IAR C-SPY® Hardware Debugger Systems

User Guide

for Renesas

E1/E20 and E100 Emulators

CSE100HW-4

EDITION NOTICE

Fourth edition: December 2011

Part number: CSE100HW-4

This guide applies to version 3.5 of IAR Embedded Workbench® for Renesas M16C/1X–3X, 5X–6X and R8C Series of CPU cores.

Internal reference: IJOA, Too6.0.3

Contents

Preface

Welcome to the *IAR C-SPY® Hardware Debugger Systems User Guide for Renesas E1/E20 and E100 Emulators*. The purpose of this guide is to provide you with detailed reference information that can help you use the features in the IAR C-SPY® Hardware Debugger Systems.

Who should read this guide

You should read this guide if you want to get the most out of the features in the C-SPY hardware debugger systems. In addition, you should have working knowledge of:

- The C or C++ programming language
- Application development for embedded systems
- The architecture and instruction set of the target processor (refer to the chip manufacturer's documentation)
- The operating system of your host machine.

This guide also assumes that you already have working knowledge of the target system you are using, as well as some working knowledge of the IAR C-SPY Debugger. For a quick introduction to the IAR C-SPY Debugger, see the tutorials available in the *IAR Embedded Workbench® IDE User Guide*.

How to use this guide

This guide describes the C-SPY interface to the target system you are using; this guide does not describe the general features available in the IAR C-SPY debugger or the hardware target system. To take full advantage of the whole debugger system, you must read this guide in combination with:

- The *IAR Embedded Workbench® IDE User Guide* which describes the general features available in the C-SPY debugger
- The documentation supplied with the target system you are using.

Note that additional features might have been added to the software after the *IAR C-SPY® Hardware Debugger Systems User Guide* was printed. The release notes contain the latest information.

What this guide contains

Below is a brief outline and summary of the chapters in this guide.

- *Introduction to C-SPY® hardware debugger systems* introduces you to the C-SPY hardware debugger system. The chapter briefly shows the difference in functionality provided by the different available C-SPY drivers.
- *Hardware-specific debugging* describes the additional options, menus, and features provided by these debugger systems.

Other documentation

The complete set of IAR development tools for the target processor are described in a series of guides. These guides can be found in the `m16c\doc` directory or reached from the **Help** menu.

All of these guides are delivered in hypertext PDF or HTML format on the installation media. Some of them are also delivered as printed books.

Recommended web sites:

- The Renesas web site, **www.renesas.com**, contains information and news about the M16C/1X–3X, 5X–6X and R8C Series of CPU cores.
- The IAR Systems web site, **www.iar.com**, holds application notes and other product information.

Document conventions

When, in this text, we refer to the programming language C, the text also applies to C++, unless otherwise stated.

When referring to a directory in your product installation, for example `m16c\doc`, the full path to the location is assumed, for example `c:\Program Files\IAR Systems\Embedded Workbench 6.`*`n`*`\m16c\doc`.

TYPOGRAPHIC CONVENTIONS

This guide uses the following typographic conventions:

Style	**Used for**
`computer`	• Source code examples and file paths. • Text on the command line. • Binary, hexadecimal, and octal numbers.

Table 1: Typographic conventions used in this guide

Style	Used for
`parameter`	A placeholder for an actual value used as a parameter, for example *`filename`*`.h` where *`filename`* represents the name of the file.
`[option]`	An optional part of a command.
`[a|b|c]`	An optional part of a command with alternatives.
`{a|b|c}`	A mandatory part of a command with alternatives.
bold	Names of menus, menu commands, buttons, and dialog boxes that appear on the screen.
italic	• A cross-reference within this guide or to another guide. • Emphasis.
…	An ellipsis indicates that the previous item can be repeated an arbitrary number of times.
🛠	Identifies instructions specific to the IAR Embedded Workbench® IDE interface.
🖥	Identifies instructions specific to the command line interface.
💡	Identifies helpful tips and programming hints.
⚠	Identifies warnings.

Table 1: Typographic conventions used in this guide (Continued)

NAMING CONVENTIONS

The following naming conventions are used for the products and tools from IAR Systems® referred to in this guide:

Brand name	Generic term
IAR Embedded Workbench® for M16C/R8C	IAR Embedded Workbench®
IAR Embedded Workbench® IDE for M16C/R8C	the IDE
IAR C-SPY® Debugger for M16C/R8C	C-SPY, the debugger
IAR C-SPY® Simulator	the simulator
IAR C/C++ Compiler™ for M16C/R8C	the compiler
IAR Assembler™ for M16C/R8C	the assembler
IAR XLINK Linker™	XLINK, the linker
IAR XAR Library Builder™	the library builder
IAR XLIB Librarian™	the librarian
IAR DLIB Library™	the DLIB library

Table 2: Naming conventions used in this guide

Brand name	Generic term
IAR CLIB Library™	the CLIB library

Table 2: Naming conventions used in this guide (Continued)

Introduction to C-SPY® hardware debugger systems

This chapter introduces you to the C-SPY hardware debugger systems and to how they differ from the C-SPY Simulator.

The chapters specific to C-SPY hardware debugger systems assume that you already have some working knowledge of the target system you are using, as well as of the IAR C-SPY Debugger.

The C-SPY hardware debugger systems

C-SPY consists of both a general part which provides a basic set of C-SPY features, and a driver. The C-SPY driver is the part that provides communication with and control of the target system. The driver also provides a user interface—special menus, windows, and dialog boxes—to the functions provided by the target system, for instance special breakpoints. This driver is automatically installed during the installation of IAR Embedded Workbench.

At the time of writing this guide, the IAR C-SPY Debugger for the M16C/R8C Series CPU cores is available with drivers for these target systems and evaluation boards:

- Simulator
- Renesas E100 Emulator
- Renesas PC7501 Emulator
- Renesas Compact Emulator
- Renesas E1/E20 Emulator
- Renesas E8 Emulator
- Renesas E8a Emulator.

For further details about the concepts that are related to C-SPY and general debugger features, see the *IAR Embedded Workbench® IDE User Guide*.

For information about the other drivers, see the separate documentation for those drivers.

DIFFERENCES BETWEEN THE C-SPY DRIVERS

This table summarizes the key differences between the simulator and the E1/E20 and E100 drivers:

Feature	Simulator	E100	E1/E20
Code breakpoints	Unlimited	x	x
Data breakpoints	x	x	x
Execution in real time	--	x	x
Zero memory footprint	x	x	--
Simulated interrupts	x	--	--
Real interrupts	--	x	x
Interrupt logging	x	--	--
Cycle counter	x	--	--
Code coverage	x	x	--
Data coverage	x	--	--
Function/instruction profiler	x	x	--
Profiling	x	x	--
Trace	x	x	x

Table 3: Driver differences

The C-SPY E1/E20 driver

The C-SPY E1/E20 driver is automatically installed during the installation of IAR Embedded Workbench. Using the C-SPY E1/E20 driver, C-SPY can connect to an E1/E20. All M16C/R8C CPU cores have built-in, on-chip debug support. Because the hardware debugger logic is built into the CPU core, no ordinary ROM-monitor program or extra specific hardware is needed to make the debugging work.

COMMUNICATION OVERVIEW

The C-SPY E1/E20 driver uses USB to communicate with the E1/E20. The E1/E20 communicates with the Front-end firmware (FFW) interface module. The FFW interface module, in turn, communicates with the Back-end firmware (BFW) module on the emulator.

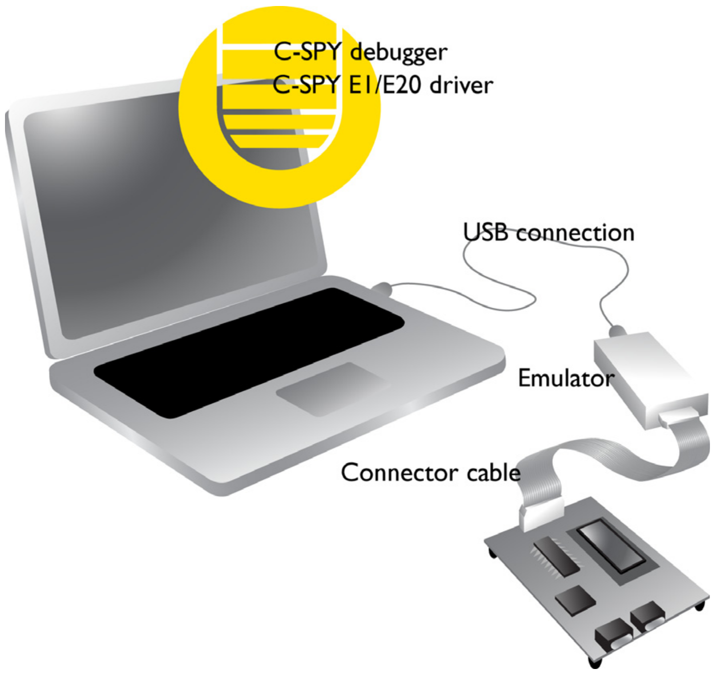


Figure 1: C-SPY E1/E20 communication overview

The connector cable is a 14-pin cable. For the E20, you need a 38-pin-to-14-pin conversion adapter.

For more information, see the documentation supplied with the E1/E20.

When a debug session is started, your application is automatically downloaded and programmed into flash memory. You can disable this feature, if necessary.

HARDWARE INSTALLATION

USB drivers are automatically installed during the installation of IAR Embedded Workbench. If you need to re-install them, they are available both on the installation CD and in the `m16c\drivers\Renesas\` directory in the installation directory.

For more information about the hardware installation, see the documentation supplied with the E1/E20 from Renesas. The following power-up sequence is recommended to ensure proper communication between the target board, E1/E20, and C-SPY:

1. Power up the target board.
2. Start the C-SPY debugging session.

The C-SPY E100 driver

The C-SPY E100 driver is automatically installed during the installation of IAR Embedded Workbench. Using the C-SPY E100 driver, C-SPY can connect to E100. To use the hardware debugger, you also need a USB driver. The first time you connect an emulator to the USB port on the host computer, a USB setup wizard will help you locate and install the required USB driver. The driver is located in the `drivers\renesas\e100\`*`os_version`* directory of your IAR Systems product installation.

The C-SPY E100 driver uses the USB port to communicate with the E100.

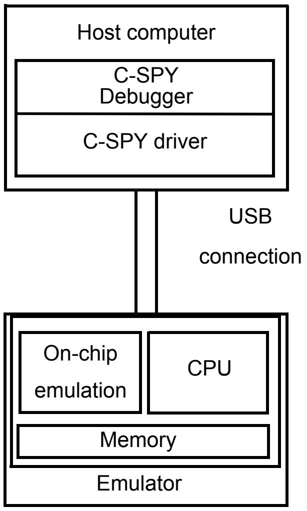


Figure 2: C-SPY E100 communication overview

For further information, refer to the documentation supplied with the E100.

When a debugging session is started, your application is automatically downloaded and programmed into flash memory. You can disable this feature, if necessary.

HARDWARE INSTALLATION

For information about the hardware installation, see the documentation supplied with the E100 from Renesas. The following power-up sequence is recommended to ensure proper communication between the target board, E100, and C-SPY:

1. Power up E100.
2. Install USB drivers if it is the first time that you connect an emulator to the USB port on your host computer.
3. Start the C-SPY debugging session.

