device with a total capacity of 128 KBYTES	Emulation Memory Size (KBYTES):	<2>
<1> + <2>	Top of Emulation Memory:	<3>
2-16bit EPROMS = 32 4-8bit EPROMS = 32 2-8bit EPROMS = 16	Emulation data path WIDTH (bits):	<4>
<4> / 8 (used in <board>.h)	INTERVAL:	<5>
8 bit *Uni*ROMs have 1 emulation board. 16 bit *Uni*ROMs have 2 emulation boards	*Uni*ROM Emulation Boards:	<6>
(used in <board>.h)	URCOM_BASE:	<7>
(URCOM_BASE - Start Address) / INTERVAL ((<7> - <1>) / <5>) (used in unirom.cfg)	URCOM OFFSET:	<8>

Sanity Checks:

INTERVAL = 1, 2 or 4?
Total *Uni*ROM Emulation Boards(<6>) = INTERVAL(<5>)?
URCOM ADDRESS(<7>) >= Emulation Start Address(<1>)?
URCOM ADDRESS (<7>) < Top of Emulation Memory(<3>)?
URCOM ADDRESS (<7>) evenly divisible by INTERVAL(<4>)?
URCOM OFFSET(<8>) evenly divisible by 16 (ends in 0x0)?
URCOM OFFSET(<8>) <= (Emulation Memory Size - 16)?

Building a Test Monitor

Before building the test monitor, you will need to evaluate the <board>.def and <board>.h files to insure that they accurately describe your target. If your target board was one of the pre-built options, they are probably in good shape. The following section describes the CHANGES to the <board>.h file necessary to use *Uni*ROM for communications. It does NOT address the remaining non-communications related parameters.

Changes to the <board>.H file

The <board>.h file was configured to match a particular target board. You may need to modify this file to reflect your target environment. In addition, a few minor changes are necessary to configure the monitor to use *Uni*ROM for debugger communications instead of the target's serial port. These changes are discussed below:

Define URCOM Base Address

Add the following line to the top of the <board>.h file. Where "<addr>" is the URCOM ADDRESS selected in the previous section (item <7> in the summary table).

```
#define URCOM_BASE  <addr>
```

Comment Out 5 Lines

*Uni*ROM/URCOM automatically handles HARDWARE flow control. When the internal 128 byte FIFO becomes nearly full, *Uni*ROM will deassert the "clear to send" signal to the HOST until the monitor removes at least 16 bytes from the FIFO.

*Uni*ROM will stop removing characters from the target's TX buffer when the HOST deaserts his "clear to send" signal. The monitor will not write another character to the TX buffer until the previous character has been sent. This mechanism makes hardware handshaking completely automatic, eliminating the need for software control of these functions. SingleStep uses several defines to determine if it should build support for these function into the monitor. We comment out FLOW_IN, FLOW_OUT and COMBINED to eliminate them from the build.

This version of the *Uni*ROM device file does not support interrupt driven communications. We comment out VECT_READ and READ_LEVEL to generate a POLLED mode monitor.

Place a semicolon in front of the following lines to disable them:

```
; FLOW_IN
; FLOW_OUT
; COMBINED
; VECT_READ
; READ_LEVEL
```

Add or Edit the following 6 Lines

```
#define INTERVAL  <x>
DEV_IN=UNIROM
DEV_OUT=UNIROM
BASE_IN=URCOM_BASE
BASE_OUT=URCOM_BASE
BAUD_IN=38400
BAUD_OUT=38400
```

<x> is line item <5> from the summary table. BAUD_IN and BAUD_OUT do NOT set the communications speed. However, they must be present, valid and set to the same value to make the monitor happy. The debugging communications speed is set by a parameter to the CONSOLE command in UNIROM.CFG.

Building a Test Monitor

SingleStep includes a test monitor file to assist in initial bring-up. This file is linked with the actual device files and your target specific files. This allows simple testing without the added complexities of the debugger monitor itself. We highly recommend building and loading this file before proceeding on to the actual monitor file. Once this file is built, the actual monitor can be built with two simple commands and NO changes to any of the configuration files!

From this <board> directory, run "..\..\debug\mktmon <board>".

On success, this will generate "u.out", a fully linked test file that is built to run on your target and to communicate through *Uni*ROM's URCOM option.

