Freescale In-Circuit Emulator Base

User Manual

Rev. 1.1
1/2005

FSICEBASEUM

How to Reach Us:

USA/Europe/Locations Not Listed:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

Japan:
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
852-26668334

Home Page:
www.freescale.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Learn More: For more information about Freescale products, please visit www.freescale.com.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2005. All rights reserved.

Table of Contents

1 Overview 11

System Requirements 11
System Features 12
System Components 13
Basic Components. 13
Additional Components. 14

2 Getting Started 15

Setting Up The Development System. 15
Setting Up The Hardware 15
Installing CodeWarrior Software. 17
Creating a Project 20
Adding Files 26
Building (Compiling) a Project. 27
Establishing Communication 28
Establishing Communication Through Ethernet Port 28
Establishing Communication Through USB Port 29
Starting the Debugger 30

3 Configuring The System 35

Setting Up The System 35
Specifying A Target. 35
Specifying Communication Information. 36
Assigning An IP Address To The FSICEBASE 38
Specifying a Memory Map 40
Specifying the Clock Speed 42
Loading Program In Debugger 44
Setting Up Logic Cables And Connectors 44
Modifying Values In Memory. 46
Using Log Files. 47

4 Using The MON08 Debug Port 49

Setting Up Hardware For the MON08 debug port 49
Establishing Communication Through The MON08 Debug Port. 50

5 Using the Debugger 55

Starting The Debugger 55
Changing CPU Register Values. 56
Using the Register Window to Change Register Values 56
Using the Command Line to Change Register Values. 57
Viewing Memory. 57
Resetting The Emulation System 59
Using Breakpoints And Watchpoints 60
Setting Breakpoints on Lines of Code. 61
Setting Breakpoints on Addresses 63
Controlling Program Execution 65

6 Using the Bus State Analyzer 67

Overview 67
Using the Bus State Analyzer 68
Defining Events 68
Recording Modes. 70
Continuous: all cycles 70
Continuous: events only. 70
Counted: all cycles. 70
Counted: events only 70
Sequential: A+B+C+D. 71
Sequential: A+B -> C+D 71
Sequential: A -> B -> C !D 71
Sequential: A -> B -> C -> D. 71
Sequential: Nth event: A+B+C+D 71
Time Tag Clock Frequency 72
Collecting Bus Data. 72
Viewing Data 73

A S-Record Format 75

S-Record Content 75
S-Record Types 76
Creating S-records 77
Example 77

B FSICEBASE Debugger Scripting Commands 79

Command Reference 79

Index 81

Table of Contents

About This Book

This manual is for developers, testers, application engineers, and anyone interested in using the Freescale In-Circuit Emulator Base (FSICEBASE) development system.

Organization

- **Introduction**

  An overview of the Freescale In-Circuit Emulator Base (FSICEBASE) and development system.
- **Getting Started**

  Explains basic procedures to get started using the FSICEBASE and development system.
- **Configuring The System**

  Explains how to configure the software and hardware.
- **Using The MON08 Debug Port**

  Explains how to MON08 debug port of the FSICEBASE.
- **Using the Debugger**

  Explains how to use the debugger to execute and analyze code. Explains how to perform debugging tasks such as stepping through code, changing values in registers and memory, and viewing data in memory.
- **Using The Bus State Analyzer**

  Explains how to use the Bus State Analyzer (BSA). Explains how to define events, capture data, and view captured data.
- **Appendix A: S-Record Format**

  Describes the format of the S-Record file.
- **Appendix B: FSICEBASE Debugger Scripting Commands**

  Lists commands only used with FSICEBASE.

Revision History

The following table summarizes revisions to this document.

Revision History

Location	Revision
Metrowerks, Austin, TX	Rev. 1.0 (Initial publication)
Metrowerks, Austin, TX	Rev. 1.1 (Content update)

Suggested Reading

Application Note: Getting Started with HCS08 and CodeWarrior Using C, AN2616, 11/2003.

Motorola HC08 CPU Awareness and True-Time Simulation, Metrowerks, 2004.

CodeWarrior™ Development Studio IDE 5.5 User's Guide, Metrowerks, 2004.

Conventions

This document uses the following notational conventions:

- Text in `Courier` monospaced type indicates commands, command parameters, code examples, expressions, datatypes, and directives.
- Text in *italic* type indicates replaceable command parameters.
- Text in **bold** type indicates names of labels and menu items on-screen.

Definitions, Acronyms, and Abbreviations

The following list defines the abbreviations used in this document.

BSA	Bus State Analyzer
EM	Emulator Module
FSICEBASE	Freescale In-Circuit Emulator Base
LAN	Local Area Network

MCU	Microcontroller Unit
MON08	Monitor for HC08 MCUs

1

Overview

This manual contains information about how to use the Freescale In-Circuit Emulator Base (FSICEBASE). The FSICEBASE is a tool that helps you develop applications for embedded systems based on MC68HC08 microcontroller units (MCU).

NOTE The Freescale In-Circuit Emulator Base (FSICEBASE) is similar to a system formerly produced by Motorola called the MMDS05/08 Motorola Modular Development System (MMDS). If you worked with the MMDS05/08, many of the features of the FSICEBASE will be familiar to you.

This chapter contains the following sections:

- System Requirements
- System Features
- System Components

The FSICEBASE works with CodeWarrior software, which is produced by Metrowerks, Inc. The CodeWarrior software provides an integrated development environment that includes an editor, assembler, and a user interface to the FSICEBASE system.

The environment allows you to perform source-level debugging. The CodeWarrior software also simplifies the process of managing and building a software project, and debugging code for an embedded MCU system. The benefit to you is reduced development time.

System Requirements

The FSICEBASE system requires a host computer with the following minimum specifications:

- Processor: 200 MHz Pentium® II processor or AMD-K6® class processor
- Operating System: Microsoft® Windows® 2000/XP
- RAM: 128 MB
- Hard drive space: Compact software installation: 232 MB
  Full software installation: 344 MB
- USB port or Ethernet port to connect host computer to the FSICEBASE

System Features

The Freescale In-Circuit Emulator Base (FSICEBASE) is a full-featured development system that provides in-circuit emulation. Features include:

- Real-time, non-intrusive, in-circuit emulation
- Allows you to set four complex data or instruction breakpoints; a breakpoint can be qualified by an address, an address range, data, or externally connected logic clips.
- Up to 128k real-time variables (any ROM or RAM memory area)
- Up to 128 Kbytes of emulation memory to accommodate the largest available ROM size of current HC08 MCUs
- Unlimited hardware instruction breakpoints and data watchpoints over up to 128-K memory map (conditional or unconditional)
- Built-in real-time bus state analyzer:
  - Large 1.33M x 92-bit real-time trace buffer - capture up to 24-bit address, 8-bit data, 32-bit time tag, instruction bit, R/W bit, extended access bit, trigger, and 24-bits of external clip information
  - Four hardware triggers for controlling real-time bus analysis or to provide additional complex breakpoints
  - Nine triggering modes
  - Display or dump real-time trace data as raw data, disassembled instructions, raw data and disassembled instructions, or assembly-language source code
  - As many as 1.33M pre- or post-trigger points
  - Trace buffer can be filled while single-stepping through user software
  - Large 32-bit time tag
  - Custom time tag clock from 4100Hz to 40MHz in 5kHz steps or from an external source, permitting wide time variance between analyzer events
  - Up to 24 general-purpose logic inputs, 5 of which can be used to trigger the bus state analyzer
- Custom MCU clock from 4100Hz to 40MHz in 5kHz steps or from external source
- Command and response logging to disk files
- C/C++ and/or assembly-language source-level debugging
- On-screen, context-sensitive help via pop-up menus and windows
- Emulation that allows multiple types of reset
- Meets ECC92 European electromagnetic compatibility standards

System Components

The FSICEBASE system includes the basic components that you need to use with an emulation module (EM) and target cable/adapter assemblies. You can separately purchase these and other additional components that will complete or enhance your debugging and emulation debug system.

Basic Components

The Freescale In-Circuit Emulator Base product includes the following components

- Base station

  The connectors on the top of the box let you connect an emulation module (EM).
- Cables, connectors, and adapters:
  - Crossover Ethernet cable (connects directly to an Ethernet Network Interface Card (NIC) on a PC)
  - Straight-through Ethernet cable (connects to a hub or switch)
  - Universal Serial Bus (USB) cable
  - MON08 port cable (16-pin ribbon cable)
  - External power supply (+5VDC 3.7A regulated) and universal plug adapters
  - Bus state analyzer logic clip cable assemblies: twisted-pair cables and hooks that connect the station module to your target system. You can also use the cable assembly to connect a test fixture, a clock, or any other circuitry that you might use to perform analysis. One end of each cable assembly has a molded connector, which fits into the FSICEBASE. Leads at the other end of each cable terminate in female probe tips. Hooks come with the cables.
- System software

  CodeWarrior software, featuring an editor, C/C++ compiler, assembler, and C/C++ and/or assembly source level debugger
- Documentation:
  - Freescale In-Circuit Emulator Base User Manual (this manual)
  - CodeWarrior User's Manual (online)
  - Freescale In-Circuit Emulator Base Quick Start
  - Online Help and PDFs

Additional Components

You can purchase other components to complete and/or enhance your development efforts:

- An emulation module (EM) - required for emulation, but not for MON08 debug

  An emulation module (EM) is a printed circuit board that emulates the features of a specific set of microcontroller units (MCUs). An EM completes the functionality of the FSICEBASE for a particular MCU or MCU family and is required to use the FSICEBASE emulation features. The FSICEBASE works with a variety of EMs. You can purchase EMs separately from the FSICEBASE.

  The two connectors on the bottom of the EM fit into one of the sets of connectors on the top of the FSICEBASE box. The connections provides power and signal connections.

  Connection to your target system is then made through a separately-purchased target cable, target head adapter, and footprint adapter that attaches to a target connector located on the top of the EM board.

- Optional target cable

  You can separately purchase a target cable that is part of a cable assembly which is used to connect a target system to the FSICEBASE.

- Optional target head adapter

  You can separately purchase a target head adapter that is part of a cable assembly which is used to connect a from a footprint adapter to the target cable.

- Optional footprint adapter
  You can separately purchase a footprint adapter that connects to the target system and plugs into the target head adapter.

- Optional Bus State Analyzer (BSA) cables

  The base station contains ports for three BSA pods. You can purchase BSA cables in addition to those supplied with the FSICEBASE system.

- Optional programming adapter
  A programming adapter is a board that provides a MON08 connector and several sockets you can use to program the flash of MCUs in particular packages.

2

Getting Started

This chapter explains how to set up the development system. This chapter contains the following sections:

- Setting Up The Development System
- Creating a Project
- Adding Files
- Building (Compiling) a Project
- Establishing Communication
- Starting the Debugger

Setting Up The Development System

The Freescale In-Circuit Emulator Base (FSICEBASE) development system includes basic connection cables and software. You need to connect the cables and install the software in order to use the FSICEBASE. This section contains the following topics to help you set up the system:

- Setting Up The Hardware
- Installing CodeWarrior Software

Setting Up The Hardware

This section explains how to connect a host computer to the Freescale In-Circuit Emulator Base (FSICEBASE). There are three ways to connect a host computer to the FSICEBASE:

- directly from the USB port of a host computer to the FSICEBASE USB port
- directly from the Ethernet port of a host computer to the FSICEBASE Ethernet port (crossover connection)
- from the host computer, through a Local Area Network (LAN), to the FSICEBASE Ethernet port (straight-through connection)

To connect the FSICEBASE to a host computer:

1. If you are using an Ethernet connection to connect your host computer to the FSICEBASE through a Local Area Network (LAN):
   a. Connect host computer to LAN
   b. Connect FSICEBASE to LAN
      - Make sure power supply is not connected to FSICEBASE
      - Connect one end of Ethernet cable to Ethernet port of FSICEBASE (make sure to use the straight-through Ethernet cable when connecting to LAN)
      - Connect other end of Ethernet cable to Local Area Network (LAN)

NOTE To complete the connection through a LAN, you will need to obtain the IP address, subnet mask, and default gateway information from your network administrator. You will use this information in a later step.