Getting started

IAR Embedded Workbench comes with example applications. You can use these examples to get started using the development tools from IAR Systems or simply to verify that contact has been established with your target board. You can also use the examples as a starting point for your application project.

You can find the examples in the `m16c\examples` directory. The examples are ready to be used as is. They are supplied with ready-made workspace files, together with source code files and linker command files.

RUNNING THE DEMO PROGRAM

1. Choose the **Debug** build configuration from the drop-down list at the top of the workspace window.
2. Choose **Project>Options>General Options>Target** and select your device from the **Device** drop-down list.
3. Choose **Project>Options>Debugger>Setup** and select the emulator from the **Driver** drop-down list. For more information about the options, see *Debugger options for debugging using hardware systems*, page 15. Click **OK** to close the **Options** dialog box.
4. Choose **Project>Make** or click the **Make** button to compile and link the source code.
5. To start C-SPY, click the **Debug** button or choose **Project>Debug**. The **Hardware Setup** dialog box appears. Click **OK**.
6. To open the Terminal I/O window, choose **View>Terminal I/O**.
7. Choose **Debug>Go** or click the **Go** button to start the execution.
8. Click the **Stop** button to stop the execution or wait until program exit is reached.

Hardware-specific debugging

This chapter describes the additional options, menus, and features provided by the C-SPY® hardware debugger systems. The chapter contains the following sections:

- Debugger options for debugging using hardware systems
- E1/E20-specific debugging
- E100-specific debugging
- Using the trace system
- Using the profiler
- Using breakpoints
- C-SPY system macros
- RAM-monitor
- Start debugging a running application
- Resolving problems.

Debugger options for debugging using hardware systems

Before you start any C-SPY hardware debugger you must set some options for the debugger system—both C-SPY generic options and options required for the hardware system (C-SPY driver-specific options). Follow this procedure:

1 To open the **Options** dialog box, choose **Project>Options**.

2 To set C-SPY generic options and select a C-SPY driver:

- Select **Debugger** from the **Category** list

- On the **Setup** page, select the appropriate C-SPY driver from the **Driver** drop-down list.

For information about the settings **Setup macros**, **Run to**, and **Device descriptions**, see the *IAR Embedded Workbench® IDE User Guide*.

For information about the pages **Extra Options** and **Plugins**, see the *IAR Embedded Workbench® IDE User Guide*. For information about the **Images** page, see the online help system.

Note that a default device description file and linker configuration file is automatically selected depending on your selection of a device on the **General Options>Target** page.

3 To set the driver-specific options, select the appropriate driver from the **Category** list. Depending on which C-SPY driver you are using, different sets of available option pages appear.

4 When you have set all the required options, click **OK** in the **Options** dialog box.

DOWNLOAD

The general **Download** options let you modify the behavior of the download.

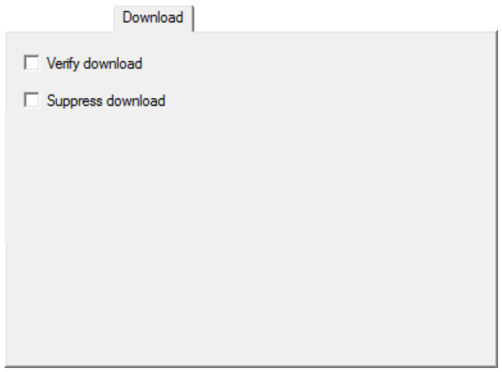


Figure 3: General C-SPY Download options

Verify download

Use this option to verify that the downloaded code image can be read back from target memory with the correct contents.

Suppress download

Use this option to debug an application that already resides in target memory. When this option is selected, the code download is disabled, while preserving the present content of the flash.

If this option is combined with the **Verify download** option, the debugger will read back the code image from non-volatile memory and verify that it is identical to the debugged program.

COMMUNICATION

On the driver-specific **Project>Options>*Driver*>Communication** page you can set part of the options specific to the selected C-SPY driver.

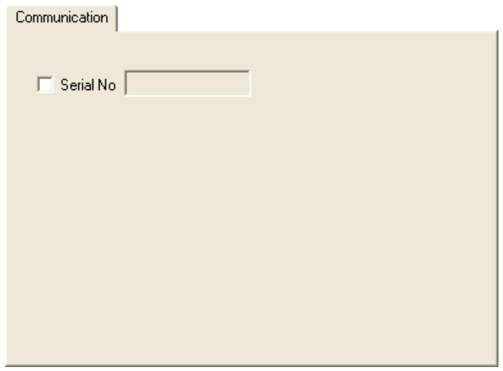


Figure 4: Communication options

Serial No

If more than one Renesas emulator is connected, choose which one to use with the **Serial No** option. Select the option and type the serial number of the emulator.

E1/E20-specific debugging

In this section you can read about the features specific for the C-SPY E1/E20 driver:

- *E1/E20 Emulator menu*, page 18
- *Download (for E1/E20)*, page 19
- *Download Emulator Firmware dialog box*, page 20
- *Hardware Setup dialog box (E1/E20)*, page 21.

E1/E20 EMULATOR MENU

When you are using the C-SPY E1/E20 driver, the **E1/E20 Emulator** menu is added to the menu bar.

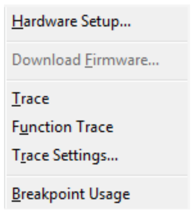


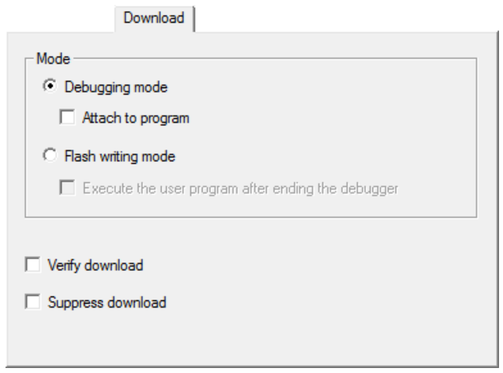
Figure 5: The E1/E20 Emulator menu

These commands are available on the menu:

Hardware Setup	Displays a dialog box where you can configure how the emulator operates, see *Hardware Setup dialog box (E1/E20)*, page 21.
Download Firmware	Displays a dialog box where you can update the firmware of your emulator if needed, see *Download Emulator Firmware dialog box*, page 20.
Trace	Opens a window which displays the collected trace data, see *Trace window*, page 30.
Function Trace	Opens a window which displays the trace data for function calls and function returns, see *Function Trace window*, page 33.
Trace Settings	Displays a dialog box where you can configure the trace generation and collection, see *Trace Settings dialog box*, page 29.
Breakpoint Usage	Displays a dialog box which lists all active breakpoints, see *Breakpoint Usage dialog box*, page 53.

DOWNLOAD (FOR E1/E20)

By default, C-SPY downloads the application to RAM or flash memory when a debug session starts. The **E1/E20>Download** options let you modify the behavior of the download.



(● Debugging mode)
[ ] Attach to program
( ) Flash writing mode
[ ] Execute the user program after ending the debugger
[ ] Verify download
[ ] Suppress download

Figure 6: C-SPY Download options (E1/E20)

Debugging mode

Makes the emulator operate as a debugger. After downloading your application, you cannot disconnect the debugger and use the target system as a stand-alone unit. In this mode, you cannot write ID Codes (that protect the memory from being accessed) to the flash memory.

In this mode, the emulator writes to the OFS, OFS2, and ID code areas.

Attach to program

Makes the debugger attach to a running application at its current location, without resetting the target system. To avoid unexpected behavior when using this option, the **Debugger>Setup** option **Run to** should be deselected.

For more information, see *Start debugging a running application*, page 59.

Flash writing mode

Makes the emulator operate as a flash memory programmer. You cannot use the emulator as a debugger in this mode. After downloading your application, an ID Code (that protects the memory from being accessed) is written to the flash memory.

Note: If any other ID code than `FFFFFFFFFFFFFF` was written to the flash memory, you will be forced to enter that ID code when prompted, to be able to debug the application afterwards.

Execute the user program after ending the debugger

Launches your application on the target board when the code has been downloaded.

Verify download

Use this option to verify that the downloaded code image can be read back from target memory with the correct contents.

Suppress download

Use this option to debug an application that already resides in target memory. When this option is selected, the code download is disabled, while preserving the present content of the flash.

If this option is combined with the **Verify download** option, the debugger will read back the code image from non-volatile memory and verify that it is identical to the debugged program.

DOWNLOAD EMULATOR FIRMWARE DIALOG BOX

The **Download Emulator Firmware** dialog box is available from the **E1/E20 Emulator** menu.

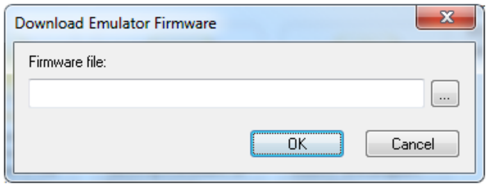


Figure 7: Download Emulator Firmware dialog box

This dialog box is available for the C-SPY E1/E20 driver.

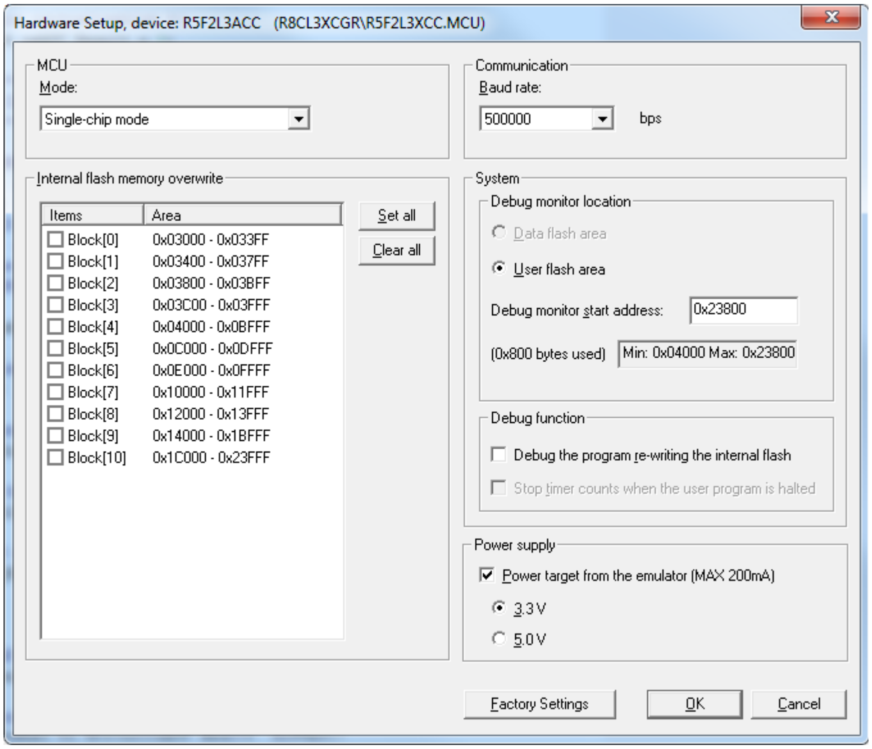
Use this dialog box to update the firmware of your emulator if needed. The only reason for doing this manually is if the automatically downloaded firmware is not working correctly.