Creating Test Monitor Image Files

The *Uni*ROM script templates are setup to load one BINARY IMAGE file into each emulator board. The U.OUT file is split into multiple BINARY IMAGE files with "down" commands. We enclosed a batch file (mkroms.bat) to automate this process.

Run "mkroms U 0x<ADDR> <COUNT>" to create files for URLOAD. Substitute the Emulation starting address (item <1> from the summary table) for "<ADDR>". Substitute the INTERVAL (item <5> from the summary table) for "<COUNT>".

Example1: Target has 16bit EPROM data path at 0x60000
enter: mkroms U 0x60000 2

Example2: Target has a 32 bit EPROM data path at 0x800000
enter: mkroms U 0x800000 4

This creates one or more EPROM images named "MON.x", where 'x' is the byte order for this file as follows:

INTERVAL	Files Produced	Byte Positions
1	mon.0	ALL
2	mon.0 mon.1	HH and ML (0,2,4) MH and LL (1,3,5)
4	mon.0 mon.1 mon.3 mon.4	HH (0,4,8,C) MH (1,5,9,D) ML (2,6,A,E) LL (3,7,B,F)

Configuring a Script File

URLOAD uses script files to control *Uni*ROM configuration and file transfers. We included several script templates to get you started. Note that these are not your only options. URLOAD scripts allow complete flexibility. It is possible to mix-n-match any number of *Uni*ROMs in any desired configuration. For example, one could put 2-16 UniROMs on one target, 4-8bit UniROMs on another target and 1-16bit and 2-8 bit UniROMs on a third target. A single script file could configure, load and control all 9 *Uni*ROMs. The included templates cover the most common configurations. Other configurations are easily extrapolated from these. See the *Uni*ROM Users' Manual for detailed information on script files and their commands and uses.

Select a Script Template file

Determine which URLOAD script template file matches your target EPROM data width (INTERVAL) and *Uni*ROM hardware as follows:

Target EPROM data path	*Uni*ROM HARDWARE	Template
8	one 8 bit *Uni*ROM	ur08x1.cfg
16	two 8 bit *Uni*ROMs	ur08x2.cfg
16	one 16 bit *Uni*ROM	ur16x1.cfg
32	two 16 bit *Uni*ROMs	ur16x2.cfg
32	four 8 bit *Uni*ROMs	ur08x4.cfg

Copy the selected file to the default URLOAD script file name "UNIROM.CFG".

Edit UNIROM.CFG to make it match your exact setup. The following sections describes the lines that may need modification. The *Uni*ROM Users' Manual contains complete descriptions of all commands and options, but you will probably be able to work from these instructions and the comments in the template file itself.

Port Selection

This should be set to the port you intend to use for SingleStep communications. At first, we will set the BAUD RATE to 38400, SingleSteps highest setting.

Valid PORT settings are COM1, COM2 COM3 or COM4.

Device SIZE and Type

The SIZE parameter is in KBYTEs. Notice that a 27512 is 512Kbit device or 64KBytes total. Also note that 16 EPROMs can be confusing. A 27240 is a 4Mbit device, organized as 256Kx16. This is read as 256K WORDS or a total of 512KBytes.

The type parameter simply indicates the type of device *Uni*ROM is emulating. If your target sockets are wired for FLASH devices, select FLASH, otherwise select EPROM

The '0' in this command is the ADDRESS parameter. We have chosen to leave this set to 0 so that all memory references are OFFSETS from beginning of emulation memory. This is simpler when interleaved files are being loaded into individual boards.

Reset

DON'T overlook this one. When debugging a program (or anything else) it is very important that your tools operate consistently. Configuring and connecting the reset line helps to insure that consistency.

Fill

We use this command to initialize the RAM underlying the URCOM board. This insures that the target will see that the URCOM has not been initialized yet and will NOT make false decisions based on the data in the RAM at the URCOM's location. This initialization capability eliminates the need to hard-code it within the monitor as "define constant" statements. It also allows us to release reset and wait for target activity BEFORE initializing the CONSOLE connection (and therefore URCOM). The target monitor will hit the initialization code long before URCOM is initialized. This insures a consistent start by forcing the target to wait for URCOM initialization.