2. If you are using an Ethernet connection to connect your host computer directly to the FSICEBASE (not through a LAN):
   - Make sure power supply is not connected to FSICEBASE
   - Connect one end of Ethernet cable to Ethernet port of FSICEBASE (make sure to use the cross-over Ethernet cable when connecting directly to a Network Interface Card (NIC))
   - Connect other end of USB cable to host computer

NOTE The host computer (PC) must have an assigned IP address and a subnet mask that matches the FSICEBASE.

3. If you are using a USB connection to connect your host computer directly to the FSICEBASE:
   - Make sure power supply is not connected to FSICEBASE
   - Connect square-shaped end of USB cable to FSICEBASE
   - Connect other end of USB cable to host computer

NOTE You must first install the CodeWarrior software for your PC to properly recognize the FSICEBASE USB device.

4. Connect Power supply to FSICEBASE
   a. Connect round end of 5-volt power cord to barrel connector on FSICEBASE
   b. Plug power supply into surge-protected strip
   c. Connect surge-protected strip to AC outlet

5. Switch Power switch to on

   Power and ready LED lights after the base station finished start-up sequence.

There are three status LEDs on the box: busy, ready, and error. The FSICEBASE base station takes about 5-10 seconds to start up. After powering the unit, you must wait for the ready LED to light before attempting to communicate.

The FSICEBASE is now ready to accept communication with a host computer. You will need to install the CodeWarrior software, create a project, and start the debugger to establish communication between your host computer and the FSICEBASE.

Installing CodeWarrior Software

The software portion of the development system consists of two parts. One part is the CodeWarrior Development Studio for HC(S)08 v3.1, which does not contain support for the FSICEBASE. The other part is a software patch that adds support to the CodeWarrior software Development Studio for HC(S)08 v3.1 for the FSICEBASE. This software patch is only required for CodeWarrior fDevelopment Studio for HC(S)08 v3.1. The following procedure explains how to install both pieces of software.

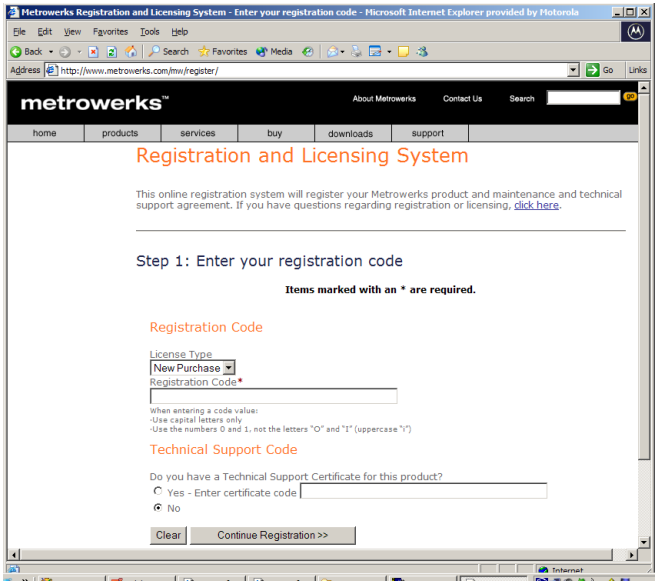
To install the CodeWarrior software on your host computer:

1. Install CodeWarrior IDE
   a. Insert **CodeWarrior Development Studio** CD into CD-ROM drive — CW Auto Install begins

NOTE If Auto Install does not start, run launch.exe, which is located in the root directory of the CD.

   b. Follow setup program's on-screen instructions
2. Restart your computer — operating system reboots which ensures that CodeWarrior IDE finds newly installed drivers
3. Launch CodeWarrior IDE
   a. Select **Start > Programs > Metrowerks CodeWarrior > CW08 V3.1** — menu appears
   b. Select **CodeWarrior IDE** — IDE starts; main window appears
4. Register CodeWarrior software

Figure 2.1 Step 1 of the Register Form



a. Select **Help > Register Product** from IDE main menu — CodeWarrior IDE starts browser, takes you to Step 1 of Metrowerks' on-line registration form (Figure 2.1)

b. Enter your registration code in appropriate field

NOTE If you downloaded the software from the Metrowerks web site, you might not have a registration code. You can request a registration code from one of these addresses:
Americas, Asia: `license@metrowerks.com`
Europe, Africa: `license_europe@metrowerks.com`
Japan: `j-license@metrowerks.com`

c. Click **Continue Registration** button — Step 2 appears

d. Follow on-screen instructions to complete remaining pages of form (Thank You page is last) — within a few minutes Metrowerks emails your license authorization code

NOTE You do not need to wait for your license authorization code to begin using your CodeWarrior development studio. Your temporary license lets you create the sample project in demo mode.

5. Obtain license key

a. From email message you receive from Metrowerks, copy license authorization code

b. Start CodeWarrior IDE

c. From CodeWarrior main menu bar, select **Help** > **License Authorization** — **Licence Authorization** dialog box appears

d. Paste license authorization code into **License Authorization** dialog box

e. Click **OK** button — **License Authorization** dialog box updates; Metrowerks emails you copy of new license key

f. Copy license key from **License Authorization** dialog box to clipboard, or wait for email from Metrowerks

g. Click **OK** button — **License Authorization** dialog box closes

h. From IDE main menu bar, select **File** > **Exit** – IDE closes

6. Install license key

a. Open license.dat — you can use a plain text editor such as Notepad to open `license.dat` file

NOTE You can find the `license.dat` file in the directory where you installed the CodeWarrior software. The directory default is:
`C:\Program Files\Metrowerks\CW08 V3.1`

b. Copy license key you received from Metrowerks

c. Paste license key on new line at bottom of `license.dat` file

d. Save `license.dat` file

e. Close `license.dat` file – license is installed; IDE uses new license next time you start the CodeWarrior IDE

NOTE Do not move or delete the `license.dat` file. If you receive additional keys in the future for other CodeWarrior components, you can add the additional keys to the `license.dat` file.

7. Install software patch

a. Insert the software service pack CD into your CD-ROM drive or download the service pack from http://www.metrowerks.com.

b. Run the installation executable and follow the on-screen instructions to install the service pack to the directory where you installed the CodeWarrior software. The default installation directory for CodeWarrior Development Studio for HC(S)08 v3.1 is:

`C:\Program Files\Metrowerks\CW08 V3.1`

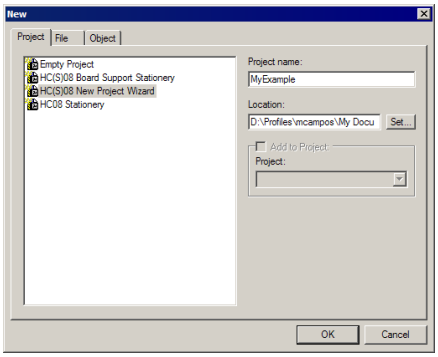
After you have installed the CodeWarrior software and license, you need to establish communication between your host computer and the FSICEBASE. You can perform application development with the CodeWarrior software without the connection, but you will not be able to use the features of the FSICEBASE until you establish communication.

Creating a Project

To create a project in the CodeWarrior IDE:

1. Launch CodeWarrior IDE
   a. Select **Start > Programs > Metrowerks CodeWarrior > CW08 V3.1** — menu appears
   b. Select **CodeWarrior IDE** — IDE starts, and CodeWarrior window appears
2. From IDE main menu bar, select File > New – New window appears (Figure 2.2)

Figure 2.2 New Window in the CodeWarrior IDE

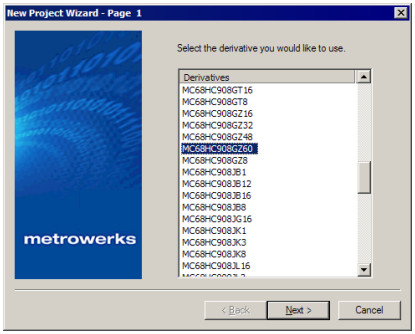


3. Create new project

NOTE You can use the New Project Wizard, the Board Support Stationery, or the HC08 Stationery to create a project. This procedure shows you how to use the New Project Wizard. We use an MC68HC908GZ60 target as an example.

   a. Select **HC(S)08 New Project Wizard**
   b. In **Project name** text box, type name you want to give project – IDE automatically adds `.mcp` extension when it creates project
   c. Click **OK** button — first page of New Project Wizard appears

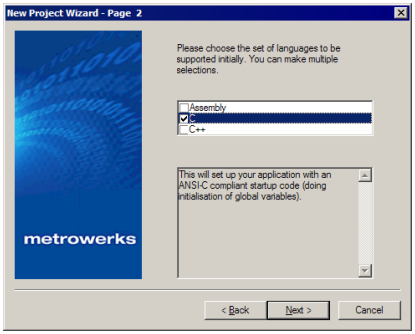
Figure 2.3 New Project Wizard - Page 1



d. Select **MC68HC908GZ60**

e. Click **Next** button — Page 2 of New Project Wizard appears

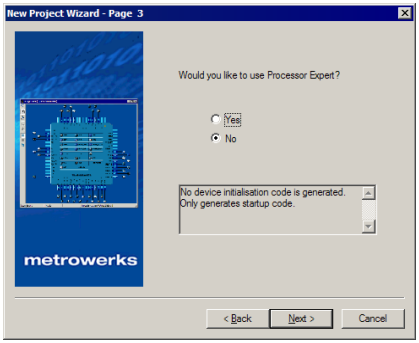
Figure 2.4 New Project Wizard - Page 2



f. Make sure **C** checkbox is marked

g. Click **Next** button — Page 3 of New Project Wizard appears; allows you to specify whether you want project configured to use Processor Expert

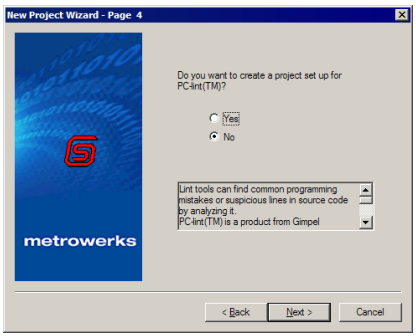
Figure 2.5 New Project Wizard - Page 3



h. Select **No**

i. Click **Next** button — Page 4 of New Project Wizard appears; allows you to specify whether you want project configured to work with PC-lint

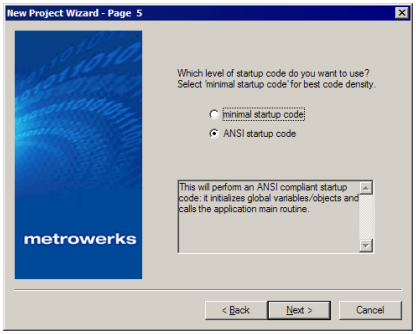
Figure 2.6 New Project Wizard - Page 4



j. Select **No**

k. Click **Next** button — Page 5 of New Project Wizard appears; allows you to specify the code that the Wizard will place in the project as the startup code

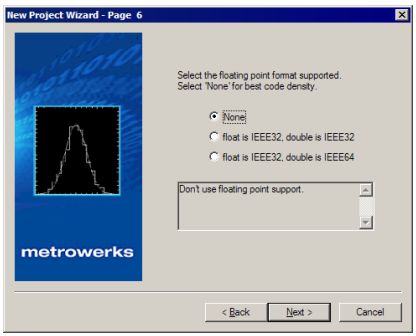
Figure 2.7 New Project Wizard - Page 5



l. Select **ANSI startup code**

m. Click **Next** button — Page 6 of New Project Wizard appears; allows you to specify floating point format that project should be configured to support

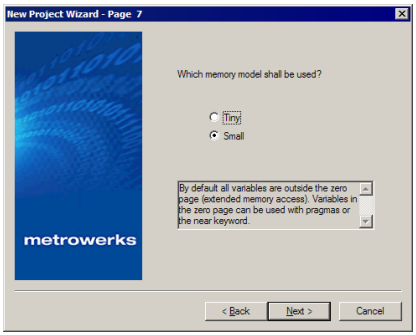
Figure 2.8 New Project Wizard - Page 6



n. Select **None**

o. Click **Next** button — Page 7 of New Project Wizard appears; allows you to specify memory model that project should be configured to support

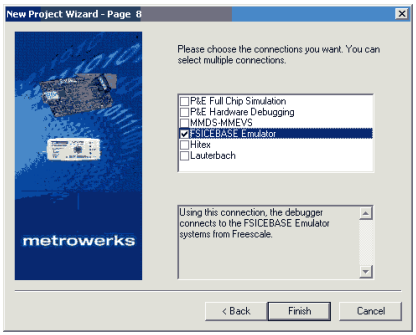
p. New Project Wizard - Page 7



q. Select **Small**

r. Click **Next** button — Page 8 of New Project Wizard appears; allow you to specify connections that project should be configured to support; Wizard creates a separate build target for each connection that you select

Figure 2.9 New Project Wizard - Page 8

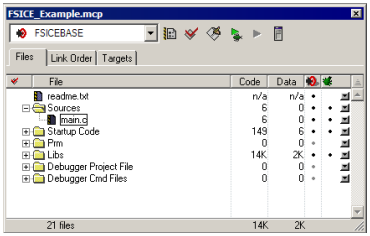


s. Check **FSICEBASE Emulator** checkbox

NOTE If you want your project to include a build target that connects through the MON08 target interface, select **P&E Hardware Debugging**. You can select more than one connection.

t. Click **Finish** button — system creates new project based on information you specified in New Project Wizard; Project window appears, docked at left side of main window

Figure 2.10 Project Window



NOTE To undock Project window, double-click the docking handle (double gray lines at top of the Project window). To re-dock window, right click in title bar of Project Window, and select **Docked**.