Firmware file

Browse to the firmware file on your host computer and click **OK** to download it to your emulator hardware. The emulator firmware files have the filename extension `.s` and are located in subdirectories of the `m16c\config\Renesas\` directory of your product installation.

HARDWARE SETUP DIALOG BOX (E1/E20)

The **Hardware Setup** dialog box for E1/E20 emulators is available from the **E1/E20 Emulator** menu.



Hardware Setup, device: R5F2L3ACC (R8CL3XCGR\R5F2L3XCC.MCU)

MCU
Mode: Single-chip mode

Communication
Baud rate: 500000 bps

Internal flash memory overwrite

Items	Area
☐ Block[0]	0x03000 - 0x033FF
☐ Block[1]	0x03400 - 0x037FF
☐ Block[2]	0x03800 - 0x03BFF
☐ Block[3]	0x03C00 - 0x03FFF
☐ Block[4]	0x04000 - 0x0BFFF
☐ Block[5]	0x0C000 - 0x0DFFF
☐ Block[6]	0x0E000 - 0x0FFFF
☐ Block[7]	0x10000 - 0x11FFF
☐ Block[8]	0x12000 - 0x13FFF
☐ Block[9]	0x14000 - 0x1BFFF
☐ Block[10]	0x1C000 - 0x23FFF

System
Debug monitor location
Data flash area [✓ User flash area]
Debug monitor start address: 0x23800
(0x800 bytes used) Min: 0x04000 Max: 0x23800

Debug function
☐ Debug the program re-writing the internal flash
☐ Stop timer counts when the user program is halted

Power supply
☑ Power target from the emulator (MAX 200mA)
[✓ 3.3 V] 5.0 V

Figure 8: Hardware Setup dialog box (E1/E20)

Use this dialog box to make settings that control how the emulator operates. Before C-SPY is started for the first time in a new project, and when you change devices, the hardware must be configured.

Mode

Controls the MCU operation based on the pin settings. The only available setting is **Single-chip mode**.

Baud rate

Choose the communication speed between the E1/E20 emulator and the MCU.

Always try the default speed (500,000 bps) first. Depending on the wired length of the MODE signal and how it is wired on the user system, you might need to reduce the speed.

The communication speed cannot be changed during emulator operation. To change the speed, you must disconnect the emulator and the MCU temporarily and then reconnect them. (At communication speeds below 57,600 bps, writing to flash takes a long time and the emulator might appear to be giving no response.)

Note: When large amounts of data are displayed or handled in a C-SPY memory window, a time-out error might occur because the communication takes up much time.

Internal flash memory overwrite

Allows you to specify whether individual blocks of flash memory should be overwritten by the downloading process, when the debug image is written to the target MCU.

Selected blocks will keep their previous contents after the downloading process and be merged with the downloaded data.

Deselected blocks will be erased before download and replaced by the contents of the downloaded image. If the downloaded image does not have any data for a memory area, it will contain the initial values of the flash memory.

These settings are used also for writing in the on-chip flash memory mode.

Debug monitor location

Specifies the area in which the debug monitor is located.

Data flash area	Places the debug monitor in the data flash area. If you select this area, reduce the communication speed to 250,000 bps or less. For some devices, this setting is not available.
User flash area	Places the debug monitor in the user flash area.
Debug monitor start address	Specifies the start address of the debug monitor location. For some devices, this setting is unnecessary and unavailable.

Debug function

Selects the debug function. The available options are:

Debug the program re-writing the internal flash	Select this option when you debug in CPU rewrite mode (which involves your application overwriting the internal flash memory of the MCU). This is necessary to make the displayed information in the IDE and the contents of the MCU's internal ROM match.
Stop timer counts when the user program is halted	Select this option to stop all timer counts except that of the watchdog timer, when the application you are debugging has stopped executing.

Memory accesses when C-SPY execution is halted are always made to the internal cache of the emulator. Actual accesses to flash memory are made before the application restarts and immediately after it stops.

Note: When you debug using the option **Debug the program re-writing the internal flash**, software breakpoints cannot be used and you should not rewrite the internal ROM area displayed in the memory windows.

Power target from the emulator

Select this option and the correct voltage if you are supplying the target board with power from the E1/E20 emulator, and not from an external power supply.

If you select this option but connect an external power supply to the target board, the external power supply will be used instead and these settings will be ignored.

E100-specific debugging

In this section you can read about the features specific for the C-SPY E100 driver:

- *The E100 Emulator menu*, page 24
- *Hardware Setup (E100), page 25*
- *Download Firmware*, page 28.

THE E100 EMULATOR MENU

When you are using the C-SPY E100 driver, the **E100 Emulator** menu appears in C-SPY.

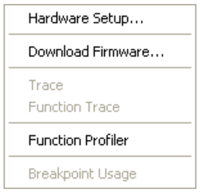


Figure 9: The E100 Emulator menu

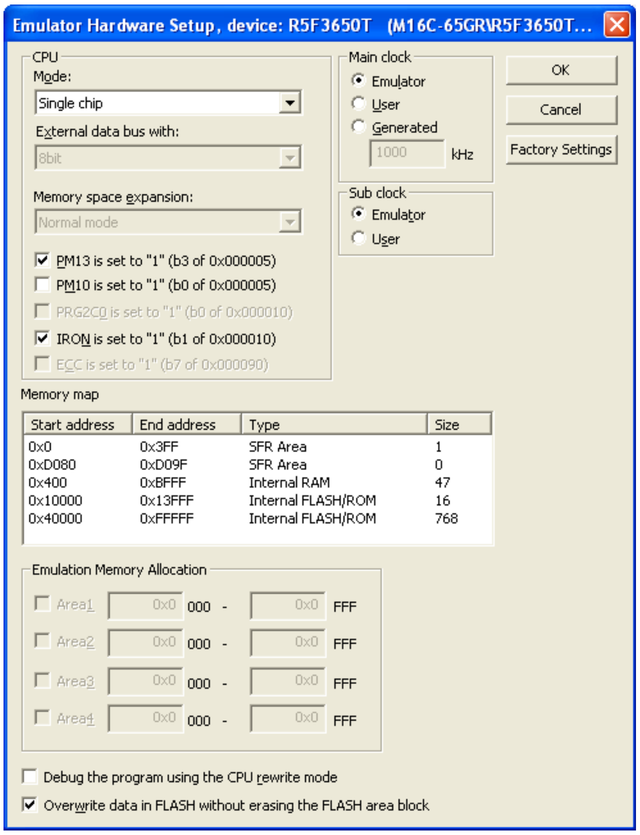
These commands are available on the **E100 Emulator** menu:

Menu command	Description
Hardware Setup	Displays the **Hardware Setup** dialog box, where the basic configuration for the emulator is done. For more information, see *Hardware Setup (E100), page 25*.
Download Firmware	Opens the **Download Firmware** dialog box for selecting a firmware file to download to the target board. For more information, see *Download Firmware*, page 28.
Trace	Opens the Trace window which displays the recorded trace data; see *Trace window*, page 30.
Function Trace	Opens the Function Trace window which displays the trace data for which functions were called or returned from; see *Function Trace window*, page 33.
Function Profiler	Opens the Function Profiler window which displays function profiling information, see *Function Profiler window*, page 38. To use function profiling in the simulator, you must include debug information in the debug file.
Breakpoint Usage	Displays the **Breakpoint Usage** dialog box which lists all active breakpoints; see *Breakpoint Usage dialog box*, page 53.

Table 4: Commands on the E100 menu

HARDWARE SETUP (E100)

In the **Hardware Setup** dialog box—available from the **E100 Emulator** menu—you can configure the emulator debugger.


Emulator Hardware Setup, device: R5F3650T (M16C-65GR\R5F3650T...

CPU
Mode: Single chip
External data bus with: 8bit
Memory space expansion: Normal mode

[✓] PM13 is set to "1" (b3 of 0x000005)
[ ] PM10 is set to "1" (b0 of 0x000005)
[ ] PRG2C0 is set to "1" (b0 of 0x000010)
[✓] IRON is set to "1" (b1 of 0x000010)
[ ] ECC is set to "1" (b7 of 0x000090)

Main clock: [● Emulator] User Generated 1000 kHz

Sub clock: [● Emulator] User

Memory map

Start address	End address	Type	Size
0x0	0x3FF	SFR Area	1
0xD080	0xD09F	SFR Area	0
0x400	0xBFFF	Internal RAM	47
0x10000	0x13FFF	Internal FLASH/ROM	16
0x40000	0xFFFFF	Internal FLASH/ROM	768

Emulation Memory Allocation

[ ] Area1	0x0 000	-	0x0 FFF
[ ] Area2	0x0 000	-	0x0 FFF
[ ] Area3	0x0 000	-	0x0 FFF
[ ] Area4	0x0 000	-	0x0 FFF

[ ] Debug the program using the CPU rewrite mode
[✓] Overwrite data in FLASH without erasing the FLASH area block

Figure 10: Emulator Hardware Setup dialog box (E100)

Mode

Use this option to set an operation mode. These modes are available:

- **Single chip**
- **Memory expansion**
- **Microprocessor mode**.

External data bus width

Use this option to specify the width of the external data bus. You can choose between:

- **8 bits**
- **16 bits (initial value)**.

This option is available when you have selected one of the operation modes **Memory expansion** or **Microprocessor mode**.

Memory space expansion

Use this option to specify that the hardware has extended memory. You can choose between:

- **Normal mode (initial value)**
- **4 Mbyte Mode**.

This option is available when you have selected one of the operation modes **Memory expansion** or **Microprocessor mode**.

PM10 is set to "1" (b0 of 0x000005)

Use his option to toggle the setting of the CS2 area. By default, the CS2 area select bit—the third bit of the processor mode register—is set to `0`. This option sets the CS2 area select bit to `1`. When this option is selected, data flash memory can be used.

PM13 is set to "1" (b3 of 0x000005)

Use this option to expand the internal reserved memory area. This option sets the control bit—bit zero of the processor mode register—to `1`. By default, the internal reserved area expansion bit is set to `0`. When this option is selected, flash memory larger than 192 Kbytes and RAM larger than 16 Kbytes can be used.

PRG2C0 is set to "1" (b0 of 0x000010)

Use this option to enable or disable the ROM 2 area. This option sets the Program ROM 2 disable bit—bit zero of the program 2 area control register—to `1`. If you set the bit to `0`, the ROM 2 area is enabled. By default, this bit is set to `0`.

IRON is set to "1" (b1 of 0x000010)

Use this option to enable or disable the ROM 1 area. This option sets the first bit of the program 2 area control register to `1`.

ECC is set to "1" (b7 of 0x000090)

Use this option to enable or disable Error Check and Correct for the E2dataFlash memory. This option sets bit 7 of the E2dataFlash Mode (E2FM) register to `1`. By default, this bit is set to `0`.

Main clock

Use the **Main clock** options to set the CPU clock source:

- Select **Emulator** to use the emulator as the CPU clock
- Select **User** to use the target clock as the CPU clock
- Select **Generated** to use a clock generated by the E100 Emulator, running at a frequency between 1000 kHz and 99,999 kHz. Type your desired frequency.

Sub clock

Use the **Sub clock** options to set the CPU sub clock source:

- Select **Emulator** to use the emulator as the CPU sub clock
- Select **User** to use the target clock as the CPU sub clock.

