CSi-Mon™

User's Guide

Version 5.0
February 1998

Copyright and Trademark Information

United States Software Corporation
Concurrent Sciences Division
205 East 5th Street, Suite 6
Moscow, ID 83843
Telephone: 208.882.0445
Fax: 208.882.9774
E-Mail: tech@debugger.com

Quick Contents

Documentation Conventions

Computer output and code examples: `Courier`, usually in a separate paragraph.

Function names and command names: ***Bold italic***, functions are followed by parentheses, as in ***main()*** function.

Variables: Courier 11 italic (*`mt_busy`*).

File names: Times bold (the file **usrclk.asm**), in lower case.

Key names: Initial capital, in angle brackets, as in press <Enter>.

Menu names and selections, dialog box names, screen titles, window titles: Times bold, as in **File** menu**.**

Notes: Indicate important information.

Cautions: Indicate potential damage to hardware or data.

Warnings: Indicate potential injury to users.

Revision History

Revision Number	History	Date
2.0	Second Edition	June 1996

Quick Contents

D

I

Contents

Figures

Tables

Preface

Thank you for purchasing CSi-Mon™ monitor for Soft-Scope® debugger. CSi-Mon is a monitor that can be embedded in a remote target for use with Soft-Scope to aid in debugging an embedded application.

About this Manual

This manual provides a reference to the features and functions of CSi-Mon for Soft-Scope. We have organized the manual into three sections with several chapters in each section. The *Preface* (this section) contains information about the manual. *Getting Started* contains everything necessary to get a monitor running on a target. The *User Reference* contains advanced topics not covered in *Getting Started*. To make the best use of this manual, read the chapters in *Getting Started* in their entirety. The *User Reference* contains a collage of stand alone chapters that can be read independently of each other. A brief description of the chapters in each section follows:

Preface

About this Manual provides a road map to this manual, describing major sections.

Technical Support describes how to reach Technical Support and when to contact them.

Getting Started

Installation explains how to install CSi-Mon.

Preparation lists the tools you need to build the monitor and lists the resources the monitor will use.

Demonstration provides an introductory guide to building, installing, and testing a monitor on a target.

Configuration provides step-by-step instructions on configuring a new monitor.

User Reference

Combining the Monitor and your Application explains two methods for merging the monitor and your application.

Troubleshooting lists common problems and their solutions; things to try before contacting Technical Support.

Appendices

CSi-Mon Commands details the format and purpose of all the commands between Soft-Scope and CSi-Mon.

Register Tables describes each register and how CSi-Mon communicates it to Soft-Scope.

Breakpoint Handle Tables defines how breakpoints are stored and communicated.

CSi-Boot Error Messages lists and defines CSi-Boot error messages.

Technical Support

Concurrent Sciences, Inc. provides technical support to its customers via phone, FAX, E-mail, and the World Wide Web. Before you contact us, ensure that you have done the following:

- Be sure the information isn't already covered in the manual or in the **readme.doc** on the installation disks.
- If the problem is related to source changes, be ready to provide an example that reproduces the problem.

When you do contact us, be sure to provide:

- Your name
- The name of your company
- A method to contact you after we have researched the problem (phone number, FAX number, or E-mail address)
- Version of the monitor (found at the top of CSICFG and on the installation diskettes)
- Serial number for the monitor (found on the installation diskettes)
- Processor type
- Tool chain(s) used to build the monitor with version of each tool
- The **csicfg.cfg** file that was used to build the monitor should the technician request it
- A description of the problem and your efforts to solve it

How to contact Technical Support:

Phone: 208.882.0445 (between 9 AM and 5 PM Pacific Time)

FAX: 208.882.9774

E-mail: tech@debugger.com

World Wide Web: http://www.debugger.com

Getting Started

CSi-Mon User’s Guide

1. Installation

1

Before you can begin using CSi-Mon™, you need to install the software. We suggest having at least 3 megabytes of disk space available on the hard disk: 2 megabytes for the installation and 1 megabyte for building the new monitor.

Chapter Contents

Installing CSi-Mon

CSi-Mon is installed at the DOS prompt. You will create a directory on the hard disk for CSi-Mon. In this example, we will create a directory called CSIMON on the C: partition.

1 At the DOS prompt from the root of your C: partition, type MKDIR CSIMON and press <Enter>.

2 Move to the CSIMON directory. At the DOS prompt type CD CSIMON and press <Enter>.

3 Insert the installation disk in the floppy disk drive from which you will install CSi-Mon and type A:INSTALL A: and press <Enter>.

4 Insert additional diskettes when prompted until the installation is completed.

5 Refer to the file **contents.doc** in the CSIMON directory to ensure that all files are present.

CSi-Mon Source Files

Except for some start-up code written in assembly language, CSi-Mon is written in C. The assembly code is found in six files.

The monitor source is found in the SRC subdirectory. The following is a list of the source files and a description of each:

break.c—Breakpoint management code

common.asm—Common (to protected and real modes) low-level monitor support code

csicfg.h—(Generated by the CSi-Config configuration utility) Contains configuration information needed by C files

csicfg.inc—(Generated by the CSi-Config configuration utility) Contains configuration information needed by assembly files

csiloc.inc—Macros for CSi-Locate and CSi-Link™ support

csimon.h—Monitor definitions used by all C files

csimon.inc—Monitor definitions used by all assembly files

environ.h—Special directives for various compilers

extend.c—User extensions to monitor management code

extend.h—Prototypes of extension management; also location for user defined extensions of the monitor

main.c—Command processor and dispatch code

mapi.c—MAPI management code

mapi.h—Provide macro support for MAPI to the user

memory.c—Memory management (read, write, move, linear to logical translation, logical to linear translation, etc.)

ns486sxf.asm—NS486SXF processor startup code

pic.c—PIC controller management software

pmonitor.asm—Protected-mode specific low-level monitor support code

promice.c—Monitor communications management using Grammer Engine's PromICE

pstart.asm—General protected-mode processor startup code

register.c—Register management code

rmonitor.asm—Real-mode specific low-level monitor support code

rstart.asm—General real-mode processor startup code

siuart.c—Monitor communications management using various UARTs

xlate.c—Translation of communications content to something usable (ASCII decimal to binary decimal and vice versa)

2. Preparation

2

The purpose of this chapter is to describe what resources are required. The first section describes what tools are needed to build the monitor along with what tool suites are supported. A common question asked is "Why do I need a locator?" Because of this, the second section is dedicated to this topic. The last section details the hardware resources the monitor requires and works with.

Chapter Contents

Tools Required

As with any application, tools are required to build it. The monitor is written in MASM compliant assembly and in ANSI-C. Both assembly and C source make heavy use of macros and conditional statements to allow the most flexibility both for tools and for portability. The following tool suites are supported (see **readme.doc** found on the installation diskettes for the exact versions tested):

Assemblers:	Microsoft's MASM/ML*, Borland's TASM*, Pharlap's 386|ASM*
Compiler suites:	Microsoft Visual C*, Borland C*, Watcom C*, MetaWare's HighC*
Linkers/Locators:	Concurrent Sciences' CSi-Locate, Concurrent Sciences' CSi-Link™, Pharlap's LinkLoc*
Librarians:	Microsoft's LIB*, Borland's TLIB*, Pharlap's 386|LIB*

Why Is a Locator Needed?

When working in an embedded environment the developer generally needs to bootstrap the processor and related hardware into a known state. There isn't necessarily any loader or operating system that has already done this. Linkers, on the hand, combine the objects of an application into a relocatable format with the assumption there is an operating system that will place it where it wants it. The locator does the final job that an operating system would normally do by resolving the finished application into a physical location where it will always load.

The monitor is adept to this type of environment. It contains the startup code necessary to initialize the hardware as well as code necessary to move ROM data into RAM space. Because of this, it is designed to be burned into PROMs. However, to burn a PROM, a

.hex file needs to be generated, which is an absolutely located piece of code that is accomplished by the locator.

Hardware Resources

ROM/RAM Resources

The monitor requires roughly 8K of RAM and 20K of ROM. These values will vary widely depending upon what tools you use and what memory model is selected.

Processor Resources

The monitor will work with just about any x86 processor. The following is a list of processors supported as of this writing. The **readme.doc** on the installation diskettes should contain any newly included processors not listed below.

Table 2-1: Processors supported by CSi-Mon™

Intel Parts:					
8086	80286	80376	80386DX	80486DX	Pentium
8088			80386SX	80486SX	
80188	80C188	80188EA	80188EB	80188EC	80188XL
80186	80C186	80186EA	80186EB	80186EC	80186XL
AMD Parts:					
188EM	188EX	188ER			
186EM	186ES	186ER			

Table 2-1: Continued

National Semiconductor Part:			
NS486SXF			
NEC Parts:			
V20	V30	V40	V50

The standard 80x87 coprocessors are also supported.

The monitor also requires resources contained within the processor (varies depending on the above chosen processor). These include GDT space, IDT space, task register, IRQs, and debug registers. The number of GDT and IDT slots that the monitor uses varies with different configurations. You should reserve GDT[0..63] and IDT[0..39] to ensure compatibility with future monitor upgrades.

Communication Resources

The **readme.doc** will contain any new means of communication support more recent than this writing.

Serial Communications

In a typical installation, CSi-Mon communicates with Soft-Scope via a serial port on the target board. This communication is usually done with a UART, baud rate generator, and a Programmable Interrupt Controller (PIC). All of these are configurable. The following is a list of devices supported:

Table 2-2: Devices supported by CSi-Mon

UARTS:			
National Parts: **Intel Parts:**	8250 8251	16450 8274	16550
Baud Rate Generator:			
Intel Part:	8254		
PIC Controller:			
Intel Part:	8259		

NOTE: Those processors listed previously that have serial capabilities integrated within them are also supported (e.g., 186EB).

ROM Emulator

CSi-Mon can also communicate with Soft-Scope via a ROM socket using the PromICE ROM emulator from Grammar Engine. This is useful if your target board does not have a serial port. When building the monitor, use the **promice.c** module in place of **siuart.c**. See the **promice.doc** located in the SRC directory for instructions on building a monitor for the PromICE.