The project is ready for your use. You can add build targets, edit source files, modify build options, add files, and perform other development tasks. The following steps explain how to perform some of these tasks.

For more complete information on how to perform all development tasks with the IDE, see *The CodeWarrior IDE User Guide*.

4. Select build target

You can define multiple build targets in a single project. To use the build target that connects to the FSICEBASE, select FSICEBASE.

   a. In Project window click drop-down menu

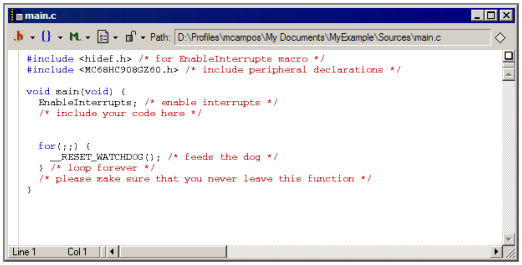
   b. Select **FSICEBASE**

5. Edit source code

   a. Click + sign next to `Sources` folder — tree expands

   b. Double click **main.c** – Editor window opens displaying contents of `main.c` file

Figure 2.11 `main.c` in Editor Window



   c. Make changes to contents of `main.c` file if desired

d. If you make changes to `main.c` file, from IDE main menu bar, select **File > Save** – IDE saves changes

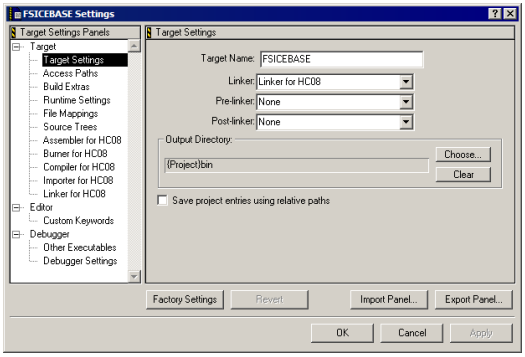
6. Specify build options

   a. From IDE main menu, select **Edit**

   b. Select **FSICEBASE** Settings – Target Settings panel appears (Figure 2.12)

NOTE The name of the build target determines the name of the menu item. If FSICEBASE is not the name of the current build target, a different menu item appears under the Edit menu, in the form *build_target_name* Settings.

Figure 2.12 Target Settings Panel



   c. From left hand pane, select **Compiler for HC08** – right hand pane displays Compiler for HC08 panel

   d. In Compiler for HC08 panel, specify options that you want the compiler to use

   e. Click **Apply**

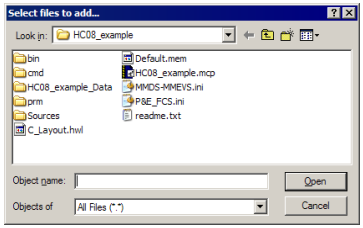
   Compiler uses the options that you specified the next time that you build the project.

Adding Files

To add files to a project:

1. In Project window, Highlight `Sources` folder
2. From IDE main menu bar, select **Project** — menu appears
3. Select **Add Files** — dialog box appears (Figure 2.13)

Figure 2.13 Add Files Dialog Box



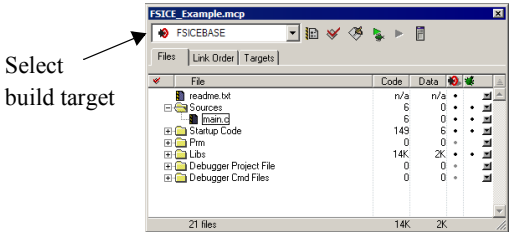
4. Navigate to directory that contains file you want to add
5. Select (highlight) filename of file you want to add to project
6. Click **Open** button — Add Files dialog box appears
7. Check checkbox for each build target to which the file applies
8. Click **OK** button — Add Files dialog box closes; in Project window, filename of added file appears under `Sources` folder

Building (Compiling) a Project

To build a project:

1. Select the build target that you want to build (Figure 2.14)

Figure 2.14 Build Target Drop Down Box



2. From IDE main menu bar, select **Project**
3. Select **Make** — IDE builds (assembles, compiles, and links) project; Error & Warnings window opens showing error messages and warning messages if appropriate

Establishing Communication

The Freescale In-Circuit Emulator Base (FSICEBASE) allows you to connect to a host computer in two ways:

- through an Ethernet port
- through a USB port

Establishing Communication Through Ethernet Port

If you use an Ethernet connection to establish communication between your host computer and the FSICEBASE through a LAN, you need to do three things:

- have network administrator assign IP address on LAN to the FSICEBASE
- set IP address on FSICEBASE
- specify IP address in the debugger

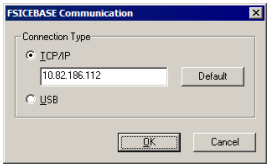
The following procedures explains how to do these things step by step.

To establish communication through a LAN:

1. Set up hardware as explained in “Setting Up The Hardware” on page 15.
2. Obtain the IP address that your network administrator assigned to the FSICEBASE
3. Start the debugger

   (See “Starting the Debugger” on page 30.)
4. Make sure target interface is FSICEBASE
   a. From debugger main menu, select **Component**
   b. Select **Set Target** — Set Target dialog box appears
   c. Select **FSICEBASE Target Interface** from Target Interface drop-down box
   d. Click **OK** — the debugger adds the FSICEBASE-HC08 menu to the main menu bar
5. From debugger main menu, select **FSICEBASE-HC08**
6. Select **Communication**— Communication dialog box appears (Figure 2.16)

Figure 2.15 Communication Dialog Box



7. Select **TCP/IP**
8. In the text box, type the IP Address that your network administrator assigned to the FSICEBASE

NOTE For more information on the IP address of the FSICEBASE, see "Assigning An IP Address To The FSICEBASE" on page 38. You may also connect using the default IP address directly to a computer using the Ethernet crossover cable, however you must set the PC IP address and mask.

9. Click **OK**

   The debugger connects to the FSICEBASE through the Ethernet port.

Establishing Communication Through USB Port

If you use a USB connection to establish communication between your host computer and the FSICEBASE:

1. Set up hardware as explained in "Setting Up The Hardware" on page 15.
2. Start the debugger

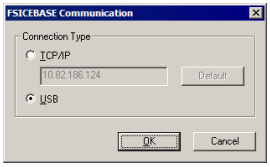
   (See "Starting the Debugger" on page 30.)

NOTE If you have started the debugger from your project previously, when you select Debug in the IDE, the Debugger attempts to connect to the FSICEBASE with the last known settings. If the Debugger connects, you do not need to perform the following steps.

3. Make sure target interface is FSICEBASE
   a. From debugger main menu, select **Component**
   b. Select **Set Target** — Set Target dialog box appears
   c. Select **FSICEBASE Target Interface** from Target Interface drop-down box
   d. Click **OK**
4. From debugger main menu, select **FSICEBASE-HC08**

5. Select **Communication**— Communication dialog box appears (Figure 2.16)

Figure 2.16 Communication Dialog Box



6. Select **USB**
7. Click **OK**

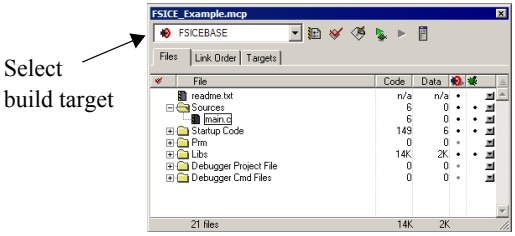
   The debugger connects to the FSICEBASE through the USB port.

Starting the Debugger

To start the debugger:

1. Select build target that you want to debug

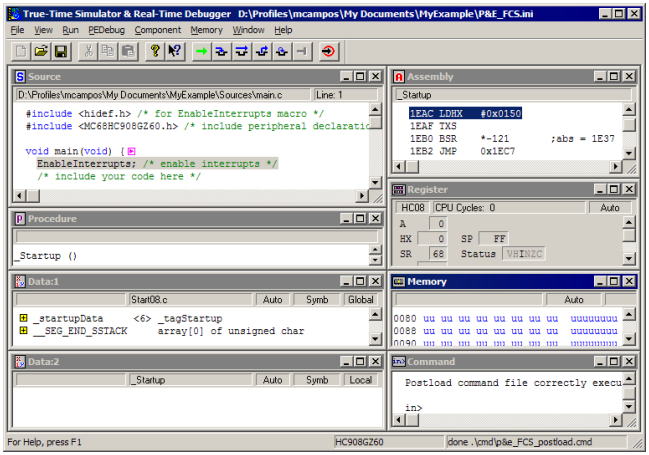
Figure 2.17 Build Target in Project Window



2. From main menu bar, select **Project**
3. Select **Debug**

   The IDE opens the **True-time Simulator & Real-time Debugger** window. The IDE loads the application in the debugger, downloads the code, and establishes communication with the FSICEBASE station.

Figure 2.18 True-time Simulator & Real-time Debugger Window

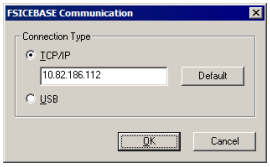


NOTE Source and Assembly panes display the `main.c` program and code.

4. Specify communication information if necessary.

   If the you have not previously specified communication information, the debugger opens the Communication dialog box (Figure 2.19). You must specify communication information to make a connection.

Figure 2.19 Communication Dialog Box



   a. If you are using an Ethernet connection to connect your host computer to the FSICEBASE station (through a LAN):

      - select **TCP/IP,** and
      - type the IP address of the FSICEBASE

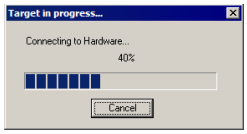
NOTE The network administrator of your Local Area Network (LAN) needs to assign the IP address of the FSICEBASE on the network. You can use the default IP address, and give this address to your network administrator. Or your network administrator might choose to create a different IP address.

b. If you are using a USB cable to connect your host computer directly to the FSICEBASE station, select **USB**,

c. Click **OK**

The debugger attempts to connect to the FSICEBASE. An information box (Figure 2.20) shows the progress.

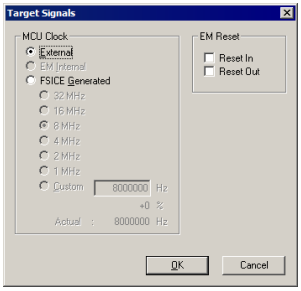
Figure 2.20 Connection to Target Progress



After the debugger makes initial contact with the FSICEBASE station, the Target Signals dialog box (Figure 2.21) appears allowing you to specify target specific information.

5. Specify target signal information

Figure 2.21 Target Signals Dialog Box



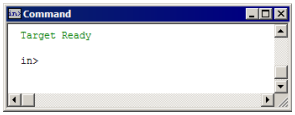
a. Specify MCU clock frequency

b. Specify emulator module reset type if applicable

c. Click **OK**

The debugger attempts to finish making a connection to the FSICEBASE station. After successfully connecting to the FSICEBASE station, the Command window (Figure 2.22) of the debugger shows the status message “Target Ready.”