Memory map

The memory map displays available internal memory for the device you have chosen.

To edit an existing memory area, use the **Emulation Memory Allocation** options.

Emulation Memory Allocation

Use these options to allocate memory for up to four emulation memory areas. Select a text box for the area to be used, and type the start address and the end address in hexadecimal notation.

The addresses can be set in a unit of 4 Kbytes. Therefore, the low 12 bits of the start address and the end address are fixed.

These options are not available when single chip mode has been selected.

Debug the program using the CPU rewrite mode

This option specifies whether or not to debug programs that run in the CPU Rewrite Mode.

Overwrite data in FLASH without erasing the FLASH area block

If you select this option, writing to the flash memory area will merge the new data with the previous flash contents, and the addresses that are not being written to will keep their previous contents.

Factory Settings

Click the **Factory Settings** button to restore the factory settings.

DOWNLOAD FIRMWARE

If you need to download new firmware, for example if you are changing the processor configuration or if you need to upgrade the firmware, choose the **E100 Emulator>Download Firmware** command. The emulator firmware files have the filename extension `.s` and are located in the subdirectories of the `m16c\config\Renesas\` directory of your product installation.

Using the trace system

This section gives you information about using the trace system. More specifically, these topics are covered:

- *Reasons for using the trace system*, page 28
- *Briefly about the trace system*, page 29
- *How to use the trace system*, page 29
- *Trace Settings dialog box*, page 29
- *Trace window*, page 30
- *Function Trace window*, page 33
- *Find In Trace window*, page 34
- *Find in Trace dialog box*, page 34.

REASONS FOR USING THE TRACE SYSTEM

By using the trace system, you can trace the program flow up to a specific state, for instance an application crash, and use the trace information to locate the origin of the problem. Trace information can be useful for locating programming errors that have irregular symptoms and occur sporadically. Trace information can also be useful as test documentation.

BRIEFLY ABOUT THE TRACE SYSTEM

In C-SPY, a *trace* is a recorded sequence of executed machine instructions. The Function Trace window only shows trace data corresponding to calls to and returns from functions, whereas the Trace window displays all instructions.

Note: For the E1/E20 driver, trace is limited to up to four branch instructions (branch source/destination PC).

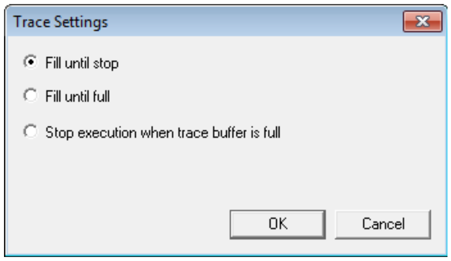
HOW TO USE THE TRACE SYSTEM

Start the execution. When the execution stops, for instance because the program exit is reached or a breakpoint is triggered, trace data is displayed in the Trace window. For more information about the window, see *Trace window*, page 30.

For more information about using the generic features in the trace system, see the *IAR Embedded Workbench® IDE User Guide.*

TRACE SETTINGS DIALOG BOX

The **Trace Settings** dialog box is available from the **E1/E20 Emulator** menu.



(●) Fill until stop
( ) Fill until full
( ) Stop execution when trace buffer is full

Figure 11: Trace Settings dialog box

This dialog box is available for the E1/E20 driver.

Use this dialog box to configure the E1/E20 trace generation and collection.

Fill until stop

Continues to collect trace data until the execution stops. The last four branches that occurred before the stop are recorded.

Fill until full

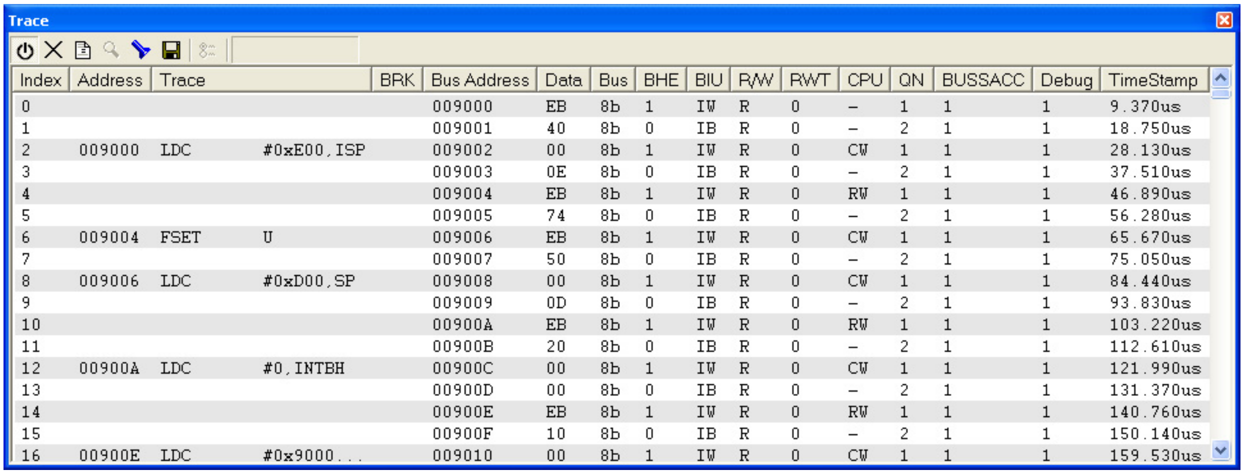
Continues to collect trace data until the buffer is full. The first four branches that occurred are recorded.

Stop emulator when trace buffer is full

Halts the execution when the trace buffer is full.

TRACE WINDOW

The Trace window—available from the **Driver** menu—displays a recorded sequence of trace data.



Trace

Index	Address	Trace	BRK	Bus Address	Data	Bus	BHE	BIU	R/W	RWT	CPU	QN	BUSSACC	Debug	TimeStamp
0				009000	EB	8b	1	IW	R	0	–	1	1	1	9.370us
1				009001	40	8b	0	IB	R	0	–	2	1	1	18.750us
2	009000	LDC #0xE00,ISP		009002	00	8b	1	IW	R	0	CW	1	1	1	28.130us
3				009003	0E	8b	0	IB	R	0	–	2	1	1	37.510us
4				009004	EB	8b	1	IW	R	0	RW	1	1	1	46.890us
5				009005	74	8b	0	IB	R	0	–	2	1	1	56.280us
6	009004	FSET U		009006	EB	8b	1	IW	R	0	CW	1	1	1	65.670us
7				009007	50	8b	0	IB	R	0	–	2	1	1	75.050us
8	009006	LDC #0xD00,SP		009008	00	8b	1	IW	R	0	CW	1	1	1	84.440us
9				009009	0D	8b	0	IB	R	0	–	2	1	1	93.830us
10				00900A	EB	8b	1	IW	R	0	RW	1	1	1	103.220us
11				00900B	20	8b	0	IB	R	0	–	2	1	1	112.610us
12	00900A	LDC #0,INTBH		00900C	00	8b	1	IW	R	0	CW	1	1	1	121.990us
13				00900D	00	8b	0	IB	R	0	–	2	1	1	131.370us
14				00900E	EB	8b	1	IW	R	0	RW	1	1	1	140.760us
15				00900F	10	8b	0	IB	R	0	–	2	1	1	150.140us
16	00900E	LDC #0x9000...		009010	00	8b	1	IW	R	0	CW	1	1	1	159.530us

Figure 12: Trace window

This figure shows the window for the E100 driver. The window looks different for the E1/E20 driver.

C-SPY generates trace information based on the location of the program counter.

Trace toolbar

The Trace toolbar at the top of the Trace window and in the Function trace window provides these toolbar buttons:

Toolbar button	Description
Enable/Disable	Enables and disables tracing. This button is not available in the Function trace window.
Clear trace data	Clears the trace buffer. Both the Trace window and the Function trace window are cleared.

Table 5: Trace toolbar buttons

Toolbar button	Description
Toggle Source	Toggles the Trace column between showing only disassembly or disassembly together with corresponding source code.
Browse	Toggles browse mode on and off for a selected item in the Trace window. For more information about browse mode, see the *IAR Embedded Workbench® IDE User Guide*.
Find	Opens the **Find In Trace** dialog box where you can perform a search; see *Find in Trace dialog box*, page 34.
Save	Opens a standard **Save As** dialog box where you can save the recorded trace information to a text file, with tab-separated columns.
Progress bar	Displays a backlog of trace data that is still being processed. If the rate of incoming data is higher than the rate of the trace processing the data, a backlog is accumulated. The progress bar indicates that the trace is still processing data, but also approximately how far the trace has come in the process. Note that because the trace consumes data at a certain rate and the target system supplies data at another rate, the amount of data remaining to be processed can both increase and decrease. The progress bar can grow and shrink accordingly.

Table 5: Trace toolbar buttons (Continued)

Trace display area for E1/E20

The display area contains these columns:

Trace window column	Description
#	A serial number for each row in the trace buffer. Simplifies the navigation within the buffer.
Trace	The recorded sequence of executed machine instructions. Optionally, the corresponding source code can also be displayed.

Table 6: E1/E20 Trace window columns

Trace display area for E100

The display area displays trace information in these columns:

Trace window column	Description
Index	A serial number for each row in the trace buffer. Simplifies the navigation within the buffer.

Table 7: E100 Trace window columns

Trace window column	Description
Address	The address of the instruction.
Trace	The recorded sequence of executed machine instructions. Optionally, the corresponding source code can also be displayed.
BRK	A software breakpoint was triggered.
Bus Address	Addresses on the processor bus.
Data	Data of the data bus in hexadecimal notation.
Bus	The external bus width. The width is `8b` when the bus is 8 bits wide, and `16b` when the bus is 16 bits wide.
BHE	The state of the Byte High Enable Signal, BHE. `0`: an odd address is being accessed `1`: an even address is being accessed.
BIU	The state between the Bus Interface Unit (BIU), the memory, and I/O. `-`: No change `DMA`: Data access such as Direct memory Access requested from other than the CPU. `INT`: INTACK sequence start `IB`: Instruction code read (in bytes) requested from the CPU `DB`: Data access (in bytes) requested from the CPU `IW`: Instruction code read (in words) requested from the CPU `DW`: Data access (in words) requested from the CPU.
R/W	The data bus state. `R`: Read state `W`: Write state `-`: No access made
RWT	A signal indicating the valid position of the bus cycle. When the position is valid, the signal is `0`. The Address, Data, and BIU signals are active when this signal is `0`.
CPU	The state between the CPU and the Bus Interface Unit (BIU). `-`: No change `CB`: Op-code read (in bytes) `RB`: Operand read (in bytes) `QC`: Instruction queue buffer clear `CW`: Op-code read (in words) `RW`: Operand read (in words)
QN	The number of bytes stored in the instruction queue buffer, in the range from 0 to 4.