3. Demonstration

The purpose of this chapter is to demonstrate how to set up and run a monitor. The purpose is twofold. First, we want you to have the tools to perform a "sanity check." That is, we want to provide you with a quick method to discover that our product works. The second reason is to present a methodology for building, testing, and loading the monitor using working examples.

This chapter consists of three sections: *Loading/Installing a Monitor*, *Testing the Monitor*, and *Rebuilding the Monitor*. The first section will guide you through the process of loading or installing a stock monitor onto various targets. Once you have the monitor installed, the second section will give you tips, hints, and suggestions on how to test the monitor to determine if it is working properly. The third section will list step-by-step instructions on rebuilding the stock monitor with your tools.

Chapter Contents

Loading/Installing a Monitor

You will find several stock monitors in the standard directory on the installation diskettes. Please read the **readme.doc** found in this directory, as it contains information on each monitor included. This information details which targets the monitors were built for in addition to important configuration information (e.g., communication parameters). The **readme.doc** also contains information about monitors included on the diskettes but not listed in the manual.

Because monitors are capable of working on a variety of targets, we recommend that you try installing a monitor on a standard PC target before moving onto your own hardware. The following sections demonstrate both PC absolutely located monitors, the DOS EXEDBG monitor, and embedded monitors.

PC Absolutely Located Monitors

The easiest and quickest way to see a monitor work is to install one on a PC. The hardware is generally standard and therefore known, unlike embedded targets which vary. However, once loaded the monitor does control the PC, thus there are some restrictions to consider before loading and executing the monitor.

First, be sure that your PC is strictly in DOS mode. If it is not, the monitor will conflict with a multitasking operating system. You cannot install the monitor from within a DOS window (such as from Microsoft Windows* or OS/2*).

Second, for best results boot your machine without any device drivers or TSRs (they'll be useless after the monitor is loaded anyway). This is particularly true of EMM386, which interferes with serial communications.

Third, test your serial communications between the PC you will use for the monitor as a target and the PC that will act as the host. You will probably be using a NULL modem cable to connect the two. Be certain that you note which serial port of the target machine you are

using! Once you are sure that the two machines are capable of communicating, select the stock monitor appropriate for your needs.

We have included four monitors (two protected-mode and two real-mode) on the installation diskettes. (See the **readme.doc** for information on each monitor and for a listing of monitors included on the diskettes but not listed in the manual.) All of the monitors are configured identically except for the processor mode and the serial port used. All monitors are configured to communicate at 9600bps. The following is a description of the PC-based monitors:

r86com1.hex A real-mode, absolutely located monitor with no NPX support, configured for COM1. This monitor will run on most processors.

r86com2.hex A real-mode, absolutely located monitor with no NPX support, configured for COM2. This monitor will run on most processors.

p386com1.hex A protected-mode, absolutely located 386 monitor with no NPX support, configured for COM1. This monitor will run on any processor greater than the 386.

p386com2.hex A protected-mode, absolutely located 386 monitor with no NPX support, configured for COM2. This monitor will run on any processor greater than the 386.

Once you have decided which monitor to load, use the utility CSi-Boot (located in the UTIL directory) to load the monitor. To do so, enter the following at the DOS prompt:

```
csiboot r86com1.hex
```

(Replace **r86com1.hex** with the monitor you have chosen to use.)

Something similar to the following should appear on the target's screen:

```
CSi-Boot V3.0 Concurrent Sciences, inc. (C) 1990-1994
Load File name: r86com1.hex
Start location: 747c:0200
Hex file image: 00070000P to 000749c4P, 18830 bytes used, 0% unused
Loader address: 00039cc0P to 00049e20P
Scratch memory: 00010000P to 0001ffffP, 64K at 1000:0000
Booting...
```

Each line is described below:

Load file name	is the name of the monitor **.hex** file.
Start location	is the memory address where the monitor begins execution.
HEX file image	defines the area of memory where the monitor is located, the number of bytes it uses, and how much memory is available for other uses.
Loader address	defines the memory where the CSi-Boot utility is located.
Scratch memory	defines temporary memory where the monitor is placed while it configures the target before booting. This memory is available after the boot is complete.

When the load program prints the message "Booting..." on the screen, the monitor is ready to use with Soft-Scope. It has taken over control of the processor and is unable to write any further messages to the screen indicating that the boot process is complete. To reboot CSi-Mon, restart the target PC and repeat the process described above.

NOTE: CSi-Boot cannot load a CSi-Mon monitor into a memory location above 1MB nor can it load monitors that exceed 64K.

If monitor loading/installation was successful, you may proceed to the next section *Testing the Monitor*.

DOS EXEDBG

exedbg.exe is a DOS-loadable **.exe** application. It will only debug real-mode DOS applications.

First, your PC must be in DOS mode. Do not install the monitor from within a DOS window (such as from Microsoft Windows* or OS/2*), as we have found this type of installation to act erratically and unpredictably with the operating system.

Second, for best results, boot your machine without device drivers or TSRs. This is particularly true of EMM386, which interferes with serial communications.

Third, test your serial communications between the PC you will use for the monitor as a target and the PC that will act as the host. You will probably be using a NULL modem cable to connect the two. Know which serial port of the target machine you are using! Once you are certain the two machines can communicate, select the stock DOS EXEDBG monitor (currently, only one is provided; see the **readme.doc** to note the serial port configuration).

Run **exedbg.exe** at the DOS prompt. Next, run the application you wish to debug (see *Debugging .exe Executable Files* as described in the Soft-Scope User's Guide).

NOTE: You must run the application before testing the monitor. The application will activate the monitor.

Embedded Targets

Because there are a variety of vendors and evaluation boards available, we have included several monitors. See the **readme.doc** for information on each monitor and for a listing of monitors included

on the diskettes but not listed in the manual. Review the documentation from your vendor on how to place the monitor on the evaluation board.

Testing the Monitor

The bootstrap process of setting up and running a monitor can be stressful. This section will give you a method for quickly evaluating whether the monitor is functioning properly. Initially, use the same terminal program on your host that you used to ensure the connectivity of the PC, as described in previous sections. This information also applies for embedded monitors. Again, make sure the baud rates are correct. If the terminal program is up when the monitor first starts, you should see the following (or similar) sign-on string:

```
CSiMON-386DXP - Loaded PC/AT V5.01 (386DX PROTECTED MODE)
>
```

If you don't see this string, press the <Enter> key to see if the prompt ('>') appears. Occasionally the sign-on string may get discarded depending on the board you are using. (If you are using the TSR version, nothing will appear until after the application is executed. The ***ss_brk*** function embedded in the application will trigger CSi-Mon. See your Soft-Scope manual for more information.).

To troubleshoot, look for the following items:

- If no sign-on string appears and no prompt ('>') can be initiated with repeated presses of the <Enter> key, check the hardware (cabling, ports, etc.) and configuration (proper baud rate, port, IRQ, etc.). You may also need to ensure that the GDT table and pointer are located correctly and that the lgdt instruction has the appropriate segment override.

- If the sign-on string appears but there is no apparent response when the <Enter> key is pressed, check the hardware (cabling, ports, etc.).
- If the prompt (‘>‘) appears but no sign-on string is present, this may still be acceptable.

Once you have a prompt that can be initiated with the <Enter> key, the next step is to check the functionality of the monitor. Although the monitor is communicating, it is possible that other problems can occur due to misconfiguration.

NOTE: See *CSi-Mon Commands* in the Appendix to learn the commands to further test the monitor.

At the prompt type a ‘**v**’ and press the <Enter> key as follows:

```
>v
CSiMON-386DXP - Loaded PC/AT V5.01 (386DX PROTECTED MODE)
>
```

You should see the same string as the initial sign-on string as when the monitor first came up. If you don’t see the string, trouble shoot a second time by looking for the following:

- If the prompt appears but the sign-on string is either nonexistent or useless data, check that the monitor RAM memory is located in actual RAM space (look at the memory map in your locator’s map file).

Try the command ‘**V**’ and press the <Enter> key.

```
>V
80010110
>
```

This command tells SoftScope the parameters of the monitor. The leading ‘8’ tells Soft-Scope that the monitor is interrupt driven. If you are using a non-interrupt driven monitor, you may only see a ‘10110’. The importance of this command will become clear later when you actually run Soft-Scope.

- If a <02> returns or the monitor immediately hangs, you may have a problem with improperly setup segments (CS != DS) for flat model monitors. Some compilers will embed the jump table within the code segment. When the monitor attempts to evaluate the switch statement (within the monitor code) for the command processor, useless data is read and an error is returned, thus the monitor assumes that any command entered is incorrect.

Next, try the series of extension commands: attempt **E0**-**E4**. A discussion will follow each command.

```
>E0

Code=70000,75E70  Data=75E80,76CF0  Stack=75E90,76290,2C8
```

The first extension '**E0**' simply displays where the code, data, and stack start and stop are located in memory. This command is used to confirm that the command processor is working properly. It also further proves that the monitor is properly located (you may want to confirm this is correct by checking it against the map file created by your locator).

```
>E1
CSiMON extension number 1.
>E2
CSiMON extension 2 line 1.
CSiMON extension 2 line 2.
```

These commands are the first to test whether RAM initialization worked properly. The version string is located in the code segment, not the data segment, and therefore does not need to be moved during the RAM initialization process. The output of the '**E0**' command is accomplished by statically sending a single character at a time (you may want to refer to **extend.c**). However, the above strings are those stored in the data segment (possibly BSS depending on your compiler) and as such need to be copied from ROM to RAM before being used.

- If you receive useless data or no string at all and if the monitor is ROM based, you may have a problem with the RAM initialization

(the process of copying the initialization records from ROM into the RAM space upon startup).

```
>E3
0,0,0,0
```

Displays the configuration values.

```
>E4
<02>
```

This display is actually the expected result as '**E4**' is not a valid command in a stock monitor.