Figure 2.22 Target Ready Message

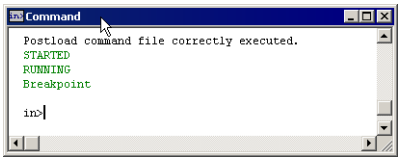


6. Set breakpoints if desired
   a. Right click mouse on executable line of source code in Source pane of debugger
   b. Select **Set breakpoint**
7. Run application
   a. From debugger main menu, select **Run** – Run menu appears
   b. Select **Start/Continue**

      Your application executes until encountering the first breakpoint; Command pane displays program status.

NOTE Alternatively, you can click on Start/Continue icon

Figure 2.23 Debugger Simulator Command Pane



8. Click Start/Continue icon — Simulator resumes program execution
9. Click Halt icon — Simulator stops program execution
10. In Debugger Simulator Window tool bar, select **File** > **Exit** to exit debugger

For more information about how to use the **True-time Simulator & Real-time Debugger**, see “Using the Debugger”.

3

Configuring The System

This chapter explains how to configure the Freescale In-Circuit Emulator Base (FSICEBASE). This chapter contains the following sections:

- Setting Up The System
- Setting Up Logic Cables And Connectors
- Modifying Values In Memory
- Using Log Files

Setting Up The System

In order to use the Freescale In-Circuit Emulator Base (FSICEBASE), you need to make sure that the system is configured properly. Configuring the FSICEBASE system includes:

- Specifying A Target
- Specifying Communication Information
- Assigning An IP Address To The FSICEBASE
- Specifying a Memory Map
- Specifying the Clock Speed
- Loading Program In Debugger

Specifying A Target

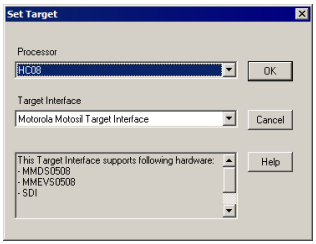
To specify the target, including the specific derivative:

1. Start the debugger — the **True-time Simulator & Real-time Debugger** window appears.

   For more information see “Starting the Debugger” on page 30.

2. From debugger main menu, select **Component**
3. Select **Set Target** — Set Target dialog box opens

Figure 3.1 Set Target Dialog Box



4. Select appropriate processor from **Processor** drop-down menu
5. Select appropriate **Target Interface**.

   To specify the FSICEBASE as the target interface, select **FSICEBASE Target Interface**.
6. Click **OK**

   The debugger configures itself to work with the target that you specified.

Notice that the main menu of the debugger reflects your selection. The menu item between the **Run** menu and the **Component** menu shows the name of the target that you selected. For example, if you selected as the target interface, the main menu contains a menu item labeled **FSCICEBASE-HC08**.

Specifying Communication Information

When you start the debugger from a the CodeWarrior IDE, the debugger automatically prompts you to specify communication information. However, if necessary, you can change the communication information directly from the debugger.

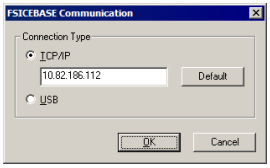
To specify communication information:

1. Start the debugger — the **True-time Simulator & Real-time Debugger** window appears.

   For more information see “Starting the Debugger” on page 30.
2. From debugger main menu, select **FSCICEBASE-HC08**

   The FSICEBASE-HC08 menu is between the Run menu and the Component menu. If you do not see the FSICEBASE-HC08 menu, you need to specify the target interface. For more information on specifying a target see, “Specifying A Target” on page 35.
3. Select **Communication** — Communication dialog box opens (Figure 3.2)

Figure 3.2 Communication Dialog Box



4. Specify communication information
   a. If you use an Ethernet connection to connect your host computer to the FSICEBASE:
      - select **TCP/IP,** and
      - in the text box, type the IP address of the FSICEBASE

NOTE The network administrator of your Local Area Network (LAN) needs to assign the IP address of the FSICEBASE on the network. You can use the default IP address, and give this address to your network administrator. Or, your network administrator might choose to create a different IP address. If the network administrator chooses the IP address, you need to assign the IP address to the FSICEBASE. For more information see, "Assigning An IP Address To The FSICEBASE" on page 38.

NOTE Make sure that the cable between the host computer and the FSICEBASE is a cross-over Ethernet cable if you are connecting directly to the PC (not through a LAN). Also make sure that the host computer uses a static IP address. (The FSICEBASE does not assign an IP address to the host computer.) If you use the default IP address of the FSICEBASE (192.168.0.1), we recommend that you assign the following IP address to the host computer: 192.168.0.2.

NOTE Make sure that the host computer and FSICEBASE both use the same subnet mask. For more information, see "Assigning An IP Address To The FSICEBASE" on page 38.

   b. If you use a USB cable to connect your host computer directly to the FSICEBASE station, select **USB**,
5. Click **OK**

   The debugger attempts to connect to the FSICEBASE. An information box shows the progress. You can click **Cancel** in the information box if you do not want to immediately connect to the FSICEBASE.

The debugger saves the communication information that you specified. It uses the communication information the next time that it connects to the FSICEBASE.

Assigning An IP Address To The FSICEBASE

The FSICEBASE ships from the factory with the following internal default IP address:

`192.168.0.1`

Depending on how you connect the host computer to the FSICEBASE, you might need to change the IP address of the FSICEBASE. The CodeWarrior software includes a utility that allows you to assign a different IP address to the FSICEBASE.

To assign an IP address to the FSICEBASE:

1. Use a USB cable to connect the host computer to the FSICEBASE

NOTE You may also connect using the setup utility to make network configuration changes via Ethernet using the default IP address directly using the Ethernet crossover cable or through your LAN using the straight through Ethernet cable if your LAN supports the default IP settings.

2. Start the FSICEBASE Configuration Utility
   a. From Windows desktop, click **Start** menu
   b. Select **Run**
   c. Browse to the following executable file:

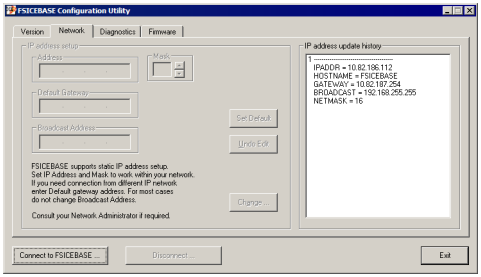
      `installation_directory\prog\GDI\FSICEBASE\setup.exe`

      The *`installation_directory`* is the directory where you installed the CodeWarrior software. The default installation directory is

      `C:\Program Files\Metrowerks\CW08 V3.1`

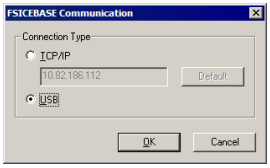
   d. Click **OK** — The **FSICEBASE Configuration Utility** starts (Figure 3.3)

Figure 3.3 FSICEBASE Configuration Utility



3. Click **Connect to FSICEBASE** button — **FSICEBASE Communication** dialog box appears

Figure 3.4 FSICEBASE Communication Dialog Box



4. Select **USB**

NOTE You can also use the default IP address to connect through TCP/IP.

5. Click **OK** — the FSICEBASE Configuration Utility connects to the FSICEBASE
6. Click **Network** tab of FSICEBASE Configuration Utility
7. In **Address** text box, type the IP address that you want to assign to the FSICEBASE

NOTE All hosts on a network must have a unique IP address. If you are connecting the FSICEBASE to a Local Area Network (LAN), consult with your network administrator to obtain a valid IP address.

8. From **Mask** combo box, select the subnet mask that you want to assign to the FSICEBASE

NOTE All the hosts in a sub-network must have the same subnet mask. For that reason, if you connect the host computer directly to the FSICEBASE (not through a LAN), you must ensure that the host computer uses the same subnet mask as the FSICEBASE.

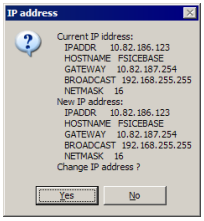
9. If applicable, in **Default Gateway** text box, type the IP address of the gateway that the FSICEBASE should use to connect to a network
10. If applicable, in **Broadcast Address** text box, type the IP address that you want the FSICEBASE to use as the broadcast address on the network

NOTE The gateway IP address is required to connect from one LAN to another. The broadcast IP address is the last IP address in the range of IP addresses on a network. The broadcast address is reserved by the network to allow a single host to make an announcement to all hosts on the network. Consult your network administrator for more information. Normally you should not need to update this value.

11. Click **Change** button

    The FSICEBASE Configuration Utility displays a dialog box (Figure 3.5) with the information that you specified. If information is wrong, click **No** to return to the Network tab and correct the information.

Figure 3.5 IP Address Change Confirmation Dialog Box



12. Click **Yes**

The FSICEBASE Configuration Utility assigns the new IP information to the FSICEBASE.

Specifying a Memory Map

Different MCU designs require different memory map configurations of the FSICEBASE system.

A personality file defines memory maps for particular MCUs. The personality file defines the memory map of each MCU supported by an emulator module (EM). Personality files ship with the separately purchased EMs. Refer to the appropriate EM user's manual to determine the personality files used by a particular EM module.

If an EM is connected to the FSICEBASE, the CodeWarrior software automatically loads the default personality file that corresponds to the EM. If the CodeWarrior software does not find an appropriate personality file, the debugger displays an error message when it tries to connect to the FSICEBASE.

After the debugger has loaded a memory map, you can view the memory map and modify it.

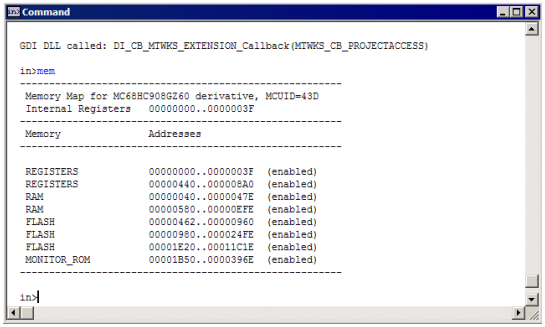
To use the Command line to view the current memory map:

1. From debugger main menu, select **Window** > **Command** to view the Command window. If you do not see the Command window:
   a. From the debugger main menu select **Component** > **Open**
   b. Select **Command**
   c. Click **OK** — the debugger opens a new Command window.
2. Click on command line (place insertion point on command line)

3. Type `MEM`

   Command window (Figure 3.6) displays memory map information: a representation of the current system memory map, and the lower and upper boundaries of the internal module that contains the MCU registers.

Figure 3.6 Command Window in the Debugger



To modify a memory map:

1. Start the debugger — the **True-time Simulator & Real-time Debugger** window appears.

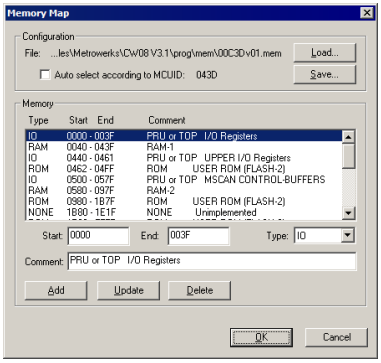
   For more information see “Starting the Debugger” on page 30.

2. From debugger main menu, select **FSCICEBASE-HC08**

   The FSICEBASE-HC08 menu is between the Run menu and the Component menu. If you do not see the FSICEBASE-HC08 menu, you need to specify the target interface. For more information on specifying a target see, “Specifying A Target” on page 35.

3. Select **Memory Map** — Memory Map dialog box opens (Figure 3.7)

Figure 3.7 Memory Map Dialog Box



4. Specify memory map information
   a. From **Memory** list box, select portion of map to change
   b. In **Start** text box, type new start address of range desired
   c. In **End** text box, type new end address of range desired
   d. Select **Type** of memory the new range should represent
   e. In **Comment** text box, type new description of range if appropriate
   f. Click **Update** button to update highlighted range, or **Add** button to add a new range (be careful not to overlap ranges)
5. To delete an existing range:
   a. From Memory list box, select portion of map to delete
   b. Click **Delete**
6. To Save the definitions of the memory map that you specified:
   a. Click **Save** — Save Memory Configuration dialog box appears
   b. In File Name text box, type name you want to give the memory map file (`.mem` file)
   c. Click **Save** — debugger saves `.mem` file, which you can use (load) later
7. Click **OK**

The debugger loads the new memory map information. The Command window of the debugger shows confirmation message.