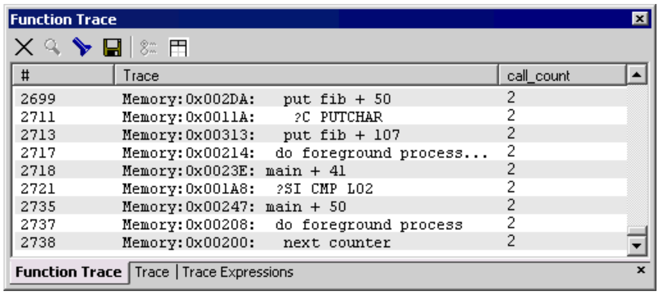
Table 7: E100 Trace window columns (Continued)

Trace window column	Description
BUSSACC	When the memory is accessed by the debugger during the execution of your application, this column shows 0 for cycles in which the MCU bus is occupied by the emulator. **Note:** Your application is suspended while memory is being accessed.
Debug	When the memory is accessed by the debugger during the execution of your application, this column shows 0 for cycles in which the MCU bus is occupied by the emulator. **Note:** Your application is suspended while memory is being accessed.
TimeStamp	Shows the time elapsed since the target program started. Each time your application starts running, the time stamp count is resumed from the previous time stamp. The time stamp is displayed in this format: h:m:s:ms:µs:ns

Table 7: E100 Trace window columns (Continued)

FUNCTION TRACE WINDOW

The Function Trace window—available from the **Driver** menu—displays a subset of the trace data displayed in the Trace window. Instead of displaying all rows, the Function Trace window only shows trace data corresponding to calls to and returns from functions.



#	Trace	call_count
2699	Memory:0x002DA: put fib + 50	2
2711	Memory:0x0011A: ?C PUTCHAR	2
2713	Memory:0x00313: put fib + 107	2
2717	Memory:0x00214: do foreground process...	2
2718	Memory:0x0023E: main + 41	2
2721	Memory:0x001A8: ?SI CMP L02	2
2735	Memory:0x00247: main + 50	2
2737	Memory:0x00208: do foreground process	2
2738	Memory:0x00200: next counter	2

Figure 13: Function Trace window

Toolbar

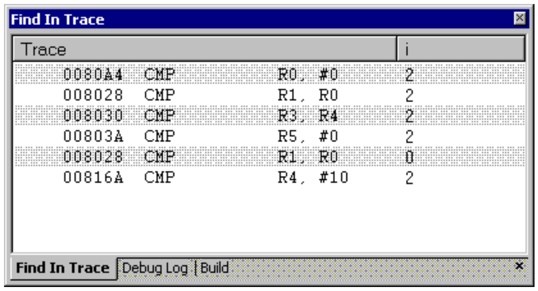
For information about the toolbar, see *Trace toolbar*, page 30.

The display area

For information about the columns in the display area, see *Trace display area for E1/E20*, page 31 and *Trace display area for E100*, page 31, respectively.

FIND IN TRACE WINDOW

The Find In Trace window—available from the **View>Messages** menu—displays the result of searches in the trace data.



Trace	i
0080A4 CMP R0, #0	2
008028 CMP R1, R0	2
008030 CMP R3, R4	2
00803A CMP R5, #0	2
008028 CMP R1, R0	0
00816A CMP R4, #10	2

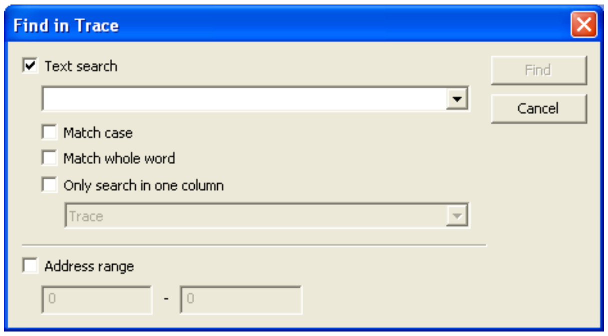
Figure 14: Find In Trace window

The Find in Trace window looks like the Trace window and shows the same columns and data, but *only* those rows that match the specified search criteria. Double-click an item in the Find in Trace window to bring up the same item in the Trace window.

You specify the search criteria in the **Find In Trace** dialog box. For information about how to open this dialog box, see *Find in Trace dialog box*, page 34.

FIND IN TRACE DIALOG BOX

Use the **Find in Trace** dialog box—available by choosing **Edit>Find and Replace>Find** or from the Trace window toolbar—to specify the search criteria for advanced searches in the trace data. Note that the **Edit>Find and Replace>Find** command is context-dependent. It displays the **Find in Trace** dialog box if the Trace window is the current window or the **Find** dialog box if the editor window is the current window.



☑ Text search

☐ Match case

☐ Match whole word

☐ Only search in one column

Trace

☐ Address range

0 - 0

Figure 15: Find in Trace dialog box

The search results are displayed in the Find In Trace window—available by choosing the **View>Messages** command, see *Find In Trace window*, page 34.

In the **Find in Trace** dialog box, you specify the search criteria with the following settings.

Text search

A text field where you type the string you want to search for. Use these options to fine-tune the search:

Match Case	Searches only for occurrences that exactly match the case of the specified text. Otherwise specifying `int` will also find `INT` and `Int`.
Match whole word	Searches only for the string when it occurs as a separate word. Otherwise `int` will also find `print`, `sprintf` and so on.
Only search in one column	Searches only in the column you selected from the drop-down list.

Address Range

Use the text fields to specify an address range. The trace data within the address range is displayed. If you also have specified a text string in the **Text search** field, the text string is searched for within the address range.

Using the profiler

This section gives you information about using the profiler. More specifically, these topics are covered:

- *Reasons for using the profiler*, page 36
- *How to use the profiler on function level*, page 37
- *How to use the profiler on instruction level*, page 37
- *Function Profiler window*, page 38

Profiling is only available for the E100 driver.

Note: This profiler is part of the C-SPY driver and should not be confused with the profiling system provided as a plug-in module, see the *IAR Embedded Workbench® IDE User Guide*.

REASONS FOR USING THE PROFILER

- *Function profiling* information is displayed in the Function Profiler window, that is, timing information for the functions in an application. The profiler measures the time between the entry and return of a function. This means that time consumed in a function is not added until the function returns or another function is called. You will only notice this if you are stepping into a function. Profiling must be turned on explicitly using a button on the window's toolbar, and will stay enabled until it is turned off.
- *Instruction profiling* information is displayed in the Disassembly window, that is, the number of times each instruction has been executed.

Function profiling can help you find the functions where most time is spent during execution. Focus on those functions when optimizing your code. A simple method of optimizing a function is to compile it using speed optimization. Alternatively, you can move the function into the memory which uses the most efficient addressing mode. For detailed information about efficient memory usage, see the *IAR C/C++ Compiler Reference Guide for M16C/R8C*.

Instruction profiling can help you fine-tune your code on a very detailed level, especially for assembler source code. Instruction profiling can also help you to understand where your compiled C/C++ source code spends most of its time, and perhaps give insight into how to rewrite it for better performance.

HOW TO USE THE PROFILER ON FUNCTION LEVEL

To display function profiling information in the Function Profiler window, follow these steps:

1 Make sure you build your application using these options:

Category	Setting
C/C++ Compiler	Output>Generate debug information
Linker	Output>Format>Debug information for C-SPY

Table 8: Project options for enabling the profiler

2 When you have built your application and started C-SPY, choose **E100 Emulator>Function Profiler** to open the Function Profiler window, and click the **Enable** button to turn on the profiler. Alternatively, choose **Enable** from the context menu that is available when you right-click in the Function Profiler window.

3 Start executing your application to collect the profiling information.

4 Profiling information is displayed in the Function Profiler window. To sort, click on the relevant column header.

5 When you start a new sampling, you can click the **Clear** button—alternatively, use the context menu—to clear the data.

HOW TO USE THE PROFILER ON INSTRUCTION LEVEL

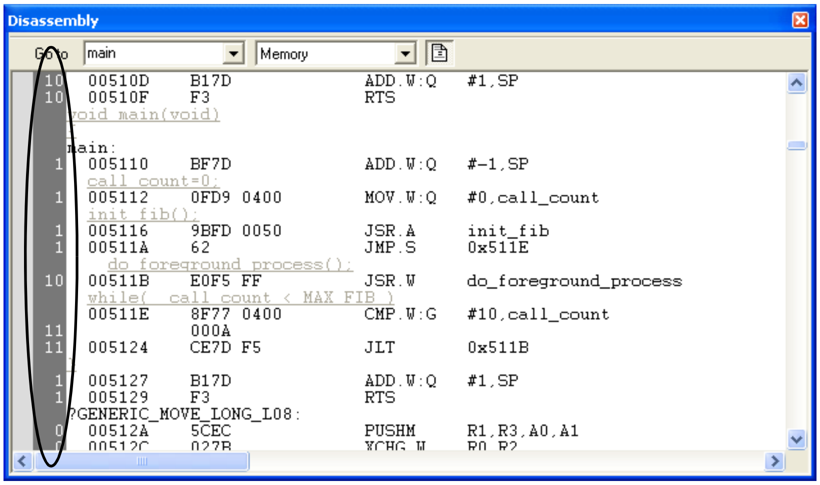
To display instruction profiling information in the Disassembly window, follow these steps:

1 When you have built your application and started C-SPY, choose **View>Disassembly** to open the Disassembly window, and choose **Enable** from the context menu that is available when you right-click in the Profiler window.

2 Make sure that the **Show** command on the context menu is selected, to display the profiling information.

3 Start executing your application to collect the profiling information.

4 When the execution stops, for instance because the program exit is reached or a breakpoint is triggered, you can view instruction level profiling information in the left-hand margin of the Disassembly window.



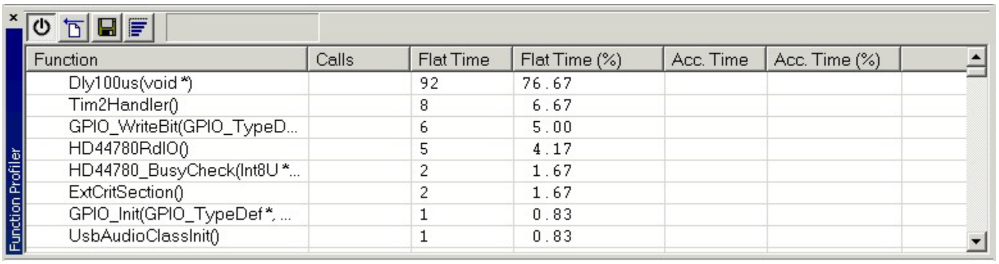
```
10  00510D    B17D          ADD.W:Q   #1,SP
10  00510F    F3            RTS
void main(void)

main:
 1  005110    BF7D          ADD.W:Q   #-1,SP
    call_count=0;
 1  005112    0FD9 0400     MOV.W:Q   #0,call_count
    init_fib();
 1  005116    9BFD 0050     JSR.A     init_fib
 1  00511A    62            JMP.S     0x511E
      do_foreground_process();
10  00511B    E0F5 FF       JSR.W     do_foreground_process
    while( call_count < MAX_FIB )
    00511E    8F77 0400     CMP.W:G   #10,call_count
11            000A
11  005124    CE7D F5       JLT       0x511B

 1  005127    B17D          ADD.W:Q   #1,SP
 1  005129    F3            RTS
?GENERIC_MOVE_LONG_L08:
 0  00512A    5CEC          PUSHM     R1,R3,A0,A1
 0  00512C    027B          XCHG.W    R0,R2
```

Figure 16: Instruction count in Disassembly window

For each instruction, the number of times it has been executed is displayed.

FUNCTION PROFILER WINDOW

The Function Profiler window—available from the **E100 Emulator** menu—displays function profiling information.