The next group of commands is used to test memory reading and writing coupled with manipulation of the register scratch area. Since we aren't switching to a live application, the registers are never updated. However, since the scratch area is in RAM, using these commands will test whether the linear to logical address translations are occurring correctly.

```
>r3c
0,0,0,0
>R3c,1,2,3,4
>r3c
1,2,3,4
>
```

The first command, '**r3c**', attempts to read some registers (see *CSi-Mon Commands* and *Register Tables* in the Appendix). Which registers the monitor attempts to read does not matter. The second command, '**R3c,1,2,3,4**', attempts to write values to those registers. The third command attempts to read the values from the registers.

- If any errors occur during these operations and you are using a protected-mode monitor, ensure that L2L_SRC and L2L_DST selectors are located in the RAM area. Because the monitor is designed to be flexibly built as segmented or not, it continues to

use selectors for translation whether the CODE/DATA is flat or not.

The next important command to try before starting Soft-Scope is the '**z**' command.

```
>z
>
```

The '**z**' command resets the monitor as if it had just come up from a power up. Occasionally a few bytes of useless data appear before the prompt—this is acceptable. The troubleshooting comments discussed earlier in this section apply here as well; be aware that initialization occurs in two locations in the code.

Lastly, repeat your attempt of the '**r3c**' command to determine that memory has been reinitialized (all of the register values should be zero).

Now you can close your terminal program and bring up Soft-Scope. Download one of the sample applications provided and do some sample debugging to further ensure that the monitor is working properly.

Rebuilding the Monitor

Now that you have set up and run a stock monitor, rebuild it using your own tools. The purpose of this exercise is to ensure that your tools are configured properly and compatible with the monitor. Once you have rebuilt the stock monitor, be assured that porting the monitor to your target board will go smoothly. To reduce the number of possible errors, we recommend that you initially only change the configuration of the tools.

For each of the stock monitors provided, there is a corresponding **.cfg** file. This file contains the configuration information that was used by the CSi-Config monitor configuration utility to build each monitor. Copy the appropriate **.cfg** file to the location of the monitor source.

Startup CSi-Config using the configuration file of the stock monitor you just tested on your target as follows:

```
CSICFG P386COM1.CFG
```

Using the <PgDn> key, scroll down to the section marked in red, *Specify the compiler to build CSi-Mon*. From there, use the <Tab> key (or <Shift+Tab>) to move the cursor to the desired compiler option. Once the cursor is located where you want, press the spacebar to select the option.

Repeat the process for your assembler in the following section, *Specify the assembler to build CSi-Mon*, marked in red.

Once you have selected your assembler, repeat the process to select a memory model in the next section, *Specify the memory model for CSi-Mon*. Depending on your compiler, small or flat is really all that is necessary.

Next, move your cursor to the *Path to library files*: entry field. Delete the stock path and replace it with the path to the location of your compiler's libraries.

Lastly, repeat the process once more to select your linker/locator in the last section, *Specify the link/locate tools to build CSi-Mon*.

Once you have completed these steps, press the <Esc> key. A box will appear giving you three options. Press '**G**' to generate the new files given the options you have selected. This will create four files: **csicfg.inc**, **csicfg.h**, **build.bat**, and **makefile**. Both the **build.bat** and **makefile** assume your tools are in your path. The **makefile** should work with any make utility as it specifically avoids a particular vendor's features.

If your tools are not in your path, you may need to edit either **build.bat** or **makefile**. All that is left is to build the new monitor.

Once the monitor is built, you can repeat the process described in this chapter to load/install and test the new monitor.

Congratulations! Now that you have enabled the monitor to work using your tools, see the chapter *Configuration* for a thorough discussion of the CSi-Config utility.

4. Configuration

CSi-Config is an interactive configuration utility in which you fill out an online form describing requirements for the monitor you want to build. Unless your target has unusual, special requirements, there is no need to edit a batch file or write special code.

If your target has special requirements, the distribution diskettes contain a copy of the CSi-Mon source code.

Chapter Contents

Before You Start

Each section in this chapter corresponds to a configuration step in CSi-Config, and each section describes technical information that is not hard to find, but may not be handy without prior preparation.

In some sections, especially the one on communications, some options listed may not apply to your hardware. It isn't necessary for you to read those options, although we recommend you carefully review each item and have the necessary information before beginning configuration of the monitor. Following is a brief summary of what you will do in each step:

1 Select an option defining the target's CPU and numeric coprocessor, if any. Specify the operating mode of the target, either real or protected; what hardware configuration it has; and what kind of monitor you want to use, absolutely located or ROMmed, for example. (Step 1 includes the sections *Configuring a Monitor* and *Target Hardware*.)

2 Specify #defines, equates, module names, and libraries for monitor extensions. (Step 2 includes the section *Configuring Monitor Extensions*.)

3 Define the specifications and capabilities of your target-hardware communications device. (Step 3 includes the sections: *Communication Configuration*, *Select a UART*, *Polling Communication*, *Interrupt-Driven Communication*, *National Semiconductor 16450/16550*, *Configuring Interrupt Controllers for National and Intel UARTs*, and *Intel 8251 or 8274 UARTs*.)

4 Configure application I/O through CSi's Monitor Application Programming Interface (MAPI). (Step 4 includes the section *Specify MAPI Functions*.)

5 Tell CSi-Config which of the tools you are using as listed in the chapter *Preparation*. (Step 5 includes the section *Selecting Tools*.)

6 Define where in the target computer's memory you want the monitor located. (Step 6 includes the section *Locate the Monitor in Memory.)*

Most descriptions of these steps are accompanied by an applicable example of the CSi-Config screen. Because the contents of each screen depend on previous configuration choices, the screens shown in the manual may not exactly match the screens you see when running the configuration utility.

Configuring a Monitor

During installation of CSi-Mon, the monitor configuration program was placed in the CSIMON\SRC subdirectory. To run the monitor configuration, type the following at the DOS prompt while in that directory:

```
csicfg [filename]
```

When the monitor has completed configuration, CSi-Config creates a file to store the specifications. The default name for this file is **csicfg.cfg**, and you can recall the file to build a new monitor or to make changes in the specifications of the existing monitor. To recall the file, include the filename on the invocation line shown above.

CSi-Config outputs the following four files:

csicfg.inc	Assembly and C include file containing specifications
csicfg.h	Assembly and C include file containing specifications
build.bat	Batch file to compile and link the monitor
makefile	Makefile equivalent to the above batch file for use with make utilities

The description of CSi-Config configuration options follows.

Target Hardware

You need to know the configuration of your target hardware. This includes whether a numeric processor exists and the CPU and numeric coprocessor types. Also, know whether your target is a PC or a custom target board.

Things to consider when configuring your monitor:

- Monitors for Pentium, 80486, 80386, and 80286 processors can be built to run in real or protected mode.
- If you are going to debug real-mode applications, indicate *Real*, no matter what your target CPU is.
- If you are going to debug protected-mode software, select *Protected*.
- Use one of the PC configurations if you are using an IBM PC or compatible motherboard. For all custom boards, use the *OTHER* configuration option.

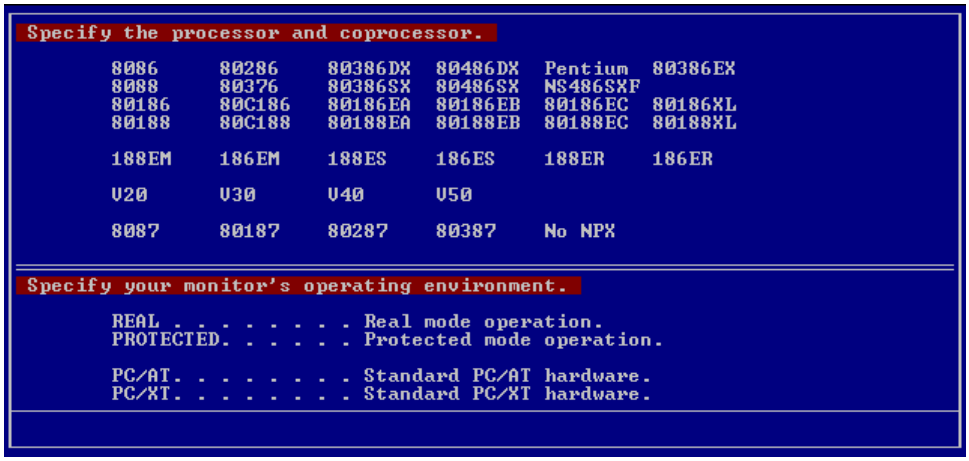

Figure 4-1: Target hardware options

Configuring Monitor Extensions

The User's Configuration Options make it easy to extend your CSi-Mon monitor.

The options offered by CSi-Config follow:

Name	Allows you to include information in the CSi-Mon sign-on string. This feature is helpful when keeping track of a series of monitors as they evolve, for example.
User (1..4)	Allows you to specify #defines and equates in **csicfg.h** and **csicfg.inc**. You can then use the #defines for any additional needs you have when extending the monitor.
Comments box	Allows you to put remarks in the include file.
Modules	Allows you to specify module names and paths.
Library	Allows you to specify additional libraries for linking to the monitor.