Specifying the Clock Speed

The FSICEBASE platform board can supply an external oscillator clock source for the MCUs OSC1 input. Note that many emulator modules (EMs) require a specific jumper

configuration so that this clock source can be used. Refer to the specific EM user's manual for EM clock source information.

The FSICEBASE has several clock frequencies available: six internally generated clock frequencies (32 MHz, 16 MHz, 8 MHz, 4 MHz, 2 MHz, and 1 MHz), an external clock source, or an on-board EM clock (only on certain EM boards). You can also define a custom internal clock speed.

If you use an external clock source, you need to use a logic clip to connect the clock to the FSICEBASE. You must use logic clip A. Use the white wire to connect to the external clock.

To specify the clock speed:

1. Start the debugger — the **True-time Simulator & Real-time Debugger** window appears.

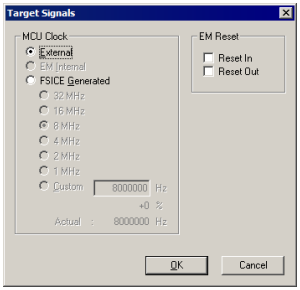
   For more information see “Starting the Debugger” on page 30.

2. From debugger main menu, select **FSCICEBASE-HC08**

   The FSICEBASE-HC08 menu is between the Run menu and the Component menu. If you do not see the FSICEBASE-HC08 menu, you need to specify the target interface. For more information on specifying a target see, “Specifying A Target” on page 35.

3. Select **Target Signals** — Target Signals dialog box opens (Figure 3.8)

Figure 3.8 Target Signals Dialog Box



4. Specify clock source. From **MCU Clock** section of dialog box, select whether the clock is connected externally, on a connected emulator module (EM), or FSICEBASE Generated.

5. Specify clock speed if internally generated

   a. If you selected **FSICE Generated**, select the clock speed to be emulated

   b. If you selected **Custom**, type clock speed in **Custom** text box

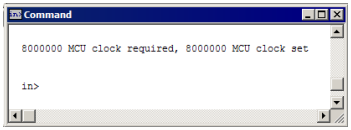
NOTE If you specify a custom clock speed, be aware that the FSICEBASE can provide clock speeds from 4100Hz to 40MHz in steps of 5kHz. The

FSICEBASE uses a clock synthesis chip to generate the clock speed. This method is not as accurate as a crystal: only within about 0.75% absolute frequency with about 5% jitter. If you choose a clock value (either from one of the radio buttons or by typing it in) that is an integer divisor of 32MHz or 9.8304MHz, you will get an accurate crystal-sourced clock.

NOTE If you set the MCU clock speed above or below the limits described in the MCU or EM board specifications for the operating voltage and temperature, you may not be able to communicate with the FSICEBASE. If this occurs, change the clock settings in the target signals dialog box so they are withing the operating limits of the MCU or EM board before attempting to connect again.

6. Click **OK**

Figure 3.9 Confirmation Message in Command Window



The debugger instructs the FSICEBASE to use new clock information. The Command window (Figure 3.9) of the debugger shows confirmation message.

Loading Program In Debugger

If you start the debugger from a project in the CodeWarrior IDE, the debugger automatically loads the program into the target memory. The debugger can load a file that is in an S-Record format (`.s19` file) or an absolute format (`.abs` file). The default project configuration produces an output file in the `.abs` format.

If want to use the debugger to load an application:

1. From the debugger main menu, select **File**
2. Select **Load Application** — the **Open File** dialog box appears
3. Specify the location of the executable program file and the type (`.abs` or `.s19` file)
4. Click **Open** — debugger loads the application you specified

Setting Up Logic Cables And Connectors

The diagram below shows the pin numbering for pod A, pod B, and pod C logic cable connectors of the station module. Table 3.1 shows the pinout information of the logic

clips. You can use the logic clips to capture external signals and for triggering. (Pin 19 of the pods provides connection to an external ground.) In addition, pod connectors A and B are used as external clock inputs for the emulator clock and bus state analyzer time tag. The table also provides color code information for each pod. The external clock inputs are through pin 17 of pods A and B. Pod A pin 17 is the external clock input for the emulator. To use this source as the external MCU clock, make the desired clock connection to the white probe tip and select the external clock source in the target signals dialog or use the OSC command to select an external source.

Pod B pin 17 is the external time tag input for the bus state analyzer. To use this source for the analyzer, make the desired clock connection to the white probe tip and select the external time tag clock source in the BSA configuration dialog or use the TIMETAG command to select an external time tag source.

Table 3.1 Pod and Logic Cable Pin Assignments

Pod Pin	Pod A Signal	Pod B Signal	Pod C Signal	Probe Color
1	LC0	LC8	LC16	Brown (BRN)
2	GND	GND	GND	
3	LC1	LC9	LC17	Red (RED)
4	GND	GND	GND	
5	LC2	LC10	LC18	Orange (ORG)
6	GND	GND	GND	
7	LC3	LC11	LC19	Yellow (YEL
8	GND	GND	GND	
9	LC4	LC12	LC20	Green (GRN)
10	GND	GND	GND	
11	LC5	LC13	LC21	Blue (BLU)
12	GND	GND	GND	
13	LC6	LC14	LC22	Purple (PUR)
14	GND	GND	GND	
15	LC7	LC15	LC23	Gray (GRY)
16	GND	GND	GND	

Table 3.1 Pod and Logic Cable Pin Assignments

Pod Pin	Pod A Signal	Pod B Signal	Pod C Signal	Probe Color
17	EXT_OSC	TT_OSC	None	White
18	GND	GND	GND	
19	GND	GND	GND	Black
20	GND	GND	GND	

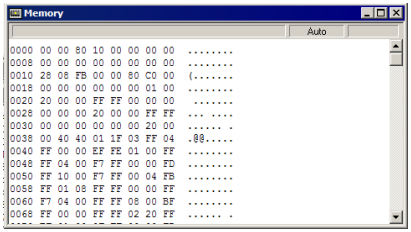
Modifying Values In Memory

During a debugging session, you often will want to manipulate the values contained in specific memory locations. You can use the Memory window to change the contents of memory.

To change a value in memory:

1. From debugger main menu, select **Window** > **Memory** to view the Memory window (Figure 3.10). If you do not see the Memory window:
   a. From the debugger main menu select **Component** > **Open**
   b. Select **Memory**
   c. Click **OK** — the debugger opens a new Memory window.

Figure 3.10 Memory Window



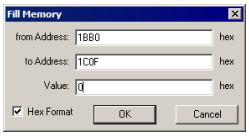
2. Use scroll bar of Memory window to navigate to desired memory addresses.

   The Memory window shows memory addresses in the far left column. The second column shows the value contained in that address. The next column shows the value contained in the next higher address, and so on. Each row shows the values of sixteen addresses. The far right area of the window shows the corresponding ASCII representation of the values. The layout of the values changes if you select a different numeric format.

3. Double click on the value to change — the Memory window highlights the value and displays “Edit mode” at the top of the window
4. Type the desired value
5. Press ENTER key

   The debugger writes the new value to memory.
6. If you want to block fill a range of memory addresses:
   a. In Memory window, right click mouse — the Memory window displays a drop-down menu
   b. Select **Fill** — Fill Memory dialog box appears

Figure 3.11 Fill Memory Dialog Box



   c. in **from Address** text box, type beginning address of range to fill
   d. in **to Address** text box, type ending address of range to fill
   e. in **Value** text box, type value to write to each address in the range
   f. Click **OK**

The debugger allows you to view and modify memory values in several other ways. For more complete information, refer to the **Help** menu of the debugger.

Using Log Files

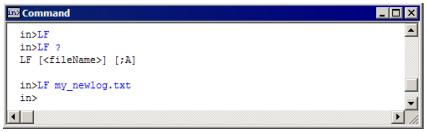
The FSICEBASE can maintain a log file that will capture events displayed on the debug screen. Entries in the log include:

- Commands entered on the command line
- Commands read from a script file
- Responses to commands
- Error messages
- Notifications of changes in status, such as breakpoints and WAIT or STOP instructions.

To use the Command line to start a log file:

1. Select **Window** > **Command** to view the Command window (Figure 3.12). If you do not see the Command window:
   a. From the debugger main menu select **Component** > **Open**
   b. Select **Command**
   c. Click **OK** — the debugger opens a new Command window.

Figure 3.12 Command Window in the Debugger



2. Click on command line (place insertion point on command line)
3. Type `LF` — **Open file** dialog box appears
4. Specify log file
   a. In **File name** text box, type name you want to give log file
   b. Click **Open** — debugger starts logging information from Command window to log file
5. To stop writing to the log file:
   a. Click on command line (place insertion point on command line)
   b. Type `LF` — **Open file** dialog box appears
   c. Click **Cancel** — debugger stops writing to log file; Command window shows confirmation message

4

Using The MON08 Debug Port

The Freescale In-Circuit Emulator Base (FSICEBASE) includes a MON08 debug port. The debug port is located on the side of the enclosure of the FSICEBASE. You communicate to the MON08 debug port using the Universal Serial Bus (USB) port.

This chapter contains the following sections:

- Setting Up Hardware For the MON08 debug port
- Establishing Communication Through The MON08 Debug Port

The MON08 debug port allows you to communicate with a target MCU device for the purpose of in-circuit debugging and programming. The MON08 debug port let's you take advantage of the monitor mode of the MCU. You can directly control:

- program execution
- reading/writing of registers
- reading/writing of values in memory
- debugging of code on the processor

The CodeWarrior software includes support for the MON08 debug port. Although other software is available to use with this port, you do not need to install additional software. In order to communicate with a target through the MON08 debug port, you need to:

- set up the hardware and install CodeWarrior Development Studio for HC(S)08 software
- establish communication

Setting Up Hardware For the MON08 debug port

The USB port of the FSICEBASE provides access from the host computer to the MON08 debug port.

To setup the hardware to connect through the MON08 debug port:

1. Make sure power supply is not connected to FSICEBASE
2. Connect rectangle-shaped end of USB cable to host computer
3. Connect other end (square-shaped end) of USB cable to FSICEBASE
4. Connect 16-pin ribbon cable to the MON08 interface connector of the FSICEBASE

CAUTION	Make sure to connect ribbon cable so that striped side of cable lines up with pin 1 of interface connector.

5. Connect other end of 16-pin ribbon cable to MON08 port of target board

CAUTION	Make sure to connect ribbon cable so that striped side of cable lines up with pin 1 of MON08 port of board or that all the board connections are appropriate for the specific MCU you are using.

6. Connect power cable to FSICEBASE
   a. Connect round end of 5-volt power cord to barrel connector on FSICEBASE
   b. Plug power supply into surge-protected strip
   c. Connect surge-protected strip to AC outlet
7. Switch on power switch of FSICEBASE

The FSICEBASE is now ready to provide communication between a target device and a host computer through the MON08 debug port.

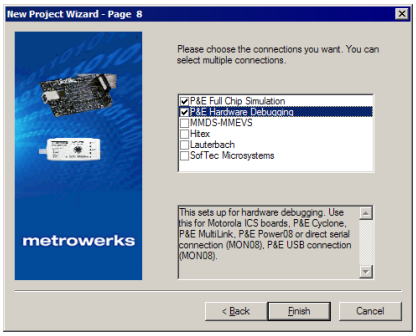
Establishing Communication Through The MON08 Debug Port

To establish communication between your host computer and the FSICEBASE, you will need to install the CodeWarrior software, create a project, and start the debugger. The following procedure assumes that have installed the CodeWarrior software and created a project.

When you use the New Project Wizard to create a project in the IDE, the last page of the wizard (Figure 4.1) allows you to specify the connection information that you want the project to contain. If you check the **P&E Hardware Debugging** checkbox, the wizard adds a build target to your project that is configured to connect through P&E Hardware. This includes support for connecting through a MON08 debug port.