The first parameter is the first byte of the URCOM offset within this emulation board. This should be set to the value calculated in line <8> of the summary table. The second parameter should be set to the first parameter + 15. This is the LAST byte to fill, for a total of 16 bytes. The third parameter is the fill value. Leave it set to 01.

Console

The CONSOLE line specifies that we should initiate a CONSOLE connection with the URCOM option board. The second parameter specifies the URCOM OFFSET within THIS emulation board. It should be the same as line <8> in the summary table. The "NONE" specifies that we should not generate interrupts. The final parameter is the CONSOLE connection BAUD rate. This MUST match the baud rate the debugger will use to communicate with the target.

Loading the Test Monitor

Loading the test monitor is as simple as typing: “URLOAD”. URLOAD looks in the current directory for a script file called “unirom.cfg” for its configuration and commands.

Verifying Test Monitor Operation

The Target STATUS LED on *Uni*ROM should be flashing randomly, showing a lot of target accesses to *Uni*ROM. If it is NOT flashing or is barely “winking”, something is wrong. Review the troubleshooting section and fix the problem before continuing.

If the target activity LED is flashing away, you are ready to verify that the test monitor is actually communicating. The test monitor simply prints a colon for a prompt and then waits for a character. It echoes any character it receives and then send another colon. You can use any terminal emulator that supports 38.4Kbaud to send characters to the test monitor and observe the results.

For simplicity, you can use URTERM.exe to do this. Type “URTERM” to start up the *Uni*ROM terminal emulator. You should see *Uni*ROM’s main ASCII menu. Select C and press the enter key to return *Uni*ROM to CONSOLE mode. You should get a message informing you of how to escape back to command mode. Start typing characters and observe the results. You are now communicating with the test monitor through *Uni*ROM’s URCOM board. If the test monitor does not echo your characters as described, refer to the troubleshooting section below. In either event, press ALT-X to exit URTERM.

Troubleshooting

If the target activity LED does not flash, try removing the reset line and cycling power on the target. If this brings it up, the reset is configured incorrectly or connected to the wrong spot on the target. If the problem persist, Verify that the target interface cables are connected to the correct target sockets (HH byte going to the HH socket ...). Incorrect byte ordering will likely create bad opcodes. Many processors will detect the bad opcode and continually reset themselves or HALT completely. Finally verify all UNIROM.CFG, <board>.h and <board>.def settings.

If the target status LED shows a lot of activity, but the test monitor does not print anything, then verify your calculations for the URCOM offset. Verify that the URCOM_BASE defined in <board>.h is the actual TARGET ADDRESS for URCOM and that the value used in unirom.cfg is the URCOM OFFSET. Verify that the selected values pass all of the “sanity” checks on the summary sheet. Verify that the MON.0 file was loaded into the LOWER emulation board of the *Uni*ROM with the URCOM option installed. Also verify the target interface cable ordering as described in the previous paragraph.

If the monitor reacts properly but the characters are garbled, the code is executing correctly, but URCOM communications are wrong. The most likely cause for this is that the target cables are installed consistently with the way the files were load into the emulation boards, but the mon.0 file was not loaded into the LOWER emulation board. The mon.0 file MUST be loaded into the LOWER emulation board of the UniROM with the URCOM option installed to work within the guidelines presented in this document. If you re-arranged the byte ordering to make the cables install cleaner, call our tech-support for assistance.

Building the Actual Monitor

Building the Monitor

Once the test monitor is built and verified, the real monitor is very easy to build. In fact, NO items need to be changed. Simply compile the real monitor by typing:

"..\..\com\mkmon <board>"

This creates "<board>.out. Next, type:

"mkroms <board> <start address> <interval>"

<start address> and <interval> are the same values used to build the test monitor (line items <1> and <5> from the summary table). This creates the individual files needed by URLOAD.

Loading the Monitor

Since the new image files have the same names as the test monitor image files, we do not need to modify unirom.cfg. Simply type "URLOAD" again and the new files are loaded into *Uni*ROM and the target is reset.

Testing the Build

Since the test monitor works correctly, we have a very high probability that the real monitor will work as well. The test monitor proves the communications path works; the target interface cable byte sequencing is correct; and the *Uni*ROM and URCOM configurations are probably correct. If it does NOT work, we can focus our attention on items that affect the real monitor, but not the test monitor. This would be very target specific. Review the settings in the <board>.h and <board>.def files. Also review the map files from the linker for clues about memory usage, stack placement, etc.