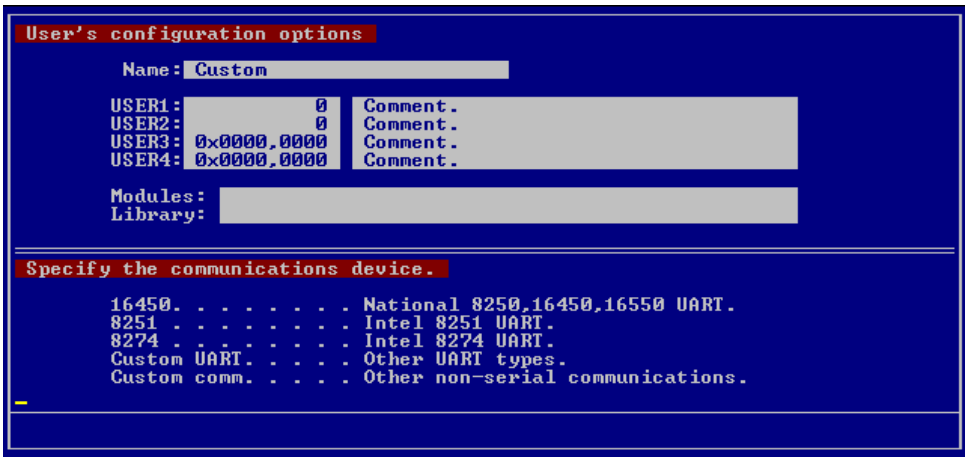

Figure 4-2: Monitor extension options

Communication Configuration

The topics below will help you configure your monitor's communication system. Each value in CSi-Config has a default value already entered.

You may need to consult the documentation for your target board to determine whether your target will support the default values. Refer to the vendor documentation of the part for the proper values.

CAUTION: CSi-Mon and Soft-Scope have been tested at 115200 baud. You can operate at this rate if your system can be configured to do so. However, systems running at high baud rates that do not have a National Semiconductor 16550-FIFO UART sometimes lose characters and cause Soft-Scope to malfunction.

Select a UART

If your board uses one of the standard UARTs listed in CSi-Config, the following tips apply:

- The National Semiconductor 16450 family of UARTs has internal baud-rate generators. Unless special circumstances require an external baud-rate generator for boards with one of these UARTs, select *None* in the baud-rate generator section.
- Default configuration values are based on PCs where applicable.

Polling Communication

Although we do not recommend it, you can configure the monitor for polled communications. When the monitor uses polled communications, Soft-Scope does not utilize many features (stopping the application while running, viewing variables while running, etc.).

However, using polled communications is an option when interrupt capabilities are not available (e.g., using Grammer Engines' PromICE).

Interrupt-Driven Communication

To stop target execution with the **Code/Stop** command or examine memory while the target executes, configure your monitor with interrupt-driven communication.

IBM and compatible PCs have two interrupt controllers. Custom boards may only have one. If your system has two interrupt controllers, selecting both the master/slave and slave configuration options, as shown below, enables your application to use the slave controller and Soft-Scope to use the master/slave system:

8259M Configure a master/slave or single PIC system
8259S Configure a slave PIC

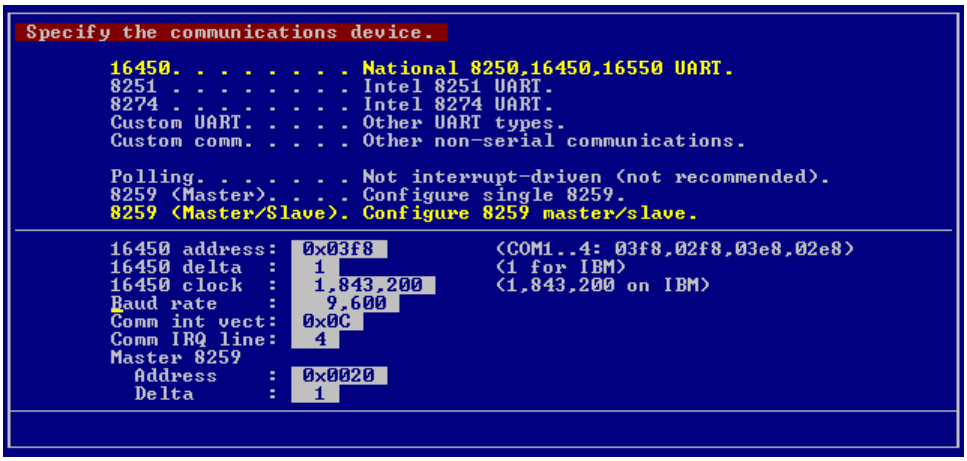

Figure 4-3: Communications options

National Semiconductor 16450/16550 UART

- You will need the base port address of the UART. If your target is a PC, the following values are valid:

COM1	0x03f8	COM3	0x03e8
COM2	0x02f8	COM4	0x02e8

- You will need the address delta between ports. To avoid having to multiplex a 16- or 32-bit data bus into an 8-bit peripheral, some hardware is addressed by ignoring low address lines. An 8-bit peripheral located on the lower data lines of a 16-bit bus uses a delta of 2. On a 32-bit bus, the delta is 4. Standard PC hardware maintains compatibility with 8-bit buses so the delta is 1 whatever the bus width is.
- Specify the UART's serial data clock input rate. For PCs, this value is usually 1843200.
- Specify the baud rate; be sure to use the same rate as configured for Soft-Scope.
- To determine the communication interrupt vector, add the IRQ and the base vector together. For example, if you are using COM1 and IRQ 4, and the base vector is 0x08, the interrupt vector would be 0x0C.

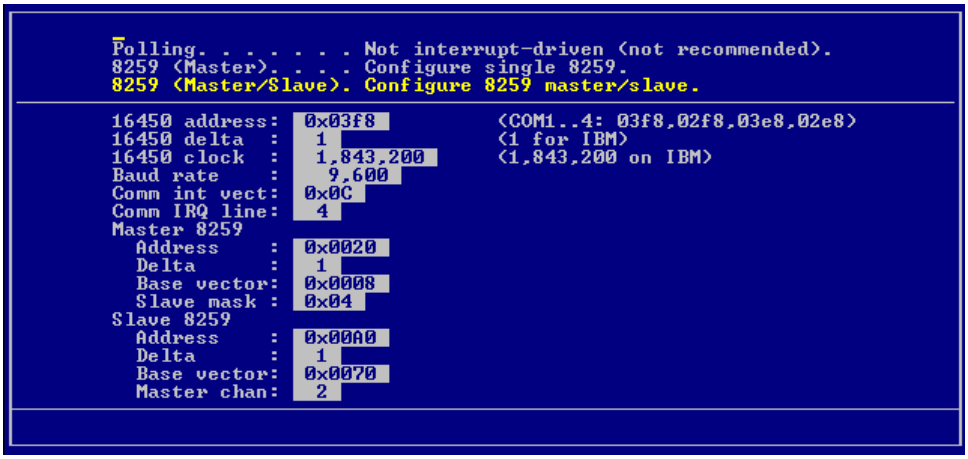

Figure 4-4: Communications system specifications, 16450 UART

Configuring Interrupt Controllers for National and Intel UARTs

To set up the master and slave Programmable Interrupt Controllers (PICs) you will need to know the following:

- Master and slave addresses
- Delta
- Base vector
- Slave mask
- Master channel

The values for PICs can be system specific. If your target is not a PC, see your hardware specifications to determine whether the default values will work.

Intel 8251 or 8274 UARTS

If the target is using Intel 8251 or 8274 UARTs:

- Know the base-port address of the UART.
- Know the address delta between ports. To avoid having to multiplex a 16- or 32-bit data bus into an 8-bit peripheral, some hardware is addressed by ignoring low address lines. An 8-bit peripheral located on the lower data lines of a 16-bit bus uses a delta of 2. On a 32-bit bus, the delta is 4. Standard PC hardware maintains compatibility with 8-bit buses, so the delta is 1 whatever the bus width is.
- Supply an acceptable input rate from an external source if the UARTs on the target does not have internal baud rate generators. Intel 8254 timers convert the rate of an external clock into an acceptable communication baud rate by dividing the clock rate by a number *n*. You do not need to know the value of *n*, but you must know which channel of the timer (0, 1, or 2) will be used.

- Know the base-port address of the Intel 8254 timer.
- Know the baud rate of the external clock that provides the timer with an initial value, and the baud rate you want the UART to use when communicating with the host computer. Be sure to specify the same rate as configured for Soft-Scope.
- Configure your Programmable Interrupt Controllers (PICs) as specified in the section *Configuring interrupt controllers for National and Intel UARTs*.

Specify MAPI Functions

The Monitor Application Programing Interface (MAPI) provides application I/O functions that communicate on the same serial channel Soft-Scope uses. The functions perform the following tasks:

CSIMON_CO	Console output. Writes a single character to CSi-Mon's communications channel. When Soft-Scope is present, output is queued for display in the Application I/O or Message windows.
CSIMON_CI	Console input. Reads a single character from CSi-Mon's communication channel. When the message, "Console input wait" displays at the top of the Code window, press <F10> to toggle the keyboard from Soft-Scope to application I/O mode before inputting from the keyboard. Press <F10> again to toggle back to Soft-Scope mode after inputting.
CSIMON_CP	Poll function. Tells the caller if data is available through CSIMON_CI.
CSIMON_MO	Message output. Sends messages to Soft-Scope. A message is a series of characters terminated by a zero. This is ignored if Soft-Scope is not present.

These routines are contained in an assembly module that you will need to include in your application. Although you may need to modify it slightly to meet your application's specific needs, we have included the files **mapi.asm**, **mapi.a38**, and the header file **mapi.h**, which can be found in the directory SSWIN\CSIMON\MAPI. Sample programs showing how to use these routines can be found in SSWIN\SAMP\MAPI86 and SSWIN\SAMP\MAPI386.

You can choose to run your application with or without Soft-Scope disabled:

X	Uppercase X, entered from Terminal mode (enabled by typing TERMINAL in the Command line dialog box) causes your application to run, but CSi-Mon doesn't know that Soft-Scope is present, so buffering is turned off.
Control-R	Tell CSi-Mon that Soft-Scope is running. Pressing any other key sequence will confuse CSi-Mon because it has no knowledge of Soft-Scope until after this command is given.

Selecting Tools

Select the tools you intend to use to build the monitor. See the chapter *Preparation* for tools we support. If you are using tools that are not listed, edit the batch file **build.bat** and/or the **makefile** to configure the monitor.

Although you can independently select the compiler, assembler, linker, locator, and librarian, all combinations are not guaranteed to work. Review the notes in the **readme.doc** on the installation diskettes for known combinations that cause problems.