If you did not include the **P&E Hardware Debugging** connection when you created your project, your project does not automatically contain a build target for that connection.

Figure 4.1 Last Page Of Connection Wizard



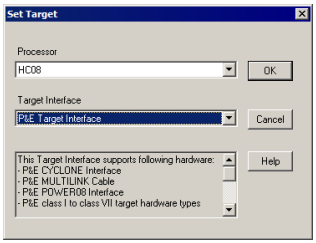
However, you can specify the connection information (and MON08 support) in the debugger. The following procedure explains how to do so.

To establish communication between your host computer and the FSICEBASE through the MON08 debug port:

1. Set up hardware as explained in “Setting Up Hardware For the MON08 debug port” on page 49.
2. Start the debugger

   (See “Starting the Debugger” on page 30.)
3. From debugger main menu, select **Component**
4. Select **Set Target** — Set Target dialog box appears (Figure 4.2)

Figure 4.2 Set Target Dialog Box



5. From the **Processor** drop-down menu, select the appropriate target processor type
6. From **Target Interface** drop-down menu, select **P&E Target Interface**

7. Click **OK**

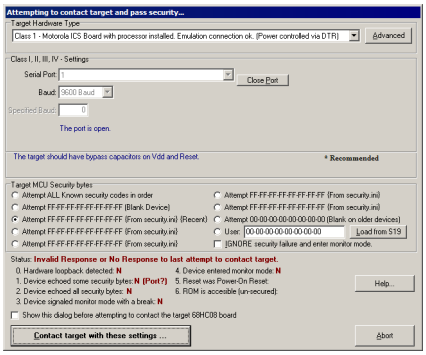
   The debugger begins to run in Full Chip Simulations mode, which is the default mode for this target interface in the debugger.

   The debugger adds a menu item labeled **PEDebug** to the main menu of the debugger. You can use the **PEDebug** menu to configure target specific information, and to instruct the debugger to connect or disconnect from the target device.

8. Select Derivative
   a. From debugger main menu, select **PEDebug**
   b. Select **Device:*xxxx***
   c. Select the derivative that corresponds to the target device to which the FSICEBASE is connected

9. Specify In-Circuit Debug mode
   a. From debugger main menu, select **PEDebug**
   b. Select **Mode:Full-Chip Simulation** — mode menu appears
   c. Select **In-Circuit Debug/Programming** — Connection dialog box appears (Figure 4.3); dialog box shows MON08 connector information for specific MCUs

Figure 4.3 PEDebug Connection Dialog Box



10. Specify connection information
    a. From Target Hardware Type drop-down box, select **Class 7 - P&E MON08 MULTILINK or USB MON08 MULTILINK connected to target via ribbon cable**
    b. Make sure the MCU you are using is selected in the device type pull-down list
    c. Specify other connection information as appropriate

d. Make sure the P&E USB connection shows up in the port pull-down list (you must have the FSICEBASE powered and switched on for it to appear in the list)

11. Click **Contact target with these settings**

    The debugger connects to the target device through the MON08 debug port of the FSICEBASE. You can view the status of the target power using the target power LED. The ready MON08 LED indicates that the USB connection is operational and the MON08 debug port is ready to communicate.

5

Using the Debugger

This chapter explains how to use some of the debugging features of the Freescale In-Circuit Emulator Base (FSICEBASE). This chapter contains the following sections:

- Starting The Debugger
- Changing CPU Register Values
- Viewing Memory
- Resetting The Emulation System
- Using Breakpoints And Watchpoints
- Controlling Program Execution

For more complete information on the features of the debugger, refer to:

- the online help of the debugger, available from the **Help** menu of the debugger
- *CodeWarrior™ Debugger* User Manual, available in PDF format from the directory where you installed the CodeWarrior software:

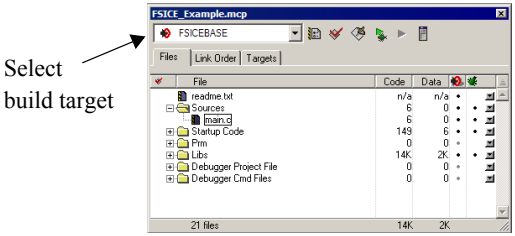
  `/Help/pdf/Manual_Engine_HC08.pdf`

Starting The Debugger

To start the debugger:

1. In the Project window of the CodeWarrior IDE, select the build target that you want to debug.

Figure 5.1 Build Target in Project Window



2. From main menu bar of the IDE, select **Project**

3. Select **Debug**

   The IDE opens the **True-time Simulator & Real-time Debugger** window. The IDE loads the application in the debugger and establishes communication with the FSICEBASE module.

Changing CPU Register Values

The Register window of the debugger displays the contents of the CPU registers and the condition code register. These registers – A, HX, SP, SR, Status, and PC – contain data that affects execution of an instruction.

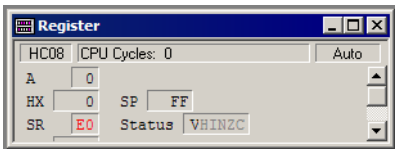
You can use the Register window or the Command line to change the value contained in registers.

Using the Register Window to Change Register Values

To use the Register window to change the value contained in a register:

1. From debugger main menu, select **Window** > **Register** to see the Register window (Figure 5.2). If you do not see the Register window:
   a. From the debugger main menu select **Component** > **Open**
   b. Select **Register**
   c. Click **OK** — the debugger opens a new Register window.

Figure 5.2 Register Window in the Debugger



2. In the **Register** Window, double-click the value that you want to change — the value becomes highlighted
3. Type the desired value
4. Press ENTER key

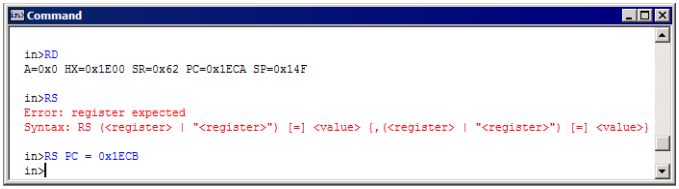
The debugger sets the value that you specify, and uses the value the next time that it executes an instruction.

Using the Command Line to Change Register Values

To use the Command line to change the value contained in a register:

1. Select **Window** > **Command** to view the Command window (Figure 5.3). If you do not see the Command window:
   a. From the debugger main menu select **Component** > **Open**
   b. Select **Command**
   c. Click **OK** — the debugger opens a new Command window.

Figure 5.3 Command Window in the Debugger



2. Click on command line (place insertion point on command line)
3. Type `RS` — the Command window displays the syntax of the `RS` command
4. Use the `RS` command to set register values — for example. to set the value of the program counter (PC) to hexadecimal `1ECB`, type:

   ```
   RS PC = 0x1ECB
   ```

   You can also use the `RD` command in the Command window. The `RD` command displays the current values contained in the registers. For more information on the `RD` and `RS` commands, see the documentation that accompanies the CodeWarrior software.

Viewing Memory

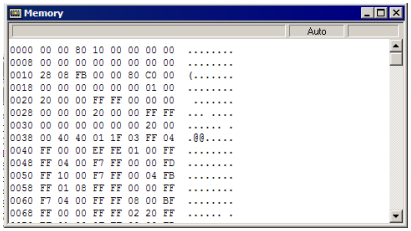
The Memory window displays the values stored in the target memory. It displays RAM and ROM values updating in real-time during code execution if you select the periodic dispay mode. You can view the values in different numeric formats, such as decimal, hexadecimal, and binary.

To view values stored in target memory:

1. From debugger main menu, select **Window** > **Memory** to view the Memory window (Figure 5.4). If you do not see the Memory window:
   a. From the debugger main menu select **Component** > **Open**
   b. Select **Memory**

c. Click **OK** — the debugger opens a new Memory window.

Figure 5.4 Memory Window



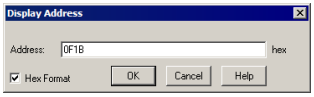
The Memory window shows memory addresses in the far left column. The second column shows the value contained in that address. The next column shows the value contained in the next higher address, and so on. Each row shows the values of sixteen addresses. The layout changes if you instruct the Memory window to display a different numeric format.

The far right area of the window shows the corresponding ASCII representation of the values. Control and other non-printing characters appear as periods.

2. To view values in a different numeric format:
   a. Right click mouse in Memory window — drop-down menu appears
   b. Select **Format** — drop-down menu appears
   c. Select a format such as Hex, Dec, or Bin

      Memory window displays values in format that you selected.
3. To change the display update mode
   a. Right click mouse in Memory window — drop-down menu appears
   b. Select **Mode** — drop-down menu appears
   c. Select a mode: **Automatic** - updates when the debugger issues a command, such as after stepping, **Periodical** - updates at the specified time interval, **Frozen** - never updates
4. Use scroll bar of Memory window to navigate to desired memory addresses.
5. To display a specific address:
   a. Right click mouse in Memory window — drop-down menu appears
   b. Select **Address** — Display Address dialog box (Figure 5.5) appears

Figure 5.5 Display Address dialog box



c. In **Address** text box, type address that you want Memory window to display

d. Click **OK**

The Memory window displays and highlights the address that you specified. The memory window also highlights the contents of the address.

To learn about viewing the memory map, see “Specifying a Memory Map” on page 40.

Resetting The Emulation System

The debugger allows you to reset the emulation MCU and set the PC register to the contents of the reset vector.

To reset the FSICEBASE:

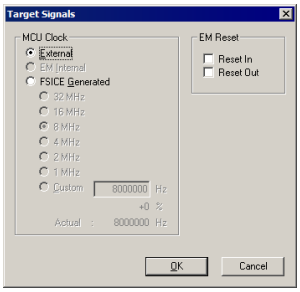
1. If the FSICEBASE is connected to an emulator module (EM), specify the type of reset available to the EM.

   a. From debugger main menu, select **FSCICEBASE-HC08**

   The FSICEBASE-HC08 menu is between the Run menu and the Component menu. If you do not see the FSICEBASE-HC08 menu, you need to specify the target interface. For more information on specifying a target see, “Specifying A Target” on page 35.

   b. Select **Target Signals** — Target Signals dialog box opens (Figure 5.6)

Figure 5.6 Target Signals Dialog Box



c. If you want to allow a reset signal coming from the target system (through the target cable), check the **Reset IN** checkbox.

   Some EMs include a hardware jumper that governs target resets. Make sure to configure jumpers as necessary to use the **Reset IN** option. For more information, refer to your EM's documentation. Many MCUs also control reset availability in certain control registers specific to the MCU.

d. If you want to allow a reset signal to be sent to the target system (through the target cable) such as from an internal MCU source or from a debugger reset, check the **Reset Out** checkbox.

NOTE If you check both **Reset IN** and **Reset Out** the internal resets of the emulator MCU will not be sent to the target system. Debugger resets will still be sent to the target.

e. Click **OK**

2. From debugger main menu, select **FSCICEBASE-HC08**
3. Select **Reset**

   The debugger sends a reset signal to the FSICEBASE if the **Reset OUT** option was checked.

Using Breakpoints And Watchpoints

A properly defined breakpoint allows you control program execution so that you can analyze the contents of registers and memory locations, and the states of various signals. You can use the debugger to set breakpoints on lines of code. You can also set breakpoints on a specific address if the address contains an executable instruction.

You can define simple breakpoints or complex breakpoints.

A simple breakpoint causes the debugger to pause execution. During program execution, when the debugger encounters an instruction that is associated with a simple breakpoint, the debugger pauses execution. It pauses the execution before executing the instruction where the breakpoint is defined.

A complex breakpoint allows you to define conditions or commands. During program execution, when the debugger encounters an instruction that is associated with a complex breakpoint, the debugger pauses execution only if a certain condition is met. You can also define the breakpoint to execute a command instead of halting execution.

For more information on breakpoints, refer to the online help of the debugger.

Watchpoints are similarly used to control program execution, but they are set on data address locations. The conditions and command execution setup are similar to breakpoints. In the memory window, you can select an address location, right-click and

choose **Set Watchpoint** from the menu that appears. Conditions are set in the same way as breakpoints, but under the watchpoints tab in the controlpoints configuration dialog.