Function	Calls	Flat Time	Flat Time (%)	Acc. Time	Acc. Time (%)
Dly100us(void *)		92	76.67		
Tim2Handler()		8	6.67		
GPIO_WriteBit(GPIO_TypeD...		6	5.00		
HD44780RdIO()		5	4.17		
HD44780_BusyCheck(Int8U *...		2	1.67		
ExtCritSection()		2	1.67		
GPIO_Init(GPIO_TypeDef *, ...		1	0.83		
UsbAudioClassInit()		1	0.83		

Figure 17: Function Profiler window

Toolbar

The toolbar at the top of the window provides these buttons:

Toolbar button	Description
Enable/Disable	Enables or disables the profiler.
Clear	Clears all profiling data.
Save	Opens a standard **Save As** dialog box where you can save the contents of the window to a file, with tab-separated columns. Only non-expanded rows are included in the list file.
Graphical view	Overlays the values in the percentage columns with a graphical bar.
Progress bar	Displays a backlog of profiling data that is still being processed. If the rate of incoming data is higher than the rate of the profiler processing the data, a backlog is accumulated. The progress bar indicates that the profiler is still processing data, but also approximately how far the profiler has come in the process. Note that because the profiler consumes data at a certain rate and the target system supplies data at another rate, the amount of data remaining to be processed can both increase and decrease. The progress bar can grow and shrink accordingly.

Table 9: Function Profiler window toolbar

The display area

The content in the display area depends on which source is used for the profiling information; the display area contains one line for each function compiled with debug information. When some profiling information has been collected, it is possible to expand rows of functions that have called other functions. The child items for a given function list all the functions that have been called by the parent function and the corresponding statistics.

More specifically, the display area provides this information:

Column	Description
Function	The name of the profiled C function.
Calls	The number of times the function has been called.
Flat time	The time in estimated number of cycles spent inside the function.

Table 10: Function Profiler window columns

Column	Description
Flat time (%)	Flat time in estimated number of cycles expressed as a percentage of the total time.
Acc. time	The time in estimated number of cycles spent in this function and everything called by this function.
Acc. time (%)	Accumulated time in estimated number of cycles expressed as a percentage of the total time.

Table 10: Function Profiler window columns (Continued)

Function Profiler window context menu

This context menu is available in the Function Profiler window:

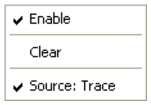


Figure 18: Function Profiler window context menu

These commands are available on the menu:

Menu command	Description
Enable	Enables the profiler. The system will collect information also when the window is closed.
Clear	Clears all profiling data.
Source	Selects which source to be used for the profiling information. The only supported source is **Trace**: **Trace**, supported by the J-Trace debug probe. The instruction count for instruction profiling is only as complete as the collected trace data.

Table 11: Commands on the Function Profiler window context menu

Using breakpoints

This section provides an overview of the available breakpoints for the C-SPY hardware debugger systems.

- *Breakpoints in the C-SPY hardware drivers*, page 41
- *Data breakpoints dialog box (E1/E20)*, page 42
- *Data breakpoint dialog box (E100)*, page 43
- *Hardware Code Breakpoint dialog box*, page 49
- *Software Code Breakpoint dialog box*, page 50

- *Address breakpoints dialog box*, page 51
- *Breakpoint Usage dialog box*, page 53.

For information about the various methods for setting breakpoints, the facilities for monitoring breakpoints, and the various breakpoint consumers, see the *IAR Embedded Workbench® IDE User Guide*.

Note: There are two system macros that can be used to automate the setting of data and address breakpoints, see *C-SPY system macros*, page 53.

BREAKPOINTS IN THE C-SPY HARDWARE DRIVERS

Using the C-SPY drivers for hardware debugger systems you can set various breakpoint types. The amount of breakpoints you can set depends on the number of *hardware breakpoints* available on the target system. If no hardware breakpoints are available, *software breakpoints* will be used.

This table summarizes the characteristics of breakpoints for the different target systems:

C-SPY hardware driver	Software code breakpoints	Hardware code and Log breakpoints	Trace breakpoints	Data breakpoints
E1/E20				
using 10 h/w breakpoints*	256	Up to 8†	—	2
E100				
using 16 h/w breakpoints*	4096	Up to 16	Up to 16	4

Table 12: Available breakpoints in C-SPY hardware drivers

*** The number of available hardware breakpoints depends on the target system you are using.**
† These 8 breakpoint resources are shared between three types of breakpoints.

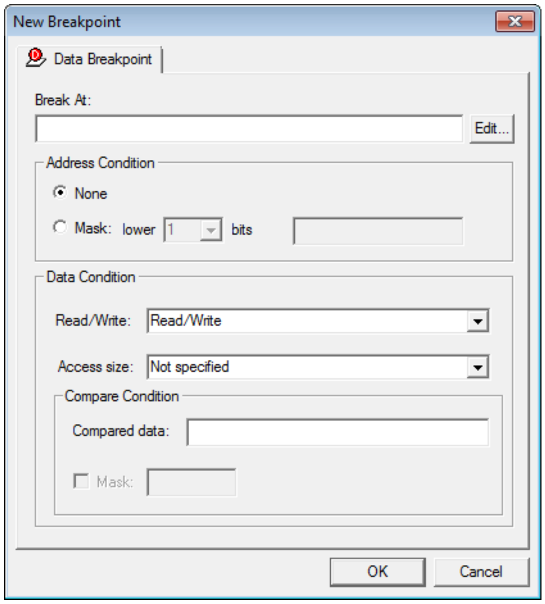
Hardware breakpoints in the emulators share the same resources.

The *software code* breakpoints use a mechanism that writes to the memory with a `BRK` instruction, and when a breakpoint is triggered the original instruction is written back to the memory. This makes it possible to use the breakpoint for code in RAM, but if used for code in flash memory the execution is slowed down by the need to reprogram the memory.

The debugger will first use any available hardware breakpoints before using software breakpoints. Exceeding the number of available breakpoints causes the debugger to single step. This will significantly reduce the execution speed. For this reason you must be aware of the different breakpoint consumers.

DATA BREAKPOINTS DIALOG BOX (E1/E20)

The **Data Breakpoint** dialog box is available from the context menu in the editor window, Breakpoints window, the Memory window, and in the Disassembly window.



Break At:

Address Condition

- None [✓]
- Mask: lower 1 bits

Data Condition

Read/Write: Read/Write

Access size: Not specified

Compare Condition

Compared data:

Mask:

Figure 19: Data breakpoints dialog box (E1/E20)

This dialog box is available for the C-SPY E1/E20 driver.

Use the **Data Breakpoint** dialog box to set a data breakpoint. Data breakpoints never stop execution within a single instruction. They are recorded and reported after the instruction is executed.

Break At

Specify the location of the breakpoint, or the start location if you select the address condition **Range**. Alternatively, click the **Edit** button to open the **Enter Location** dialog box; see the *IAR Embedded Workbench® IDE User Guide*.

Address Condition

Determines whether there should be a trigger condition for the address where the data is located:

None	There is no condition for the address.
Mask	Specify an address mask. The lower bits of the address are masked: choose how many bits to mask.

Data Condition

Determines whether there should be data trigger conditions for the breakpoint:

Read/Write	**Read** — the breakpoint is only triggered by a read access **Write** — the breakpoint is only triggered by a write access **Read/Write** — the breakpoint is triggered by all accesses.
Access size	Determines whether there should be a size—in practice, a range—of locations where the breakpoint will trigger. Each fetch access to the specified memory range will trigger the breakpoint. Choose between: **Not specified** — No access size has been specified. **Byte** — The access size will be set to a byte. **Word** — The access size will be set to a word. (If you use this size with a breakpoint on an absolute address, it must be an even address.)
Compared data	Set a value that the accessed data should be compared to, using decimal notation or hexadecimal notation (prefixed by `0x`).
Mask	Set a mask for the compared data, using decimal notation or hexadecimal notation (prefixed by `0x`).

DATA BREAKPOINT DIALOG BOX (E100)

The options for setting data breakpoints are available from the context menu that appears when you right-click in the **Breakpoints** window. On the context menu, choose **New Breakpoint>Data** to set a new breakpoint. Alternatively, to modify an existing breakpoint, select a breakpoint in the Breakpoint window and choose **Edit** on the context menu.

You can also set a data breakpoint on a memory range by right-clicking in the Memory window.

The **Data Breakpoint** dialog box appears.

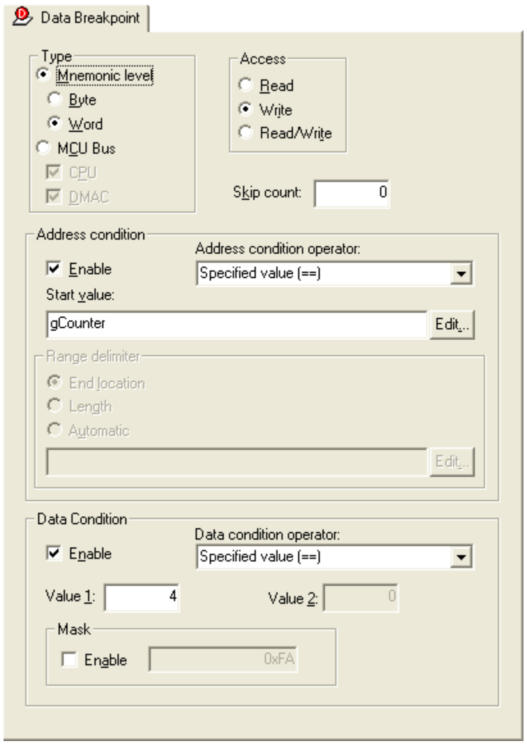


Figure 20: Data Breakpoint dialog box (E100)

This dialog box is available for the C-SPY E100 driver.

Data breakpoints are triggered when data is accessed at the specified location. Data breakpoints are primarily useful for variables that have a fixed address in memory. If you set a breakpoint on an accessible local variable, the breakpoint will be set on the corresponding memory location. The validity of this location is only guaranteed for small parts of the code. The execution will usually stop directly after the instruction that accessed the data has been executed.

Note: When a data breakpoint is triggered, the execution continues for a few cycles before it stops. The debug log shows which breakpoint that was triggered.

Type

To choose how the breakpoint detects the access, select one of the following options:

Type	Description
Mnemonic level	A detection method where data is specified according to the size handled by a mnemonic (`.B` or `.W`). Can only be used with 8- or 16-bit data types. Choose between: **Byte** – accesses the data in one byte **Word** – accesses the data in two bytes
MCU Bus	A detection method where data is specified according to the bus operation of the target microcontroller or MCU. This method must be used with 32- and 64-bit data types and cannot detect data values. Choose between: **CPU** – CPU bus cycle **DMAC** – DMAC bus cycle

Table 13: Access detection type

Access

Use the **Access** options to specify the type of memory access that triggers the breakpoint.

Memory Access type	Description
Read	Read from location.
Write	Write to location.
Read/Write	Read or write from location.

Table 14: Memory Access types

Skip count

The number of times that the breakpoint must be fulfilled before a break occurs.

Address condition

Select **Enable** to be able to specify an address condition for triggering the breakpoint.

Address condition operator

Select how the values you specify (a start location and possibly an end location) are interpreted:

- Specified value (==)
- Any other value (!=)

- Inside the range (<=Values<=)
- Outside the range !(<=Values<=)
- Greater than or equal to (>=)
- Less than or equal to (<=)
- Greater value (>)
- Less value (<)

For the settings **Inside the range (<=Values<=)** and **Outside the range !(<=Values<=)**, you must also specify the end of the address range using the **Range delimiter** option.