You have the option of selecting the memory segmentation model you want to use when building a CSi-Mon monitor. The monitor does not have to be the same model as your application. Unless you plan to combine the monitor (as described in *Combining the Monitor and*

Your Application), it does not matter which model you use. Generally, small or flat is sufficient to build the monitor.

Also, know the path to your target's C library such as the following:

```
C:\msc\libs\
```

Locate the Monitor in Memory

Know the physical starting and ending addresses of the PROM and RAM portions of the monitor—this is important—you need to know the *physical* address. The values entered into CSi-Config are physical addresses. The comma is used only to separate the high word from the low word. It is important to locate the monitor in memory that is not used by your application. The reset segment is normally 64k below the top of the processor's addressable memory.

The starting RAM and ROM addresses should be paragraph aligned (16-byte). If you want to test the monitor on a PC before you burn it into a PROM, the following addresses should work:

- Starting RAM = 0x0000,0400
- Starting ROM = 0x0007,0000
- Reset address = 0x0007,FFF0

CAUTION: CSi-Mon should not be configured so that its data area is located below 0x0000,0400 in memory because it will overwrite the interrupt vector table. Soft-Scope uses interrupts to perform some of its commands, and locating CSi-Mon below 0x0000,0400 may cause you to lose functionality of the debugger.

For processors that use a peripheral control block and/or chip selects, CSi-Config provides a means to enter the appropriate values. The default values are those from popular evaluation boards. Refer to the vendor documentation of your processor to ensure the proper values are entered.

User Reference

CSi-Mon User’s Guide

User Reference

CSi-Mon User’s Guide

5. Combining the Monitor and Your Application

This chapter describes the mechanics of combining the monitor with your application. In the first section we describe the differences between using a library with your application and actually making changes to the monitor to merge it with your application. In the second section we explain the process of making the monitor a library and linking it with your application. In the last section we describe how to modify the protected-mode monitor source and merge it with a sample application.

5

Chapter Contents

When To Make CSi-Mon a Library and When Not To

There are two methods for combining the monitor with your application: combining your application with the CSi-Mon library and merging the source of CSi-Mon with your application. Which method you use depends on why you want to combine the monitor with your application and how you plan to install the final product.

Are you going to use the startup code CSi-Mon provides? Are you going to build a ROM or RAM based application? Your answers to these questions will help you decide the best method to use.

First, let's explain the library (see the chapter *Configuration* for creating a library). When the monitor is built as a library, it relies on your application for preparing the processor and peripherals. The only peripherals the library will configure are the UART and PIC. The library version of the monitor is easy to link to RAM based applications.

Many users develop the application and then wish to take advantage of CSi-Mon's startup code and port initialization. The library version of the monitor is not suited for ROM applications because of how the monitor is built. Therefore, if you want to make a ROMmable program, you do not want to make CSi-Mon a library. In this case, you might consider merging your application and CSi-Mon source.

Whether you are combining your application with the CSi-Mon library or CSi-Mon source, you should configure CSi-Mon to use interrupt-driven communications with Soft-Scope. This configuration allows Soft-Scope to talk to CSi-Mon while your application is running. Combining your application with CSi-Mon allows you to invoke Soft-Scope/CSi-Mon and inspect your application. If CSi-Mon is configured to use polling communications, Soft-Scope cannot send commands to CSi-Mon until your application stops. Additionally, if CSi-Mon is configured to use polling communications and no breakpoints are set, the monitor will never regain control of your application.

The following section describes how to combine the library version of the monitor and your application.

Combining Your Application with the CSi-Mon Library

This section describes how to combine your application with the CSi-Mon library (**csimon.lib**). Steps to guide you through the necessary changes are provided below.

Step 1. Configure and make the CSi-Mon library

Create a library version of the monitor.

Step 2. Modify your application

Your application has set up the processor and is ready to continue executing. The processor, peripherals, and application have completed initialization. Now it is necessary for you to manually initialize the monitor. This is done by placing a call to ***csimon()*** early in your application. To place a call to ***csimon()***, type the following after ***main()***:

```
main()
{
    csimon();
    .
    .
    .
}
```

Step 3. Initiate a break to evoke the monitor

Once the monitor has been initialized, you may want to immediately stop your application so that you can start debugging it. To do this, you need to create a separate file to stop your application.

brkexe.asm:

```
name    brkexe
include csicfg.inc     ; Generated with CSi-Cfg utility.
                       ; Let us handle generic use of code segment


@ISEG_BEG              ; Side affect of CSICFG.INC,
@ISEG_END              ; it assumes this is defined.


@CSEG_BEG              ; Start of CODE segment
public  _ss_brkexe
_ss_brkexe proc far    ; Debug break procedure.
    pushf
    pop ax
    or ax,100h         ; Set single step flag
    push ax
    popf               ; Cause a single step over ret
    ret                ; Return to caller.
_ss_brkexe endp


@CSEG_END              ; End of CODE segment
       end
```

Once you have the above assembly file included with your link, you can call ***ss_brkexe()*** to evoke the monitor.

```
main()
{
    csimon();
    ss_brkexe();
    .
    .
    .
}
```

Because of how ***ss_brkexe()*** works, when you invoke Soft-Scope (see the *Soft-Scope User's Guide*), it will be ready to debug at the instruction following the call. Now you are ready to debug your application.

Merging the Source of Your Application and CSi-Mon

This section describes how to combine a sample application with the CSi-Mon source. We recommend you attempt the merging of the source for CSi-Mon and the sample application before doing so on your application. We only cover how to make changes to the protected-mode monitor in this example. However, you should be able to derive the necessary changes for the real-mode version. Steps to guide you through the necessary changes are provided below.

Step 1. Adding forward references

Add forward references to the sample application's startup code in the file **main.c**.

main.c:

```
extern void app_start( void );
```

Step 2. Calling the sample application

Modify CSi-Mon source file **main.c** so the sample application will be called after the monitor has been initialized. Modifying the sample application also enables the monitor to interrupt the sample application after it has started. Note, for clarity, the following text is bold face for lines that need to be added and italicized for lines that need to be changed.

Update the XE_BOOT case statement in **main.c** (~line 343):

main.c:

```
case XE_BOOT:
     debug_mode = 1;
     which = BRK_RUNNING;
     si_pic(SI_PIC_ENABLE);
     sendctrl('*');
     for (i=0; csi_version[i]; i++)
          sendctrl(csi_version[i]);
          sendack();
          app_start();
          break;
```

Step 3. Creating a sample application in appmain.c

appmain.c:

```
void main( void )
{
     volatile int x;
     volatile int y;
     y = 0;
     while (1) {
          x = 999;
          y += x;
    };
}

void _main( void )
{
    main();
}
```

Some compilers may automatically add the leading underscore while others do not. From the assembler, we assume that ***main()*** will have a leading underscore added by the compiler. If the compiler does not add the leading underscore, it will call the second routine, which will then, in turn, call the entry point we wish. If leading underscores are generated, the second routine will never be called. Either way, we get to where we want. Of course, you can modify both the assembly and "C" sources to suit your specific needs.

Step 4. Override the check of task register

Before you begin this step, a quick discussion is needed about the *tr* register and how it relates to the code below. CSi-Mon only uses the *tr* register for the application it loads, not for itself. Therefore, CSi-Mon assumes that the *tr* register is zero. However when the monitor is combined with your application, Soft-Scope needs to know if the *tr* register is valid for it to debug your application. The code implemented by Step 3 circumvents this confirmation by Soft-Scope by checking that the *tr* register is zero and skipping the code. This all works provided the *tr* register is not modified.

Unfortunately, some flashloaders and other loaders jump into protected mode and back to real mode before executing CSi-Mon. When this occurs the flashloaders may change the *tr* register from a non-null value. Thus, when CSi-Mon starts running with the combined application, it may report to Soft-Scope "invalid tr". Since CSi-Mon was originally designed to accept downloaded programs instead of being combined with an application, encountering an invalid *tr* register has not been a problem. More customers are combining applications with the CSi-Mon source, so we are looking at alternatives for handling the *tr* register. None are available as of this writing.

Now that you understand how we interact with the *tr* register, it is time to override the check of task register. Open the file **pmonitor.asm** and find the "mov mon_fault, BRK_WAS_MONITOR" instruction in the procedure ***init_var*** (~line 1212). Change the instruction as shown below (again, we use italics to denote the line that needs to be changed):

pmonitor.asm:

```
;
;  Local variables.
;
    mov     mon_fault, 0
    mov     xe_nmi, 0
```

Also, find the “lldt” instruction in the procedure ***xe_cute*** (~line 1571) of the file **pmonitor.asm**. Add the text shown in bold below:

pmonitor.asm:

```
lldt   treg.reg_ldtr            ; Load ldtr.
cmp    reg_tr, 0
je     tr_skip2
mov    mon_fault, BRK_WAS_TR_BAD
mov    es, cs_sys_gdt
mov    di, reg_tr               ; es:di -> tss entry?
test   di, 4
jnz    tr_load                  ; Invalid tr selector.
and    di, 0fffch               ; Mask rpl bits.
mov    al, es:[di].attr
and    al, 10111B
cmp    al, 00011B
jne    tr_load                  ; It is not a busy TSS descriptor.
and    es:[di].attr, 11111101b  ; Clear busy bit.
tr_load:ltr     reg_tr
tr_skip2:nop
```

Step 5. Adding startup code

This step shows the modifications necessary to incorporate the sample application’s assembly language startup code into the CSi-Mon’s assembly language code. These changes let Soft-Scope recognize that it is already running a valid application when it gains control of the sample application.

As part of the typical C runtime startup code, constant data must be copied from flash or EPROM to RAM. Data must be initialized. The stack must also be initialized. Much of this data and stack initialization as part of the C runtime startup code has already been done to get CSi-Mon running. At this point, you need to create a separate stack for the sample application. To do this, you will create a separate startup module for initializing the sample application.