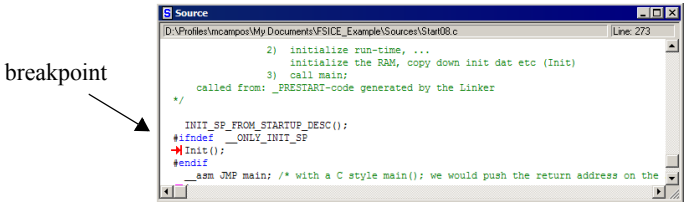
Setting Breakpoints on Lines of Code

To set a simple breakpoint on an executable line of code:

1. Load the program to debug
   a. If you launched the debugger from a project in the CodeWarrior IDE, the debugger automatically loads the program (`.abs` or `.elf` file)
   b. If the debugger has not loaded the program that you want to debug:
      - From the debugger main menu, select **File** > **Load Application**
      - Specify the location of the executable program file (`.abs` or `.s19` file)
      - Click **Open** — debugger loads the application you specified
2. In the **Source** window or in the **Assembly** window, place mouse cursor next to the executable line of code where you want to set the breakpoint
3. Right-click mouse — menu appears
4. Select **Set Breakpoint**

   The debugger sets a breakpoint on the line of code that you specified. A red arrow appears (Figure 5.7) to the left of the line of code. The debugger halts execution the next time that it encounters this breakpoint during execution.

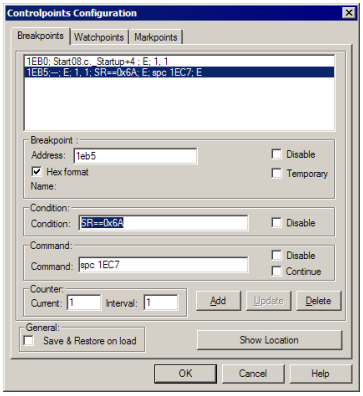
Figure 5.7 Breakpoint in Source Window



To define a complex breakpoint:

1. From debugger main menu, select **Run**
2. Select **Control Points** — **Controlpoints Configuration** dialog opens (Figure 5.8)

Figure 5.8 Controlpoints Configuration Dialog Box



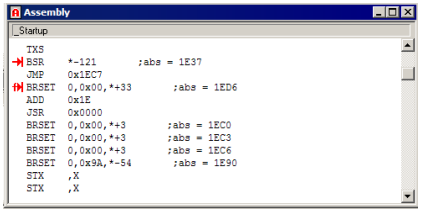
3. Click on **Breakpoints** tab
4. From breakpoint list, select breakpoint you want to change
5. If desired, in the **Condition** text box, type an expression

   When the debugger encounters the breakpoint, it evaluates the expression; if the expression evaluates to TRUE, the debugger acknowledges the breakpoint. It the expression evaluates to FALSE, the debugger ignores the breakpoint.
6. If desired, in the **Command** text box, type an executable instruction

   When the debugger encounters the breakpoint, it executes the instruction. If you specify a condition, the condition must be met for the instruction to be executed.
7. Click **Add** button — debugger adds breakpoint to list of breakpoints
8. Click **OK**

   The debugger sets all breakpoints listed in the **Controlpoints Configuration** dialog box.

Figure 5.9 Complex Breakpoint in Assembly Window



The Source window and Assembly window display arrows (Figure 5.9) to indicate location of breakpoints in code.

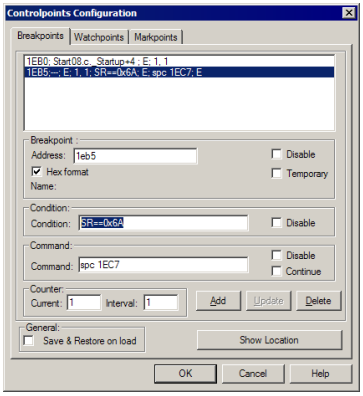
Setting Breakpoints on Addresses

You can set a breakpoint on a memory address. The address must contain an opcode (instruction fetch code). If an instruction exists at one address and the data used by the instruction exists on the next two bytes, you must set the breakpoint on the instruction address. If you set the breakpoint on an address containing data, the debugger ignores the breakpoint.

To set a breakpoint on an address:

1. Load the program to debug
   a. If you launched the debugger from a project in the CodeWarrior IDE, the debugger automatically loads the program (`.abs` or `.elf` file)
   b. If the debugger does not contain the application that you want to debug:
      - From the debugger main menu, select **File** > **Load Application**
      - Specify the location of the executable file (`.abs` or `.elf` file)
      - Click **Open** — debugger loads the application you specified
2. From debugger main menu, select **Run**
3. Select **Control Points** — **Controlpoints Configuration** dialog opens (Figure 5.6)

Figure 5.10 Control Points Dialog Box



4. Click on **Breakpoints** tab
5. In **Address** text box, type address that contains the opcode where you want to set the breakpoint

6. If desired, in the **Condition** text box, type an expression

   When the debugger encounters the breakpoint during execution, it evaluates the expression; if the expression evaluates to TRUE, the debugger uses the breakpoint. If the expression evaluates to FALSE, the debugger ignores the breakpoint.

7. If desired, in the **Command** text box, type an executable instruction

   When the debugger encounters the breakpoint, it executes the instruction. If you specified a condition, the condition must be met for the instruction to be executed.

8. Click **Add** button — **Controlpoints Configuration** dialog box adds breakpoint to list of breakpoints

9. Click **OK**

   The debugger sets all breakpoints listed in the **Controlpoints Configuration** dialog box.

Controlling Program Execution

One of the main functions of the debugger is to provide a way to control execution of the program running on the target.

To control program execution:

1. Load the program to debug
   a. If you launched the debugger from a project in the CodeWarrior IDE, the debugger automatically loads the program (`.abs` or `.elf` file)
   b. If the debugger does not contain the application that you want to debug:
      - From the debugger main menu, select **File** > **Load Application**
      - Specify the location of the executable file (`.abs` or `.elf` file)
      - Click **Open** — debugger loads the application you specified
2. From debugger main menu, select **Run**
3. Select the action that you want the debugger to perform.

   Table 5.1 describes the commands that control execution.

Table 5.1 Program Execution Commands

Command	Description
Start/Continue	Starts or continues execution of the loaded application from the current program counter (PC). Execution continues until the debugger encounters a breakpoint, a runtime error occurs, or the execution is halted.
Restart	Starts execution of the loaded application from the application entry point.
Halt	Interrupts and halts a running application. You can examine the state of each variable in the application, set breakpoints, watchpoints, and inspect source code while the application is halted.
Single Step	If the application is halted, performs a single step at the source level. Execution continues until the debugger reaches the next source reference. If the current statement is a procedure call, the debugger "steps into" that procedure. The Single Step command does not treat a function call as one statement, which is why it steps into the function.
Step Over	Performs a single step at the source level. but does not step into called functions. The debugger executes function call as if it were one statement.

Table 5.1 Program Execution Commands

Command	Description
Step Out	If the application is halted while executing a function, this command continues execution of the function. Execution stops at the instruction following the current function. If the debugger is not in the process of executing a function calls, the Step Out command is not performed.
Assembly Step	If the application is halted, performs a single step at the assembly level. Execution continues for one CPU instruction from the point it was halted. This command is similar to the Single Step command, but executes one machine instruction rather than a high level language statement.
Assembly Step Over	If the application is halted, performs a single step at the assembly level, stepping over subroutine call instructions.
Assembly Step Out	If the application is halted inside a function, this command continues execution and stops on the CPU instruction following the current function invocation. This command is similar to the Step Out command, but stops before the assignment of the result from the function call.

For more information on controlling program execution refer to the online help of the debugger, available from the **Help** menu of the debugger.

6

Using the Bus State Analyzer

This chapter explains how to use the bus state analyzer (BSA). This chapter contains the following sections:

- Overview
- Using the Bus State Analyzer

Overview

The bus state analyzer (BSA) shows the logical state of the target MCU bus and external signals during program execution. The BSA takes a snapshot of the MCU bus. It also captures the signals from the external logic clips Pods A, B, and C of the FSICBASE (24 lines in total). This capturing of data enables you to determine what is occurring in a system without actually disturbing the system.

At the end of each MCU clock cycle, the BSA takes a snapshot of the logical states of the target MCU bus and external inputs. The analyzer stores the snapshots in the trace buffer, according to its mode. (This action is known as storing cycles.)

NOTE The analyzer is a bus state analyzer: It does not show signal hold or setup times.

To start using the BSA, you need to define patterns of logical states as events (or terms). You also need to specify the analyzer mode: continuous, counted, or any of five sequential modes. This determines which cycles the analyzer stores.

Data collection (cycle storage) begins when you arm the analyzer and start program execution, depending on the triggering mode and state. Data collection continues until execution stops, through a specified number of events, or through a defined sequence of events, or optionally until the trace buffer is full.

Using the Bus State Analyzer

To use the bus state analyzer (BSA) to produce useful data that you can view and analyze, you must:

- define events (terms) and/or triggering sequence, and
- arm the BSA.

Defining Events

You define an event by specifying a combination of criteria. You can define the criteria to be particular values on the address or data bus, read or write access on an instruction or on data, extended address access, or signals sent through one of the five logic clips that you can connect to Pod A of the FSICEBASE.

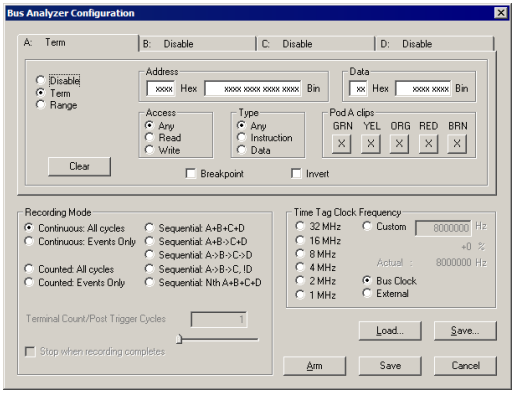
The Bus State Analyzer uses the criteria that you specify to create an event, and labels the event A, B, C, or D. When the BSA determines that the criteria of a certain event has been met, depending on the triggering mode, it records the data that is on the bus of the MCU at that particular clock cycle. It also records the data that is in the lines of Pods A, B, and C. You can control the way that the BSA records this information by specifying a recording mode.

To define an event:

1. Start the debugger
2. Load the program to debug
   a. If you launched the debugger from a project in the CodeWarrior IDE, the debugger automatically loads the program (`.abs` or `.elf` file)
   b. If the debugger has not loaded the program that you want to debug:
      - From the debugger main menu, select **File** > **Load Application**
      - Specify the location of the executable program file (`.abs` or `.s19` file)
      - Click **Open** — debugger loads the application you specified
3. From debugger main menu, select **FSCICEBASE-HC08**

   The FSICEBASE-HC08 menu is between the Run menu and the Component menu. If you do not see the FSICEBASE-HC08 menu, you need to specify the target interface. For more information on specifying a target see, “Specifying A Target” on page 35.
4. Select **Bus Analyzer Configuration** — Bus Analyzer Configuration dialog box opens (Figure 6.1)

Figure 6.1 Bus Analyzer Configuration Dialog Box



5. Select **Term** or **Range**

   A range consists of two 32-bit values. Range does not refer to a range of addresses. If you define an event as a range, the BSA triggers every time the input falls between the range starting term (the first 32-bit value) and the range ending term (the second 32-bit value).

6. In **Address** area, specify the address(es) that the BSA monitors
7. In **Data** area, specify the data that the BSA monitors
8. In **Access** area, specify the type of access that the BSA should monitor
9. In Type area, specify whether the BSA should record only if encountering data, instructions or any kind of value at the specified address
10. Specify Pod A signals (logic clips attach to pins of Pod A) that the BSA should monitor for this event

NOTE You can use five of the pod A logic clips to define an event. The other signals of Pod A, and the signals of Pods B and C cannot be used to define an event. The Bus Analyzer Configuration dialog box shows the five clips that you can use to trigger an event. The choice of these five signals is hard-set in the FSICEBASE; you cannot choose other signals to be used as event criteria, however, the BSA does capture data from all 24 lines of Pods A, B, and C.