Start value

Set the (start) location for the breakpoint. Clicking the **Edit** button opens the **Enter Location** dialog box; see the *IAR Embedded Workbench® IDE User Guide*.

Range delimiter

If the breakpoint is set on an address range, use this option to specify the end point of the range. Clicking the **Edit** button opens the **Enter Location** dialog box; see the *IAR Embedded Workbench® IDE User Guide*.

Select one of:

End location	One of the location types used for the start location.
Length	The length of the range in hexadecimal notation.
Automatic	The range is automatically based on the expression the breakpoint is set on. For example, if you set the breakpoint on a 12-byte structure, the range of the breakpoint will be 12 bytes.

Data condition

Select **Enable** to be able to specify an additional data condition for triggering the breakpoint.

Data condition operator

Select how the values you specify (a start value and possibly an end value) are interpreted:

- Specified value (==)
- Any other value (!=)
- Inside the range (<=Values<=)

- Outside the range !(<=Values<=)
- Greater than or equal to (>=)
- Less than or equal to (<=)
- Greater value (>)
- Less value (<)

For the settings **Inside the range (<=Values<=)** and **Outside the range !(<=Values<=)**, you must also specify an end value using the **Value 2** field.

Value 1

The (start) data value of the condition.

Value 2

If the data condition operator specifies a value range, use this field to specify the end value.

Mask

Select **Enable** and specify the bit mask value that the breakpoint value will be masked with.

Examples

These examples illustrate how you use the **Data Breakpoint** dialog box.

Example 1

A data breakpoint for `MOV.W #0x1234, 0x503`:

Type	Mnemonic level: Word
Address condition operator	Specified value (==)
Start value	`0x503`
Data condition operator	Specified value (==)
Value 1	`0x1234`
Mask	`0xFFFF` (set by default when you select **Mnemonic level**.)

Example 2

A data breakpoint for `MOV.W #0x1234, 0x503`, using two breakpoints:

The first breakpoint:

Type	MCU Bus
Address condition operator	Specified value (==)
Start value	`0x503`
Data condition operator	Specified value (==)
Value 1	`0x3400`
Mask	`0xFF00`

The second breakpoint:

Type	MCU Bus
Address condition operator	Specified value (==)
Start value	`0x504`
Data condition operator	Specified value (==)
Value 1	`0x0012`
Mask	`0x00FF`

Example 3

A data breakpoint that triggers when data in a specific address range is modified, in this case `char arr1[5]` located at `0x480` to `0x484`:

Type	MCU Bus
Access	Write
Address condition operator	Inside the range (<= Values <=)
Start value	`0x480`
Range delimiter: End location	`0x484`

HARDWARE CODE BREAKPOINT DIALOG BOX

The **Hardware Code Breakpoint** dialog box is available from the context menu in the editor window, Breakpoints window, and in the Disassembly window.

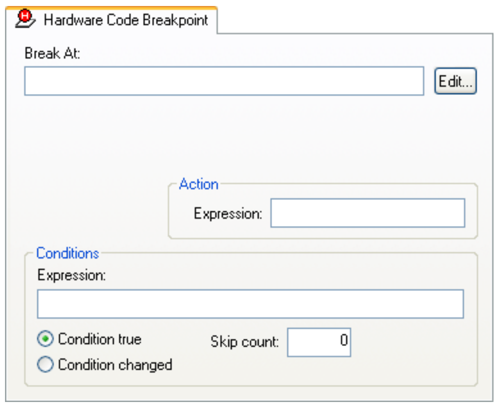


Figure 21: Hardware Code breakpoints dialog box

This dialog box is available for the C-SPY E1/E20 driver.

Use the **Hardware Code Breakpoint** dialog box to set a hardware breakpoint.

Note: When you use a C-SPY hardware debugger driver and set a breakpoint in code without specifying the type, a *hardware code* breakpoint will be set as long as there are any available. If there are no available hardware code breakpoints, a *software code* breakpoint will be set.

Break At

Specify the location of the breakpoint in the text box. Alternatively, click the **Edit** button to open the **Enter Location** dialog box; see the *IAR Embedded Workbench® IDE User Guide*.

Action

Determines whether there is an action connected to the breakpoint. Specify an expression, for instance a C-SPY macro function, which is evaluated when the breakpoint is triggered and the condition is true.

Conditions

Specify simple or complex conditions:

Expression	Specify a valid expression conforming to the C-SPY expression syntax.
Condition true	The breakpoint is triggered if the value of the expression is true.
Condition changed	The breakpoint is triggered if the value of the expression has changed since it was last evaluated.
Skip count	The number of times that the breakpoint condition must be fulfilled before the breakpoint starts triggering.

SOFTWARE CODE BREAKPOINT DIALOG BOX

The **Software Code Breakpoint** dialog box is available from the context menu in the editor window, Breakpoints window, and in the Disassembly window.

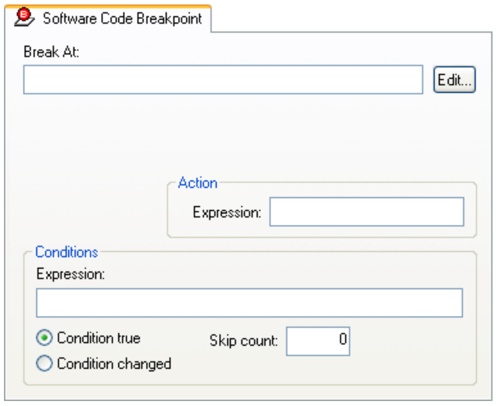


Figure 22: Software Code breakpoints dialog box

This dialog box is available for the C-SPY E1/E20 driver.

Use the **Software Code Breakpoint** dialog box to set a software code breakpoint.

Note: Because the memory must be reprogrammed after a software code breakpoint has been triggered, software code breakpoints should if possible be used only in RAM memory or for breakpoints that are not triggered too often.

Break At

Specify the location of the breakpoint in the text box. Alternatively, click the **Edit** button to open the **Enter Location** dialog box; see the *IAR Embedded Workbench® IDE User Guide*.

Action

Determines whether there is an action connected to the breakpoint. Specify an expression, for instance a C-SPY macro function, which is evaluated when the breakpoint is triggered and the condition is true.

Conditions

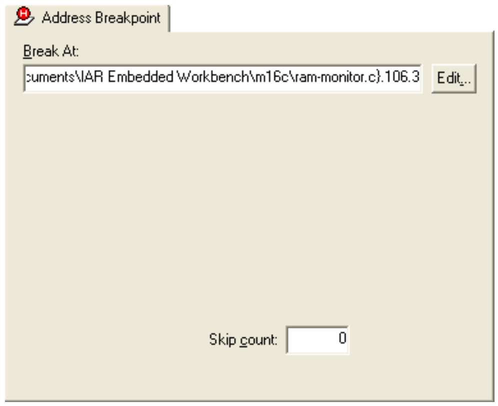
Specify simple or complex conditions:

Expression	Specify a valid expression conforming to the C-SPY expression syntax.
Condition true	The breakpoint is triggered if the value of the expression is true.
Condition changed	The breakpoint is triggered if the value of the expression has changed since it was last evaluated.
Skip count	The number of times that the breakpoint condition must be fulfilled before the breakpoint starts triggering.

ADDRESS BREAKPOINTS DIALOG BOX

The options for setting address breakpoints are available from the context menu that appears when you right-click in the **Breakpoints** window. On the context menu, choose **New Breakpoint>Address** to set a new breakpoint. Alternatively, to modify an existing breakpoint, select a breakpoint in the Breakpoint window and choose **Edit** on the context menu.

The **Address Breakpoint** dialog box appears.



Break At: cuments\IAR Embedded Workbench\m16c\ram-monitor.c}.106.3

Skip count: 0

Figure 23: Address Breakpoint dialog box

This dialog box is available for the C-SPY E100 driver.

Address breakpoints are triggered when an instruction is fetched from the specified location.

Note: When an address breakpoint is triggered, the execution continues for a few cycles before it stops. The debug log shows which breakpoint that was triggered.

Break At

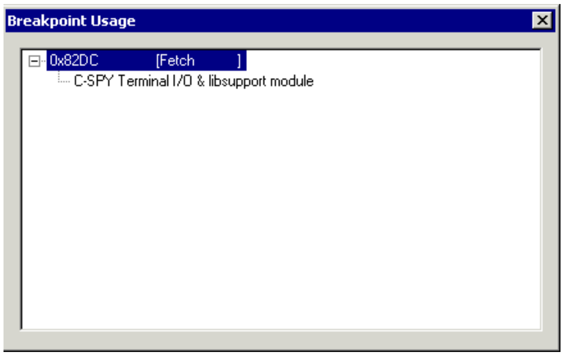
Specify the location for the breakpoint in the **Break At** text box, alternatively use the **Edit** button; see the *IAR Embedded Workbench® IDE User Guide*.

Skip count

The number of times that the breakpoint must be fulfilled before a break occurs.

BREAKPOINT USAGE DIALOG BOX

The **Breakpoint Usage** dialog box—available from the ***Driver*** menu—lists all active breakpoints.



0x82DC [Fetch]
C-SPY Terminal I/O & libsupport module

Figure 24: Breakpoint Usage dialog box

In addition to listing all breakpoints that you have defined, this dialog box also lists the internal breakpoints that the debugger is using.

For each breakpoint in the list the address and access type are shown. Each breakpoint in the list can also be expanded to show its originator.

For more information, see the *IAR Embedded Workbench® IDE User Guide*.

C-SPY system macros

C-SPY® includes a comprehensive macro system which allows you to automate the debugging process. See the *IAR Embedded Workbench® IDE User Guide* for more information about the C-SPY macro system.