NOTE: The following description uses **csicfg.inc** to determine how to create the separate startup module. Thus, the following code uses macros that allow the separate startup module to work generically regardless what model or tool suite is used.

appstart.asm:

```
     name appstart
include csicfg.inc        ; Generated with CSi-Cfg utility.
                          ; Let us handle generic use of code segment


@ISEG_BEG                 ; Side affect of CSICFG.INC,
@ISEG_END                 ; it assumes this is defined.
@APP_STACK macro          ; Determine segment type for stack.
IF(@FLAT)
    APP_STACK segment para public use32 'STACK'
ELSE
    APP_STACK segment para public use16 'STACK'
ENDIF
    endm


@APP_STACK                ; Define Stack for the application to use
        dw 2048 dup (?)
app_tos              dw    ?
APP_STACK      ends
@CSEG_BEG                 ; Start of CODE segment
        extrn   _main:FAR
```

(Continued on next page)

```
        public app_start       ; Make sure "C" can find app_start with
        public _app_start      ; either leading "_" or not.


_app_start proc far            ; Stub with leading "_" just in case
_app_start endp
app_start proc far             ; Actual application initialization code
        mov     ax, APP_STACK
        mov     ss, ax
        mov     @SP, offset APP_STACK:app_tos
        sti
        call    main           ; Note, we call assuming with leading "_",
                               ; However, stub in appmain.c will catch it
                               ; if compiler doesn't generate them.
                               ; Otherwise, here is where the application
                               ; is executed!
        app_start endp
        @CSEG_END              ; End of CODE segment
                end
```

Step 6. Sharing memory space

CSi-Mon will not let Soft-Scope write to its memory space. However, when an application and CSi-Mon are combined they must share memory space. Make the changes described below to **memory.c** to allow memory sharing.

Look in **memory.c** for the function ***mem_log2lin*** (~line 623). Comment out the entire "if" statement block by entering the two lines that are in boldface. This code fragment, once modified, should look something like this (bold depicts lines that need to be added):

memory.c:

```
#if 0          /*#* Allow Soft-Scope to modify monitor.*#*/
if( write ) {
       /* Protect the monitor from requests that overwrite monitor
        ** memory or attempt to place breakpoints within monitor
        ** resources.
        */
if( (*linear + len - 1L) >= data_start && (*linear) <= data_end)
     return ME_PROTECT;  /* Attempt to overwrite monitor data.*/
if( (*linear + len - 1L) >= code_start && (*linear) <= code_end)
    return ME_PROTECT;  /* Attempt to overwrite monitor code.*/
if( (*linear + len - 1L) >= stack_start && (*linear) <=
    stack_end )
    return ME_PROTECT;  /* Attempt to overwrite monitor stack.*/


#if( MON_PROT && (MON_ROM || MON_ABS) )
        if( (*linear + len - 1L) >= reg_gdb && (*linear) <= (reg_gdb +
          GDT_LEN * 8L - 1L) )
            return ME_PROTECT;  /* Attempt to overwrite descriptors.*/
#endif

    }
#endif
```

Step 7. Build scripts

The final step requires modifications to the **build.bat** or **makefile** which affect the modules built and the generation of the locator command file (**csimon.cmd**). For this example, we will limit our discussion to the makefile and to the CSi-Loc/CSi-Link command script.

You need to make two sets of changes in **makefile**. The first set is to build modules *appstart* and *appmain* and link them to CSi-Mon.

Find the following lines in **makefile**:

```
#
#  Minimum object files necessary to create a monitor.
#
OBJECTS=pstart.obj pmonitor.obj common.obj main.obj register.obj break.obj \
              memory.obj xlate.obj pic.obj mapi.obj extend.obj siuart.obj
```

Add the two new object files for the sample application. Enter the text shown in bold as follows:

```
#
#  Minimum object files necessary to create a monitor.
#
OBJECTS=pstart.obj pmonitor.obj common.obj main.obj register.obj break.obj \
        memory.obj xlate.obj pic.obj mapi.obj extend.obj siuart.obj \
        appstart appmain
```

Next, it is necessary to add the new stack segment to the locator command script so that it is properly located. Look for the following:

```
echo ram DATA_FIRST DATA_LAST L2L_SRC L2L_DEST           >>csimon.cmd
echo locate DATA_FIRST class DATA class BSS              >>csimon.cmd
echo +      DATA_LAST L2L_SRC L2L_DEST::00000400P        >>csimon.cmd
```

Add the text shown in bold, changing the lines to read:

```
echo ram DATA_FIRST DATA_LAST L2L_SRC L2L_DEST APP_STACK >>csimon.cmd
echo locate DATA_FIRST class DATA class BSS               >>csimon.cmd
echo +    DATA_LAST L2L_SRC L2L_DEST APP_STACK::00000400P >>csimon.cmd
```

Finally, rebuild the newly combined application/monitor.

Common Pitfalls

1 The application runs, but Soft-Scope does not display the **Code Stop** menu item. If this menu item does not appear, you did not build an interrupt-driven monitor. CSi-Mon must be built as an interrupt-driven monitor to take full advantage of combining your application with the monitor (see the chapter *Configuration*).

2 The application runs, but Soft-Scope cannot stop the target (**Code Stop** and the **Stop Toolbar** button have no effect). Make sure you changed ***si_pic(SI_PIC_DISABLE)*** to ***si_pic(SI_PIC_ENABLE)*** as shown in Step 2 of *Merging the source of your application and CSi-Mon*. The Peripheral Interrupt Controller (PIC) must be enabled to allow Soft-Scope to communicate with CSi-Mon.

3 Soft-Scope reports monitor overwrite error messages. Normally, Soft-Scope will reject any attempt to read or write to the address space of the monitor. However, when you combine the monitor with your application, the monitor will think that attempts to read or write to your application's address space are attempts to access the monitor's address space. If you are getting monitor overwrite error messages, you should check to make sure you made the changes outlined in Step 6 of *Merging the source of your application and CSi-Mon*.

6. Troubleshooting

The information in this section may help you get the monitor running, if the monitor did not work after following the steps outlined in the chapters under *Getting Started*. Let's assume the batch file has been built and the monitor modules have been compiled and linked successfully, producing a **csimon.hex** or **csimon.abs** file.

If you still have no idea what is causing the problem, we suggest you start at the beginning of the configuration program and carefully check each value, one-by-one.

Chapter Contents

The most common causes of failure with the monitor are:

- Incorrect I/O addresses for communication device
- Incompatible baud rates
- Code configured for wrong target microprocessor
- Monitor not compiled correctly
- Monitor not located properly
- PROM access split on boundaries

Each of these causes are described in the following sections. Read on for more details.

Incorrect I/O Addresses for Communication Device

If your communication devices are not specified correctly, Soft-Scope and the target will not be able to communicate. Soft-Scope will probably display one of the error messages associated with this problem such as

```
Remote - target not responding
```

Have you tested Soft-Scope with a PC version of the monitor as described in the chapter *Demonstration*? If you have conducted the test and it worked fine, the problem is probably in how one of the following items is specified:

- The UART's timing mechanism, which varies according to the particular UART that your target has
- The base port address of your UART or timer device
- The address delta between ports

Check the specifications of each of these items and try running the monitor again.

Incompatible Baud Rates

If the communication between Soft-Scope and the target is garbled–there is no output or there are only random characters output from the target–make sure the baud rate of the host and the baud rate specified in the configuration program (see the chapter *Configuration*) are the same. Some hardware may have trouble operating at the maximum baud rate (115200), so it may be necessary to lower the baud rate both in the configuration program for CSi-Mon and the settings of Soft-Scope (see the *Soft-Scope User's Guide* for details on lowering the baud rate).

Code Configured for Wrong Target

If you have accidentally specified a CPU different than the target, the possible symptoms could range from no recognizable problem to a hung target. In some cases, the monitor will run without problems, and you may not even realize anything is wrong until you load an application. In other cases, the monitor may run until it tries to access a CPU-specific feature or structure that is not available. Sometimes, the target will simply crash.

The general rule is that 8086 and real-mode monitors will run on 80286/386/486/Pentium machines, 80286 monitors will run on 80386/486/Pentium machines, and 80386 monitors will run on 80486/Pentium machines. These requirements are not reversible. For example, 80386, 32-bit protected-mode monitors will not run on 80286 machines. Additionally, the 80186/188 monitors will run on 8086 machines, but, because the 80186/188 CPU requires that the

Peripheral Control Block be specified, 8086 monitors may not run on 80186/188 CPUs.

Monitor Not Compiled Correctly

If your monitor has unresolved symbols or error messages during compilation (such as output from tool suite or files generated from tools such as the map file), check the path you specified for the configuration program to access your tools library.

Another possibility is that one or more of the files on the source disk did not install properly, or was not included in the compilation. Find the file **contents.doc** on the installation diskettes that were provided with the software. This file contains the layout of the subdirectory tree and the files that reside in each subdirectory after installation is complete. Check to be sure that all files were installed properly on your computer.

Monitor Not Properly Located

If the monitor is not located properly in memory, it will be completely unpredictable. The monitor may crash when you try to install it, or it may appear to work fine until you load an application and discover the application is trying to load in the same memory as the monitor.

Inspect the final map file generated by your linker and locator to make sure the monitor is located correctly, and verify the following items:

- **For all monitors except library versions**

All data segments (except L2L_SRC and L2L_DST for protected mode monitors) should be contained between *DATA_FIRST* and *DATA_LAST* for all monitors except library versions.

CODE_FIRST and *CODE_LAST* should similarly bracket all code segments (except *RESTART*).

These segments are used by CSi-Mon to identify which memory ranges to protect from overwriting. The architecture of the processors does not provide a way to prevent your application from overwriting CSi-Mon, but we can detect and prevent downloading of an application over the monitor.

Since the library version of CSi-Mon shares the code and data segments with the application it is linked with, it does not have this protection.

- **For absolutely located monitors, which will have all segments contiguous and located in system RAM**

The restart code is not located at any special place for absolutely located monitors. It will be the responsibility of the loader to start execution at the correct address.

Absolutely located monitors are intended to be loaded into RAM by some custom method when your system boots.