11. If you want the term to also acts as a breakpoint, check the **Breakpoint** checkbox
12. Specify the Recording Mode
    - For information about the recording mode, see “Recording Modes” on page 70.
    - If you check the **Stop when recording completes** checkbox, the debugger stops program execution when bus state analyzer recording is done.

13. Specify the Time Tag Clock Frequency
14. If an appropriate recording mode is selected, check the box to stop execution when the trace buffer is completely filled with captured information (otherwise it will continue to capture new information, overwriting older information, until execution is halted)
15. Click **Save** to apply the event information to the current debug session and close the dialog box. The BSA uses the terms when you arm the BSA.
16. Click **Save...** to save the event information to a file
17. Click **Arm** to ready the BSA to collect data.

    The BSA does not start collecting data until execution begins.The debugger indicates that the BSA is armed by showing the word Armed in the status bar.

Recording Modes

When you define an event, you can specify the recording mode that the Bus State Analyzer uses to collect data. This section explains how the different modes work.

Continuous: all cycles

After execution begins, the trace buffer begins storing data from the first cycle. This continues until execution arrives at a breakpoint, or until you halt execution.

Continuous: events only

After execution begins, the trace buffer begins storing data when data matches an event definition. This continues until execution arrives at a breakpoint, or until you halt execution

Counted: all cycles

After execution begins, the trace buffer begins storing data until after the specified number of cycles from first cycle. (A breakpoint can stop storage before the analyzer stores the specified number of cycles, as can halting execution.)

Counted: events only

After execution begins, the trace buffer begins storing data that match an event definition for the specified number until it has captured the specified number of events. (A breakpoint can stop storage before the analyzer stores the specified number of cycles, as can halting execution.)

Sequential: A+B+C+D

After execution begins, the trace buffer begins storing data from the first cycle run. This continues through the occurrence of event A, B, C, or D (whichever is enabled); data storage ends after the specified number of post-trigger cycles.

Sequential: A+B -> C+D

After execution begins, the trace buffer begins storing data from the first cycle. This continues through the occurrence of two events: A or B, followed by C or D. Data storage ends after the specified number of post-trigger cycles.

If you select this mode, you must enable event A, event B, or both. You must enable event C, event D, or both. Otherwise, the bus state analyzer cannot be triggered.

Sequential: A -> B -> C !D

After execution begins, the trace buffer begins storing data from all cycles. This continues through the occurrence of three events, A, B, and C, in order, if event D does not occur. (If D occurs, the sequencer starts again looking for event A.) Data storage ends after the specified number of post-triggert cycles.

If you select this mode, you must enable events A, B, and C. Otherwise, the bus state analyzer never can be triggered. If you disable event D, you convert this mode to a simple, three-event sequence.

Sequential: A -> B -> C -> D

After execution begins, the trace buffer begins storing data from all cycles. This continues through the occurrence of four events, A, B, C, and D, in order. Data storage ends after the specified number of post trigger cycles.

If you select this mode, you must enable all four events A, B, C, then D. Otherwise, the bus state analyzer cannot be triggered.

Sequential: Nth event: A+B+C+D

After execution begins, the trace buffer stores data until N occurrences of cycles that match the definitions of events A, B, C, or D (whichever are enabled). Then the bus state analyzer captures the next 1.33M/2 cycles. This puts the Nth captured occurrence of A, B, C, or D in the middle of the trace buffer. The first half of the trace buffer will then contain information before the Nth event and the other half will contain information after the Nth event.

Note that the terminal count or post trigger cycles are valid only for counted or sequential modes. For a counted mode, this field specifies the number of cycles to be stored. For a

sequential mode, this field specifies the number of cycles to be stored after the trigger sequence occurs.

Time Tag Clock Frequency

An optional part of analyzer setup is specifying the frequency and source of the time tag clock. This clock provides a time reference value in each frame of the trace buffer. To select the clock frequency, see “Defining Events” on page 68.

You can select from the following frequencies:

- 32 Mhz Selects the 32 MHz oscillator.
- 16 Mhz Selects the 16 MHz oscillator.
- 8 Mhz Selects the 8 MHz oscillator.
- 4 Mhz Selects the 4 MHz oscillator.
- 2 Mhz Selects the 2 MHz oscillator.
- 1 Mhz Selects the 1 MHz oscillator.
- External Selects the external clock
- Custom selects the programmable clock.
- Bus Clock selects the emulator clock, the bus clock of the emulating MCU.

If you select External, make sure to connect the TT_OSC clip (white) of the pod B cable to the external clock source.

NOTE If you specify a custom clock speed, be aware that the FSICEBASE can provide clock speeds from 4100Hz to 40MHz in steps of 5kHz. The FSICEBASE uses a clock synthesis chip to generate the clock speed. This method is not as accurate as a crystal: only within about 0.75% absolute frequency with about 5% jitter. If you choose a clock value (either from one of the radio buttons or by typing it in) that is an integer divisor of 32MHz, you will get an accurate crystal-sourced clock.

Collecting Bus Data

To instruct the Bus State Analyzer (BSA) to start collecting data:

1. From debugger main menu, select **FSCICEBASE-HC08**
2. Select **Arm Trace** — the BSA begins to collect data when the debugger starts execution of the loaded application, depending on the triggering mode. The BSA uses the events that you defined in the Bus State Analyzer Configuration dialog box.

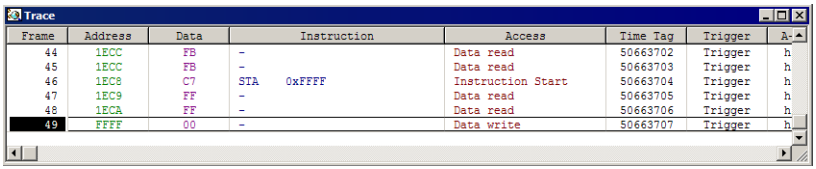
Viewing Data

You can view the data collected by the BSA in several formats. You can view:

- raw data,
- disassembled instructions,
- mixed raw data and disassembled instructions, and
- source code.

1. From debugger main menu, select **FSCICEBASE-HC08**
2. Select **Trace** — the **Trace** window opens (Figure 6.2)

Figure 6.2 Trace Window



Trace

Frame	Address	Data	Instruction	Access	Time Tag	Trigger	A-
44	1ECC	FB	-	Data read	50663702	Trigger	h
45	1ECC	FB	-	Data read	50663703	Trigger	h
46	1EC8	C7	STA 0xFFFF	Instruction Start	50663704	Trigger	h
47	1EC9	FF	-	Data read	50663705	Trigger	h
48	1ECA	FF	-	Data read	50663706	Trigger	h
49	FFFF	00	-	Data write	50663707	Trigger	h

3. To change the kinds of data and the way that data is displayed:
   a. Place mouse cursor over Trace window
   b. Right-click mouse

      Menu appears allowing you to change various aspects of the Trace window

The Trace window can display trace buffer contents as raw bus cycles, as disassembled instructions, as mixed instructions and raw bus cycles, or as source code.

A

S-Record Format

This appendix explains the S-record format. Motorola created the S-record format so that programs and data files could be encoded in a printable format. This allows for easier transportation between computer systems. The S-record format makes it easier to monitor the transport of the programs and files. Another benefit of the format is that it can be easily edited.

This appendix contains the following sections:

- S-Record Content
- S-Record Types
- Creating S-records
- Example

S-Record Content

S-records are essentially character strings grouped into fields. An S-record consists of five fields. The fields identify:

- the record type,
- the record length,
- a memory address,
- code or data, and
- a checksum.

Table A.1 explains the purpose of each field, and shows the number of possible characters that can make up the field.

Table A.1 Description of S-Record Fields

Field	Printable Characters	Contents
Type	2	S-record type — such as S0, S1, etc.
Record Length	2	Character pair count in the record, excluding the type and record length

Table A.1 Description of S-Record Fields

Field	Printable Characters	Contents
Address	4, 6, or 8	2-, 3-, or 4-byte address at which the data field is to be loaded into memory
Code/Data	0-2n	From 0 to n bytes of executable code, memory loadable data, or descriptive information. For compatibility with teletypewriter, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S-record)
Checksum	2	Least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields

The format specifies that each byte of binary data must be encoded as a two-character hexadecimal number:

- the first character represents the high-order four bits
- the second character represents the low-order four bits of the byte

Each record may be terminated with a CR/LF/NULL. Additionally, an S-record may have an initial field to accommodate other data such as line numbers generated by a time-sharing system.

The record length (byte count) and checksum fields help to verify whether the S-record is transmitted accurately.

S-Record Types

Eight types of S-records have been defined to accommodate the several needs of the encoding, transportation, and decoding functions. Table A.2 describes the different types.

The various upload, download, and other record transportation control commands, as well as cross assemblers, linkers, and other file-creating or debugging tools, use only those S-records which serve the purpose of the tool.

For specific information on which S-records are supported by a particular tool, the user manual for that program must be consulted.

NOTE The Freescale ICE Base (FSICEBASE) supports only the S0, S1, and S9 record types. All data before the first S1 record is ignored. Thereafter, all records must be S1 type until the S9 record, which terminates data transfer.

Table A.2 S-Record Types

Type	Description
S0	Header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. The address field is normally zeros.
S1	Code/data record and the two-byte address at which the code/data is to reside.
S2-S8	Not supported by the FSICEBASE.
S-9	Termination record for a block of S1 records. Address field may optionally contain the two-byte address of the instruction to which control is to be passed. If not specified, the first interplant specification encountered in the input will be used. There is no code/data field.

Only one termination record is used for each block of s-records. Normally, only one header record is used, although it is possible for multiple header records to occur.

Creating S-records

S-record format programs may be produced by dump utilities, debuggers, cross assemblers, or cross linkers. Several programs are available for downloading a file in S-record format from a host system to an 8-bit or 16-bit microprocessor-based system.

Example

A typical S-record format looks like this when printed or displayed:

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S113001000020008000825290018538123410018 13
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The above format consists of an S0 header record, four S1 code/data records, and an S9 termination record.

B

FSICEBASE Debugger Scripting Commands

This appendix lists debugger commands that are specific to the FSICEBASE. You can use these commands from the command line of the debugger. You can also use these commands to perform actions from a script.

Command Reference

The following are commands specific to the FSICEBASE emulation platform. You must prefix them with “gdi”, for example “gdi arm” will arm the BSA. The format for many of the commands can be viewed by typing “gdi <command> help”.

ARM
DISARM
CT
ST
STA
STB
STC
STD
TD
TE
SC
SQ
OSC
SIG
MEM
LOADMAP
FILLMEM

Index

A

abbreviations 8
add files 26
arming trace 72
assigning IP address 38

B

breakpoints 60
build 26, 27
Bus State Analyzer 67
    arming 72
    cables 14
    defining events 68
    logic cables 44
    selecting record mode 70
    viewing data 73

C

clock speed
    setting 42
    time tag 72
CodeWarrior
    installing 17
collecting data 72
command pane 33
communication
    establish connection 28
    USB 29
compile
    options 26
    project 27
components 13
configuration
    system setup 35
connections
    logic cables 44
    MON08 50
create project 20

D

data
    viewing trace 73
debugger
    MON08 debug port 50
default IP address 38
documentation 8, 13

E

emulation module (EM)
    defined 14
events
    defining 68
    record modes 70

F

features 12
files
    add to project 26
FSICEBASE
    communication 28
    components 13
    features 12
    IP address 38
    requirements 11

G

getting started 15

H

hardware
    components 13
    MON08 49
    setup 15

I

indicators
    emulation LEDs 17
    MON08 debug port LEDs 53
initialization
    system 35
install software 17
IP address

default 38

L

LED indicators
emulation LEDs 17
MON08 debug port LEDs 53

M

memory
change values 46
viewing 57
memory map 40
modifying 41
viewing 40
MON08 debug port
connecting 50

P

project
add files 26
compile or build 27
create 20

R

real-time memory
display real-time memory 57
recording modes 70
requirements 11

S

select
clock speed 42
memory map 41
setup 15
setup system 15
software installation 17
system components 13
system features 12
system requirements 11

T

target cable 14
target head adapter
as additional component 14
term 68
time tag 72
trace
arming 72
time tag 72
viewing data 73

V

viewing data 73

W

watchpoints
setting 60
wizard, project create 20