__setDataBreakE100

Syntax

```
__setDataBreakE100(type, access, start_loc, end_loc, addr_op,
data_cond, data_val1, data_val2, data_op, mask_enable, mask,
skip)
```

Parameters

type	A string that describes how the breakpoint detects the access:

`"MnemByte"`	Mnemonic with byte size
`"MnemWord"`	Mnemonic with word size
`"McuCpu"`	MCU bus with CPU bus cycle
`"McuCpuDmac"`	MCU bus with CPU and DMAC bus cycle
`"McuDmac"`	MCU bus with DMAC

access — A string that describes the memory access type: `"Read"`, `"Write"`, `"ReadWrite"`

start_loc — The (start) location for the breakpoint; a string with a location description. This can be either:
A *source location* on the form `{`*filename*`}.`*line*`.`*col* (for example `{D:\\src\\prog.c}.12.9`), although this is not very useful for data breakpoints

An *absolute location* on the form *zone*`:`*hexaddress* or simply *hexaddress* (for example `RAM:0x480`)

An *expression* whose value designates a location (for example `my_global_variable`).

end_loc — The end location for the breakpoint; a string with a location description. By setting this to a larger value than *start_loc*, you can specify an address range for the address condition. See the description of *start_loc*.

addr_op — A string that determines how *start_loc* and *end_loc* are used:

`"=="`	The specified start location
`"!="`	Any other location than *start_loc*
`"In"`	Inside the range *start_loc*–*end_loc*
`"Out"`	Outside the range *start_loc*–*end_loc*
`">="`	Greater than or equal to the start location
`"<="`	Lesser than or equal to the start location
`">"`	A larger value than the start location
`"<"`	A smaller value than the start location

data_cond — A string that enables or disables using a data condition, either `"DataEnabled"` or `"DataDisabled"`.

data_val1 — The (start) value of the data condition (string)

data_val2 — The end value of the data condition (string). By setting this to a larger value than *data_val1*, you can specify a value range for the data condition.

data_op	A string that determines how *data_val1* and *data_val2* are used:
	`"=="` The specified value of *data_val1*
	`"!="` Any other value than *data_val1*
	`"In"` Inside the range between *data_val1* and *data_val2*
	`"Out"` Outside the range between *data_val1* and *data_val2*
	`">="` Greater than or equal to *data_val1*
	`"<="` Lesser than or equal to *data_val1*
	`">"` A larger value than *data_val1*
	`"<"` A smaller value than *data_val1*
mask_enable	A string that enables or disables using a mask, either `"MaskEnabled"` or `"MaskDisabled"`.
mask	The data mask value (string).
skip	The number of times that a breakpoint condition must be fulfilled before a break occurs (integer)

Return value

Result	Value
Successful	An unsigned integer uniquely identifying the breakpoint. This value must be used to clear the breakpoint.
Unsuccessful	0

Table 15: __setDataBreakE100 return values

Description

Sets a data breakpoint, that is, a breakpoint which is triggered directly after the processor has read or written data at the specified location.

Applicability

This system macro is only available for the C-SPY E100 Emulator Debugger.

Example

```
__var brk;
brk = __setDataBreakE100("MnemWord","Write","gCounter","","==",
                         "DataEnabled","4","0","==",
                         "MaskDisabled","0xFFFF",0);
...
__clearBreak(brk);
```

__setAddressBreakE100

Syntax `__setAddressBreakE100(location, skip)`

Parameters

location	A string with a location description. This can be either: A *source location* on the form `{filename}.line.col` (for example `{D:\\src\\prog.c}.12.9`), although this is not very useful for data breakpoints An *absolute location* on the form `zone:hexaddress` or simply `hexaddress` (for example `RAM:0x480`) An *expression* whose value designates a location (for example `my_global_variable`).
skip	The number of times that a breakpoint condition must be fulfilled before a break occurs (integer)

Return value

Result	Value
Successful	An unsigned integer uniquely identifying the breakpoint. This value must be used to clear the breakpoint.
Unsuccessful	0

Table 16: __setAddressBreakE100 return values

Description Sets an address breakpoint, that is, a breakpoint which is triggered directly after the processor has fetched an instruction from the specified location.

Applicability This system macro is only available for the C-SPY E100 Emulator Debugger.

Example

```
__var brk;
brk = __setAddressBreakE100("MyFunc", 0);
...
__clearBreak(brk);
```

RAM-monitor

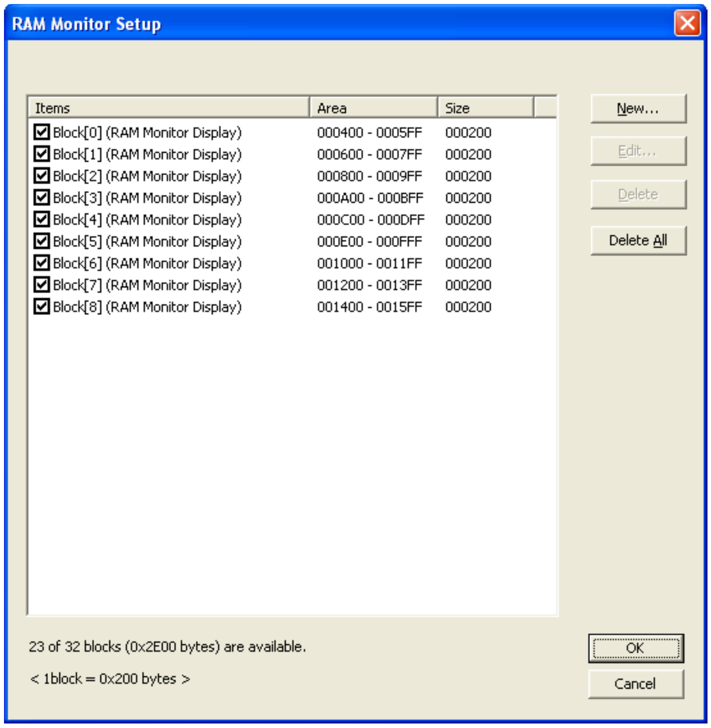
To display the values of variables in the Live Watch window or to use the live update feature in the Memory window, the E100 Emulator has a RAM-monitor function.

Using the RAM-monitoring, you can record and inspect the memory contents in an allocated monitor area in real time without obstructing the execution of your application.

The RAM-monitor memory can be allocated in up to 32 blocks of up to 512 bytes each.

RAM MONITOR SETUP DIALOG BOX

In the **RAM Monitor Setup** dialog box—available from the **E100 Emulator** menu—you can allocate memory blocks for live display in the Memory and Live Watch windows.



Items	Area	Size
☑ Block[0] (RAM Monitor Display)	000400 - 0005FF	000200
☑ Block[1] (RAM Monitor Display)	000600 - 0007FF	000200
☑ Block[2] (RAM Monitor Display)	000800 - 0009FF	000200
☑ Block[3] (RAM Monitor Display)	000A00 - 000BFF	000200
☑ Block[4] (RAM Monitor Display)	000C00 - 000DFF	000200
☑ Block[5] (RAM Monitor Display)	000E00 - 000FFF	000200
☑ Block[6] (RAM Monitor Display)	001000 - 0011FF	000200
☑ Block[7] (RAM Monitor Display)	001200 - 0013FF	000200
☑ Block[8] (RAM Monitor Display)	001400 - 0015FF	000200

23 of 32 blocks (0x2E00 bytes) are available.

< 1block = 0x200 bytes >

Figure 25: RAM Monitor Setup dialog box

Display area

The display area displays information in these columns:

Column	Description
Checkbox	Selected blocks will be used for live display in the Memory and Live Watch windows. Deselect a block to disable live update for it.

Table 17: RAM Monitor Setup dialog box display area

Column	Description
Items	Displays the ID of the address block.
Area	Displays the location in memory of the memory block.
Size	Displays the size of the memory block, in hexadecimal notation.

Table 17: RAM Monitor Setup dialog box display area

Buttons

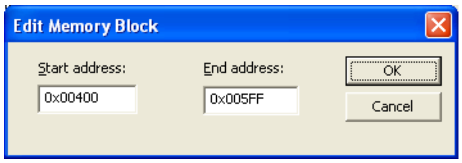
These buttons are available in the **RAM Monitor Setup** dialog box:

Button	Description
New	Displays the **Edit Memory Block** dialog box, where you can allocate a new memory block.
Edit	Displays the **Edit Memory Block** dialog box, where you can modify an already allocated memory block.
Delete	Deallocates the selected memory block.
Delete All	Deallocates all memory blocks.

Table 18: RAM Monitor Setup dialog box buttons

EDIT MEMORY BLOCK DIALOG BOX

In the **Edit Memory Block** dialog box—available from the **RAM Monitor Setup** dialog box—you can define new memory blocks to be recorded and inspected in real time, and edit already existing memory blocks.



Start address: 0x00400
End address: 0x005FF

Figure 26: Edit Memory Block dialog box

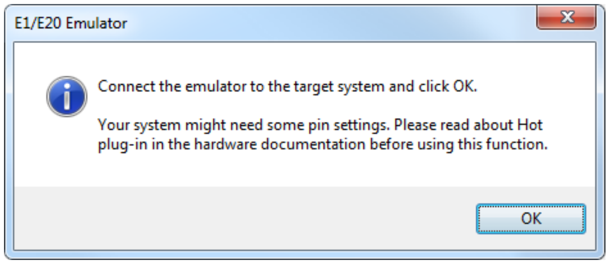
Use the **Start address** and **End address** text boxes to define the memory range for the blocks you are allocating. If the range is lesser than 512 (`0x200`) bytes, one block will be allocated. If the range is greater than 512 (`0x200`) bytes, more than one block will be allocated (if possible), to cover the entire range you are defining, up to a total maximum of 16 Kbytes (`0x4000`).

Start debugging a running application

On some devices, you can use an E1/E20 emulator to start debugging a running application at its current location, without resetting the target system.

START DEBUGGING FROM THE MIDDLE OF EXECUTION

1. Choose **Project>Options>E1/E20 Emulator>Download** and select the option **Attach to program**, see *Attach to program*, page 19.
2. Make sure that the target board is powered by external power and disconnect the emulator from the target board.
3. Choose **Project>Download and Debug** or click the **Download and Debug** toolbar button.
4. When you are prompted, connect the emulator to the target board and click **OK**.

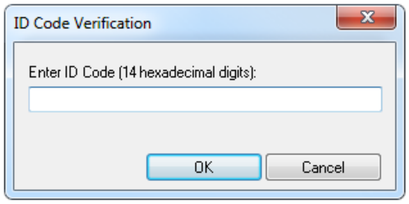


Connect the emulator to the target system and click OK.

Your system might need some pin settings. Please read about Hot plug-in in the hardware documentation before using this function.

Figure 27: Connect the emulator alert

5. Enter the ID code of the target MCU in the **ID Code Verification** dialog box.



Enter ID Code (14 hexadecimal digits):

Figure 28: ID Code Verification dialog box

6. When the debug session starts, your application is still executing but now you can monitor RAM and look at variables in the Live Watch window.
7. To stop execution, click **Stop** or set a breakpoint.
8. You can now debug your application as usual.

Resolving problems

Debugging using the C-SPY hardware debugger systems requires interaction between many systems, independent from each other. For this reason, setting up this debug system can be a complex task. If something goes wrong, it might at first be difficult to locate the cause of the problem.

This section includes suggestions for resolving the most common problems that can occur when debugging with the C-SPY hardware debugger systems.

For problems concerning the operation of the evaluation board, refer to the documentation supplied with it, or contact your hardware distributor.

WRITE FAILURE DURING LOAD

There are several possible reasons for write failure during load. The most common is that your application has been incorrectly linked:

- Check the contents of your linker configuration file and make sure that your application has not been linked to the wrong address
- Check that you are using correct linker configuration file.

If you are using the IAR Embedded Workbench, the linker configuration file is automatically selected based on your choice of device:

- Choose **Project>Options**
- Select the **General Options** category
- Click the **Target** tab
- Choose the appropriate device from the **Device** drop-down list.

To override the default linker configuration file:

- Choose **Project>Options**
- Select the **Linker** category
- Click the **Config** tab
- Choose the appropriate linker command file in the **Linker command file** area.

NO CONTACT WITH THE TARGET HARDWARE

There are several possible reasons for C-SPY to fail to establish contact with the target hardware.

- Check the communication devices on your host computer
- Verify that the cable is properly plugged in and not damaged or of the wrong type
- Make sure that the emulator is supplied with sufficient power

- Check that the correct options for communication have been specified in the IAR Embedded Workbench IDE; see *Communication*, page 17 and *Hardware Setup dialog box (E1/E20)*, page 21

Examine the linker command file to make sure that your application has not been linked to the wrong address.