- **For ROMmed monitors, which will be burned into ROM**

The public symbol *hardware_reset* should be located at your processor's hardware reset address.

ROMmed monitors will initialize their own data space. Data segments should be located in RAM.

- **For library versions of CSi-Mon that are intended to be linked to the application to be debugged and that are loaded into RAM with that application**

The library version provides no automatic initialization of its RAM area.

We recommend you don't use the library version with a ROM-based application. (See the chapter *Combining the Monitor and Your Application*).

- **PROM access split on boundaries**

It is almost impossible to predict what will happen if your PROM is not programmed correctly. Some code within the monitor may actually execute, but it may not function as expected. The most common mistake made when programming a PROM is to forget to split the hex file so it will work with 16- and 32-bit buses.

Most PROMs are 8 bits wide. To accommodate wider bus structures a program is normally split into separate PROMs. For example, on a 16-bit bus, bytes 0, 2, 4, 8 . . . are placed into the low-byte PROM and bytes 1, 3, 5 . . . are placed into the high-byte PROM. Your PROM programmer software should provide a method for splitting the file.

Appendix A: CSi-Mon Commands

This appendix contains the Monitor Command Syntax, the detailed information on the Monitor Commands, as well as related Command Tables.

Monitor Command Syntax

Command Syntax Elements

count An unsigned 32-bit integer. Specifies number of times to repeat.

dest_offset An unsigned 32-bit integer. If the segment value is 0, then the offset represents either a physical or linear address depending on the value last configured for the **'c'** command (see index number 7 in *Register Tables* in the Appendix). If the segment is nonzero, the offset represents a logical-address offset.

dest_segment An unsigned 32-bit integer. The segment portion of a logical address. If the segment value is 0, then the offset represents either a physical or linear address depending on the value last configured for the **'c'** command (see index number 7 in *Register Tables* in the Appendix). If the segment is nonzero, the offset represents a logical-address offset.

handle An unsigned 32-bit integer. Specifies an entry in the monitor breakpoint table.

index An unsigned 32-bit integer. Specifies an entry in a

	table.
length	An unsigned 32-bit integer. Specifies the size or number of data elements to be read from or written to the target.
offset	An unsigned 32-bit integer.
port	An unsigned 32-bit integer. Specifies an input or output port.
register_mask	An unsigned 32-bit integer. Specifies a register. See *Register Tables* in the Appendix for a list of registers and their masks.
segment	An unsigned 32-bit integer. Contains 0 if offset specifies a physical address.
src_offset	An unsigned 32-bit integer. If the segment value is 0, then the offset represents either a physical or linear address depending on the value last configured for the **'c'** command (see index number 7 in *Register Tables* in the Appendix). If the segment is nonzero, the offset represents a logical-address offset.
src_segment	An unsigned 32-bit integer. The segment portion of a logical address.
value	An unsigned 32-bit integer.

Monitor Commands

B Permanent breakpoint

B *[segment,offset,handle,count]*

The **'B**' command sets a permanent breakpoint at the *segment* and *offset* specified. *Handle* specifies an *index* into the monitor breakpoint table where information on this breakpoint is stored. **B** without parameters clears breakpoints.

Errors

5	ME_ARGS	Too few or invalid arguments specified
6	ME_RESOURCE	Breakpoint out of range
10	ME_SEGFAULT	Segmentation fault
11	ME_NOT_RAM	Location is not in RAM

C Initialize configuration table

C *address, config$_0$,config$_1$, . . ., config$_n$*

address Address of configuration table in monitor. Currently not used.

config$_x$ Field in configuration table. The entire table must be specified.

The **'C'** command initializes the configuration table. If you need to set just one entry in this table, use the **'c'** command.

Error

5	ME_ARGS	Too few or invalid arguments specified

c Set configuration table entry

c *index,value*

The **'c'** command assigns a *value* to the table entry specified by *index*. If you need to set several configuration table entries, use the **'C'** command. The indices and their values are listed in *Register Tables* in the Appendix.

Error

5	ME_ARGS	Too few or invalid arguments specified

D Data breakpoint

D *segment,offset,handle,count,type,length*

type An unsigned 32-bit integer. Specifies a breakpoint type.

The **'D'** command sets a permanent data breakpoint in memory at the *segment* and *offset* location specified. *Handle* specifies an index into the monitor breakpoint table where information on this breakpoint is stored. The *type* field specifies the kind of hardware breakpoint being set.

Possible types are listed in this Appendix and in **csimon.h** in the monitor source. The *length* field specifies the range of memory where you want the breakpoint set.

These codes cannot be changed without modification to Soft-Scope.

The monitor verifies that there are data registers available and returns an error (ME_RESOURCE) if it is unable to set the breakpoint. Refer to *Breakpoint Handle Tables* in the Appendix for information on data breakpoints.

NOTE: This command is useful only with the 80386/486/ Pentium protected-mode monitor.

Errors

5	ME_ARGS	Too few or invalid arguments specified.
6	ME_RESOURCE	One of several errors may have occurred. Either the handle is out of range, the length is too large (greater than 16) or there are not enough data registers available to set the breakpoint.
10	ME_SEGFAULT	Segmentation fault.

11	ME_NOT_RAM	Location is not in RAM.
12	ME_MISC	You have specified an illegal type or some other error has occurred.

Table A-1: Breakpoint types

Breakpoint Type	Code Value	Description
BKTYPE_CLEAR	0x0000	Clear breakpoint (per handle)
BKTYPE_MODIFY	0x0001	Set a breakpoint on address write
BKTYPE_RDONLY	0x0002	Set a breakpoint on data read
BKTYPE_ACCESS	0x0003	Set a breakpoint on data access
BKTYPE_FETCH	0x0004	Set a breakpoint on instruction

E Monitor extensions

E *[char] [data]*

char Any character

data User-defined data which includes a command name and parameters

The **'E'** command allows the monitor user to add commands to the monitor. *Data* can be any type desired. All of the *data* must be parsed by the monitor extension.

F Fill pattern in memory

F *segment,offset,length,value*

value Defines a byte

The **'F'** command fills a range of memory with a byte starting at the specified *segment* and *offset*.

Errors

5	ME_ARGS	Too few or invalid arguments specified
7	ME_VERIFY	If verification was requested, a byte read from memory did not match the value written
10	ME_SEGFAULT	Segmentation fault

f Return an offset

f *segment,offset,length,value*

value Defines a byte

The **'f'** command compares memory starting at the given address of *segment* and *offset* looking for the given byte *value*. If found, the offset from the starting address is returned.

Errors

5	ME_ARGS	Too few or invalid arguments specified
10	ME_SEGFAULT	Segmentation fault

I Port output

I *port,value*

The **'I'** command writes data to a *port*. The width depends on the current setting for granularity. See the **'c'** command and *Register Tables* in the Appendix.

Errors

5	ME_ARGS	Too few or invalid arguments specified
8	ME_BUSFAULT	Bus error

i Port input

`i` *`port`*

The **'i'** command reads data *value* from a *port*. The width depends on the current setting for granularity. See the **'c'** command and *Register Tables* in the Appendix.

Errors

5	ME_ARGS	Too few or invalid arguments specified
8	ME_BUSFAULT	Bus error

O Copy memory

`O` *`dest_segment,dest_offset,src_segment,src_offset,length`*

The **'O'** command copies memory starting at *src_segment* and *src_offset* to memory starting at *dest_segment* and *dest_offset* for length bytes.

Errors

5	ME_ARGS	Too few or invalid arguments specified
7	ME_VERIFY	If verification was requested, a byte read from memory did not match the value written
10	ME_SEGFAULT	Segmentation fault

o Compare memory

o *src_segment,src_offset,dest_segment,dest_offset,length*

The **'o'** command compares memory starting at *src_segment* and *src_offset* with memory starting at *dest_segment* and *dest_offset* for length bytes. The offset of the first byte from the starting address that does not match is returned.

Errors

5	ME_ARGS	Too few or invalid arguments specified
10	ME_SEGFAULT	Segmentation fault

Q Binary memory write

Q *segment,offset,datablock*

datablock A group of binary-encoded bytes, terminated with an end-of-block character (0x14)

The **'Q'** command writes the binary data in *datablock* to memory starting at the address specified by *segment* and *offset*.

Errors

5	ME_ARGS	Too few or invalid arguments specified
7	ME_VERIFY	If verification was requested, a byte read from memory did not match the value written

q Binary memory read

q *segment,offset,length*

The **'q'** command reads memory starting at the address *segment* and *offset* for length bytes. The data read is returned in binary format as a block.

Errors

5	ME_ARGS	Too few or invalid arguments specified
10	ME_SEGFAULT	Segmentation fault

R Write register

R *register_mask,value [,value]*

The **'R'** command writes a value to a register associated with the given *register_mask*. See *Register Tables* in the Appendix for valid mask values.

Errors

1	ME_PGM	Error in register_mask or value arguments
5	ME_ARGS	Too few or invalid arguments specified

r Read register

r *register_mask*

The **'r'** command reads a register associated with the given *register_mask*. The monitor returns the contents of each register specified. See *Register Tables* in the Appendix for valid mask values.

Errors

4	ME_ILLREG	Undefined register specified
5	ME_ARGS	Too few or invalid arguments specified

S Step an instruction

S *count*

The **'S'** command single steps *count* instructions.

Error

5	ME_ARGS	Too few or invalid arguments specified

s Stop target execution

s

The **'s'** command stops the target.

V Report version identification

v

The **'V'** command returns a number that is compared with a value stored internally in Soft-Scope.

Note: Bit 0x80000000L informs Soft-Scope whether the monitor is interrupt driven or not. If set, the monitor is interrupt driven, otherwise, the monitor is polled.

Error

5	ME_ARGS	Too few or invalid arguments specified

v Report version

v

The **'v'** command returns an ASCII string that identifies the monitor. The string is reported by Soft-Scope when connection is made with the monitor.