If the monitor is running correctly, it sends out three polling bytes every 5 seconds. These bytes are printable ASCII characters, so we can use URTERM again to verify that the real monitor is functioning. Simply start URTERM and enter CONSOLE mode like before. Approximately every 5 seconds, the three characters "{#@" should appear. If nothing appears, cycle power on the target. If this makes it start working, verify the reset configuration and connection point.

Once you see the {#@ sequence, type {#}. The monitor's polling sequence should change to {#+.

When you get this far, the monitor, *Uni*ROM and URCOM are all configured, installed and built properly. The only piece of the puzzle left is the SingleStep DEBUGGER itself.

Configuring SingleStep

No special configuration is required. Just remember to run URLOAD before starting SingleStep.

When starting a debugging session (File/Debug command), selelct the correct comX port and baud rate. Also be sure to check the hardware flow control box.

Enhancements/ NOTES

Increase Transfer BAUD-RATE

Once a working system is established, you choose to speed up URLOAD's transfers. URLOAD can transfer files at one baud rate, and establish a CONSOLE connection at a different rate. In the previous sections we used the same BAUD RATE for both to simplify everything. Simply edit the PORT line in UNIROM.CFG to set the BAUD RATE to 115200. This will give full-speed transfers. Leave the CONSOLE BAUD rate alone. This one sets the communications speed during CONSOLE mode.

Add Parallel Port Connection

For even faster transfers, you may choose to add a parallel connection to *Uni*ROM. URLOAD can transfer files across the parallel connection and still invoke a CONSOLE path to the SERIAL connection. Simply connect the parallel cable between the HOST parallel port and *Uni*ROM. If you are using multiple *Uni*ROMs, you will need to purchase Multi-Drop Adapter cables from TechTools to make this connection. Refer to the *Uni*ROM Users' Manual for details on parallel multi-drop configurations.

Edit the PORT line in UNIROM.CFG to change the PORT from COMx to LPTx, where x is the parallel port number.

Loading the Application with URLOAD

If your application is very large, you may choose to load it into *Uni*ROM before starting SingleStep. Simply add more LOAD lines to UNIROM.CFG to accomplish this. Even if you are using a serial connection, URLOAD will load the application many times faster than SingleStep will.

Refer to the SingleStep documentation for details on how to tell the debugger to suppress loading the application into the target, but still load its symbols.

Target Writes to UniROM

*Uni*ROM accepts target write cycles. This allows one to load an application into *Uni*ROM's emulation space and debug it "in-place". If *Uni*ROM is plugged into FLASH sockets, he will get the WRITE signal from the socket(s). If *Uni*ROM is plugged into EPROM sockets, he accepts a WRITE input from his feature connector.

HOWEVER, many targets are NOT CAPABLE of doing BYTE-WIDE WRITES to the EPROM/FLASH sockets. Often a single chip select line is used to activate ALL EPROMs in the code bank. This means that ANY WRITE to ONE of the EPROMS would result in writing garbage into the remaining EPROMs. The ideal solution would be modify the target's chip selects to qualify the EPROM chip selects with A0 (and possibly A1) to allow individual byte selection capability.

SingleStep has another work-around. The _write and _load aliases have a -w option to configure them to always do WORD wide writes. This forces the MONITOR to load applications and write to emulation memory in WORDS instead of BYTES.

Add the following to the SSTEP.INI file:

```
alias _load 'load -w $debugblk'
```


Summary

If you followed through this manual in sequence, you should have a fully functional setup. If you get stuck on any particular step, call our tech-support for assistance. Please be prepared with the following information:

1. The FIRST step that did not perform as described.
2. A completed copy of the Summary Table from page 6.
3. Your *Uni*ROM serial number(s) (located on the bottom of the units).
4. Any instructions that seem unclear or ambiguous.

TechTools Technical Support is available 9:00 to 5:00 CENTRAL, Monday - Friday.

TechTools

PO Box 462101
Garland, TX 75046-2101

Voice: (972) 272-9392
FAX: (972) 494-5814
email: support@tech-tools.com