Error

5	ME_ARGS	Too few or invalid arguments specified

w Which breakpoint stopped target

```
w
```

The **'w'** command returns the handle of the breakpoint that was triggered. A negative handle returned indicates that either a special breakpoint was hit or the breakpoint was of unknown origin. See *Breakpoint Handle Tables* in the Appendix for a list of breakpoint handles.

Error

5	ME_ARGS	Too few or invalid arguments specified

x Execute the target with breakpoints

```
x
```

The **'x'** command starts the target running. This command assumes breakpoints are set and downloads breakpoints into memory. An error will be reported if an exception occurs during execution.

Error

5	ME_ARGS	Too few or invalid arguments specified

z Reset the monitor

```
z
```

The **'z'** command resets target registers and descriptor tables. Issuing this command causes the monitor to be reinitialized.

Error

5	ME_ARGS	Too few or invalid arguments specified

Command Tables

Table A-2: Execution commands

Command	Function Performed
B	Sets a permanent execution breakpoint at a memory location
D	Sets a permanent data breakpoint at a memory location
S	Executes one instruction at a time
w	Returns information describing which breakpoint was triggered
x	Executes the target software, breakpoints are set

Table A-3: Memory commands

Command	Function Performed
F	Fills a range in memory
f	Finds a byte in memory
I	Writes data to an I/O port
i	Reads data from an I/O port
O	Copies a block of memory to another location
o	Compares two blocks of memory
Q	Transmits data in binary format to the target and writes it to memory
q	Reads target memory and transmits data in binary form

Table A-4: Register commands

Command	Function Performed
R	Sets target CPU registers
r	Reads target CPU registers

Table A-5: Miscellaneous commands

Command	Function Performed
C	Initializes configuration table
c	Assigns a value to one configuration table entry
E	Monitor extensions
V	Reports version compatibility code
v	Reports version string for monitor
z	Resets the monitor

Appendix B: Register Tables

Table B-1: General purpose and tasking registers

Register	Mask	Applicable CPU
AX/EAX	0x00000004	All processors
BX/EBX	0x00000008	All processors
CX/ECX	0x00000010	All processors
DX/EDX	0x00000020	All processors
BP/EBP	0x00000040	All processors
SI/ESI	0x00000080	All processors
DI/EDI	0x00000100	All processors
CS	0x00000200	All processors
DS	0x00000400	All processors
SS	0x00000800	All processors
ES	0x00001000	All processors
FS	0x00002000	386/486/Pentium
GS	0x00004000	386/486/Pentium
Flags/Eflags	0x00008000	All processors
IP/EIP	0x00010000	All processors
SP/ESP	0x00020000	All processors
TR	0x00400000	Protected mode only

Table B-2: Protection controls and debug registers

Register	Mask	Applicable CPU
GDL	0x81000001	Protected mode only
GDB	0x81000002	Protected mode only
IDL	0x81000008	Protected mode only
IDB	0x81000010	Protected mode only
LDT	0x81000040	Protected mode only
MSW	0x81001000	Protected mode only
CR0	0x81001000	Protected mode, 386/486/Pentium only
CR2	0x81004000	Protected mode, 386/486/Pentium only
CR3	0x81008000	Protected mode, 386/486/Pentium only
DR0	0x81010000	386/486/Pentium
DR1	0x81020000	386/486/Pentium
DR2	0x81040000	386/486/Pentium
DR3	0x81080000	386/486/Pentium
TR6	0x81100000	Protected mode, 386/486/Pentium only
TR7	0x81200000	Protected mode, 386/486/Pentium only
DR6	0x81400000	386/486/Pentium
DR7	0x81800000	386/486/Pentium

Stack Register Masks

The coprocessor stack register masks are stored as 80-bit floating point values divided into three sections. For example, for ST0:

0x84000001 will access bits 0 through 31

0x84000002 will access bits 32 through 63

0x84000004 will access bits 64 through 79

These registers are applicable to systems with numeric coprocessors only.

Table B-3: Coprocessor stack registers

Register	Mask
ST0	0x84000001=0..31, 0x84000002=32..63, 0x84000004=64..79
ST1	0x84000008=0..31, 0x84000010=32..63, 0x84000020=64..79
ST2	0x84000040=0..31, 0x84000080=32..63, 0x84000100=64..79
ST3	0x84000200=0..31 0x84000400=32..63, 0x84000800=64..79
ST4	0x84001000=0..31, 0x84002000=32..63, 0x84004000=64..79
ST5	0x84008000=0..31, 0x84010000=32..63, 0x84020000=64..79
ST6	0x84040000=0..31, 0x84080000=32..63, 0x84100000=64..79
ST7	0x84200000=0..31, 0x84400000=32..63, 0x84800000=64..79

Table B-4: Miscellaneous coprocessor registers

Register	Mask	Applicable CPU
CW	0x83000001	Systems with numeric coprocessors
SW	0x83000002	Systems with numeric coprocessors
TW	0x83000004	Systems with numeric expressions

Table B-5: Configuration table entries

Index	Description	Values
0	Verify memory writes	0=off, other=verify
1	Echo mode	0=off, 3=echo character receive
4	Granularity	1=byte, 2=word, 4=dword
7	Linear/physical	0=segment:offset If segment value=0, address is physical address; other=segment:offset If segment value=0, address is linear address

Appendix C: Breakpoint Handle Tables

Table C-1: Breakpoint handles

Handle	Value	Description
BRK_WAS_STARTUP	0x0000	Monitor startup break
BRK_WAS_UNKNOWN	-0x0001	Must always be -1
BRK_WAS_RESET	-0x0002	Unscheduled target reset
BRK_WAS_TASK	-0x0003	Task switch caused break
BRK_WAS_DEBUG	-0x0004	Target tried to muck with DR regs
BRK_WAS_ILLACCESS	-0x0005	Reserved for future use
BRK_WAS_ILLWRITE	-0x0006	Reserved for future use
BRK_WAS_USER	-0x0007	User Stop command or cold boot
BRK_WAS_STEP	-0x0008	We were single-stepping
BRK_WAS_INT0	-0x00010	Divide error
BRK_WAS_INT1	-0x00011	Step
BRK_WAS_INT2	-0x00012	NMI
BRK_WAS_INT3	-0x00013	Hard-coded interrupt 3
BRK_WAS_INT4	-0x00014	Overflow
BRK_WAS_INT5	-0x00015	Bounds check
BRK_WAS_INT6	-0x00016	Invalid opcode
BRK_WAS_INT7	-0x00017	Coprocessor not available
BRK_WAS_INT8	-0x00018	Double fault
BRK_WAS_INT9	-0x00019	Coprocessor segment overrun

Table C-1: Continued

Handle	Value	Description
BRK_WAS_INTa	-0x001a	Invalid TSS
BRK_WAS_INTb	-0x001b	Segment not present
BRK_WAS_INTc	-0x001c	Stack exception
BRK_WAS_INTd	-0x001d	General protection fault
BRK_WAS_INTe	-0x001e	Page fault
BRK_WAS_INTf	-0x001f	Interrupt 15
BRK_WAS_INT10	-0x0020	Floating point error
BRK_WAS_INT11	-0x0021	Alignment error
BRK_WAS_LDTR_BAD	-0x00110	Invalid LDTR value
BRK_WAS_TR_BAD	-0x00111	Invalid TR value
BRK_WAS_CS_BAD	0x00112	Invalid CS:IP/EIP value
BRK_WAS_SS_BAD	0x00113	Invalid SS:IP/ESP value
BRK_WAS_DS_BAD	0x00114	Invalid DS value
BRK_WAS_ES_BAD	0x00115	Invalid ES value
BRK_WAS_FS_BAD	0x00116	Invalid FS value
BRK_WAS_GS_BAD	0x00117	Invalid GS value
BRK_WAS_STACK	0x00118	Stack too small for context switch
BRK_WAS_TSS_BAD	0x00119	Invalid TSS contents

Appendix D: CSi-Boot Error Messages

D

```
< Absolute OMF86 not supported - Use OH86 to generate hex file. >
```

OMF86 is not supported.

An object to HEX converter, such as the OH86, LinkLoc, CSi-Locate, or CSi-Link™ with the HEX or -HEX32 switch, can be used to create a CSi-Boot loadable HEX file.

```
< Address ######## - Not writable RAM >
```

The given address is not writable. Either it is not a RAM location or the RAM is defective.

Check your target documentation to make sure the RAM you specified in CSi-Config is valid.

```
< Filemark ######## - Invalid record >
```

The record starting at the given filemark was found to be invalid.

```
< Filemark ######## - Unexpected end of file >
```

An unexpected end of file was encountered in the record located at the specified filemark.

```
< Line: #### - Expected end of file >
```

When CSi-Boot tried to boot your **monitor.hex** file, it encountered the end of the file but not a termination record.

```
< Line: #### - Invalid hex file >
```

The given line number in your **monitor.hex** file contains an invalid record.

```
< Line ### - Line too long >
```

The given line in your **monitor.hex** file is too long to process.

```
< Load image above real-mode addressing range >
```

CSi-Boot can only load applications in the real-mode addressable range.

```
< Load image spans more than #### bytes >
```

CSi-Boot loads from a contiguous buffer. Either your monitor is too large to accommodate, or ROM and RESET must be located closer together.

```
< No data loaded >
```

The file you specified on the CSi-Boot invocation line appeared to be empty.

```
< No scratch memory found >
```

CSi-Boot was unable to locate 64K of scratch pad memory.

```
< System: File does not exist:  “filename” >
```

The file you specified when you invoked CSi-Boot cannot be found.

D

```
< Unknown file format >
```

CSi-Boot does not understand the format of the file you entered on the invocation line. The supported formats are Intel 8086/386 **.hex** and 80286/386 absolutely located.

```
< Usage: CSiBOOT filename >
```

CSi-Boot requires either an Intel 8086/386 **.hex** or 80286/386 absolutely located file to load.

Index

F indicates figure reference

T indicates table reference

I

C

D

E

F

G

H

I

L

M

N

P

R

S

T

U

V