Freescale Semiconductor
Application Note

Document Number: AN3830
Revision: 0
02/2009

Hardware Debugging Using the CodeWarrior™ IDE

By Freescale FAE Team

This application note provides a practical guide to using Freescale's CodeWarrior IDE to debug hardware. Focusing on PowerQUICC processors, this document covers many of the key features available in the IDE to assist in bring-up and troubleshooting of a new board.

Some of the topics covered are:

- Setting up a CodeWarrior project
- Hardware connection options
- Understanding the `.cfg` file
- Displaying memory and registers
- Using the Command window
- Capturing data

This guide is not a substitute for the extensive documentation available in your CodeWarrior installation or from the help menu. Rather, this document is meant to be a practical user's guide, describing the main features needed when debugging hardware and highlighting common errors. Also note that this document refers to CodeWarrior for Power Architecture Processors 8.x in its descriptions.

CONTENTS

Contents

© Freescale Semiconductor, 2008-2009. All rights reserved.

1 Getting Started: Creating a Project

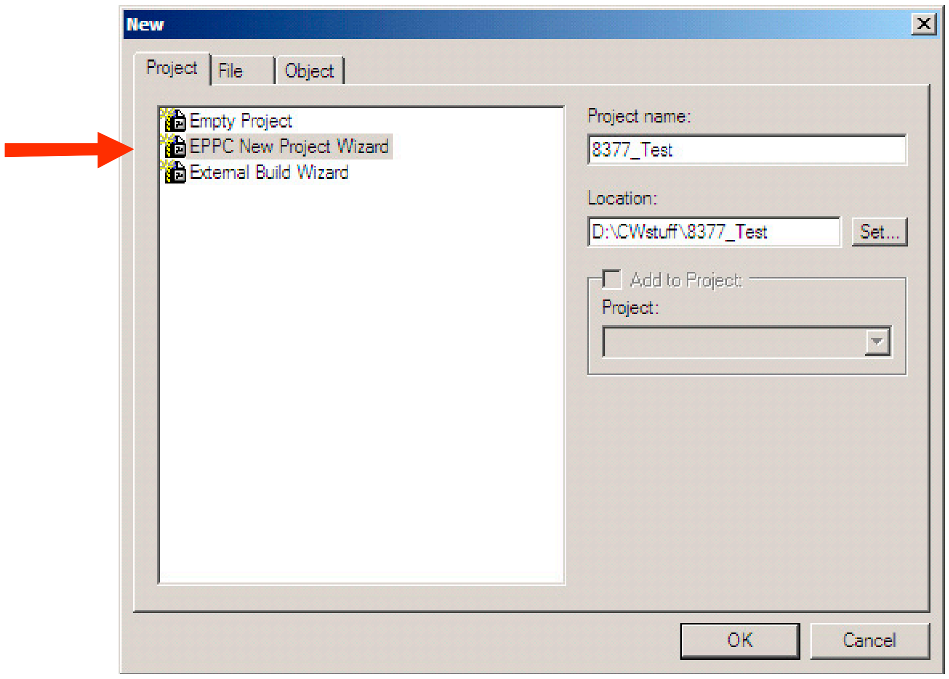
Before you can debug a board, you must provide the CodeWarrior IDE with information about the target processor, debug environment, and programming environment. The IDE stores this information in a CodeWarrior project. Before performing most functions with the IDE, you must create a project specific to the board you are debugging.

1.1 Using the EPPC New Project Wizard

Use the steps below to create a simple CodeWarrior project.

First, select **File > New** from the IDE's menu bar to display the **New** dialog box. (See Figure 1.)

Figure 1. The New Dialog Box

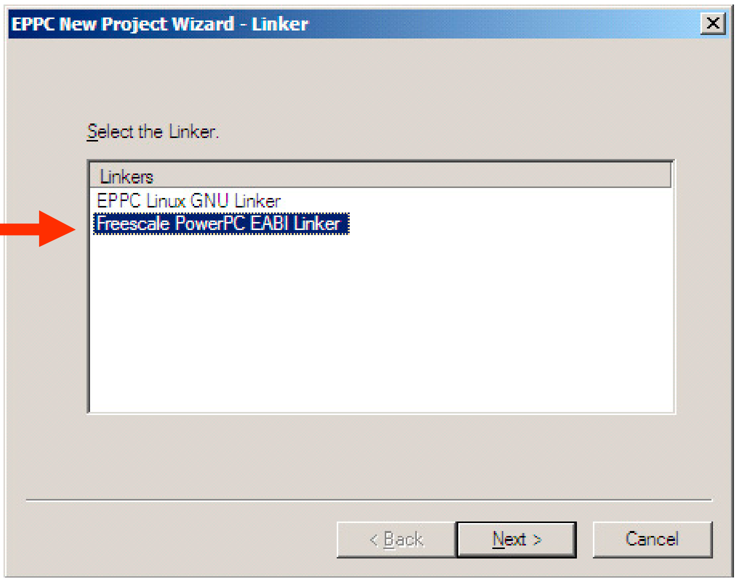


Project name: 8377_Test

Location: D:\CWstuff\8377_Test

In this dialog box, select EPPC New Project Wizard, enter a name and file location for your project, and click **OK**. The EPPC New Project Wizard starts and displays its first page. (See Figure 2.)

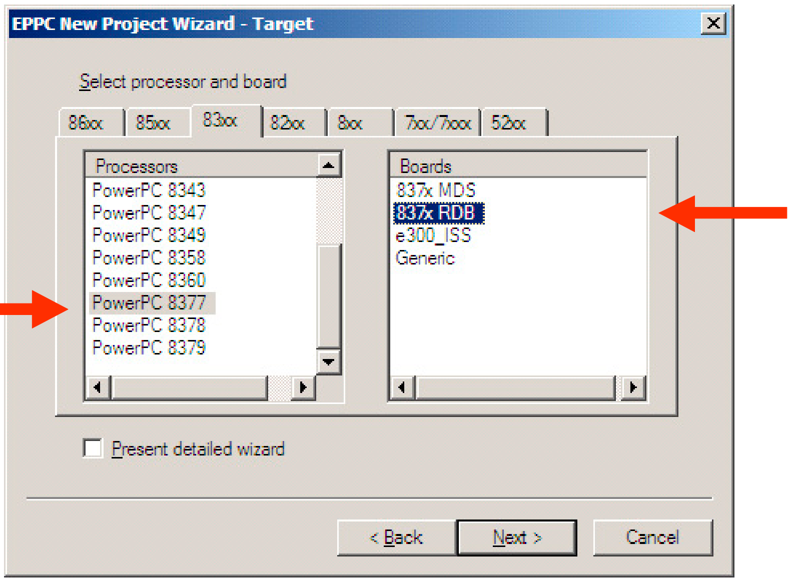
Figure 2. New Project Wizard – Linker Page



On this page, you must select the **Freescale PowerPC EABI Linker** to perform hardware debugging. you select the EPPC Linux GNU Linker instead, the next wizard page does not present the option to select the Power Architecture device you are using. This is a common mistake because the wizard's default linker selection is the EPPC Linux GNU Linker. The EPPC Linux GNU Linker is only used to generate Linux applications and kernel loadable modules.

The next wizard page (Figure 3) lets you select the processor being debugged. In this example, we selected the MPC8377 processor. Under the column for boards, you can either select one of the Freescale development boards listed or select **Generic** if you are attaching to a custom board, such as a new customer design. If you select a Freescale development board, the new project automatically points to the appropriate target initialization file (`.cfg` file). **Section 2.1** discusses the `.cfg` file.

Figure 3. New Project Wizard – Target Page



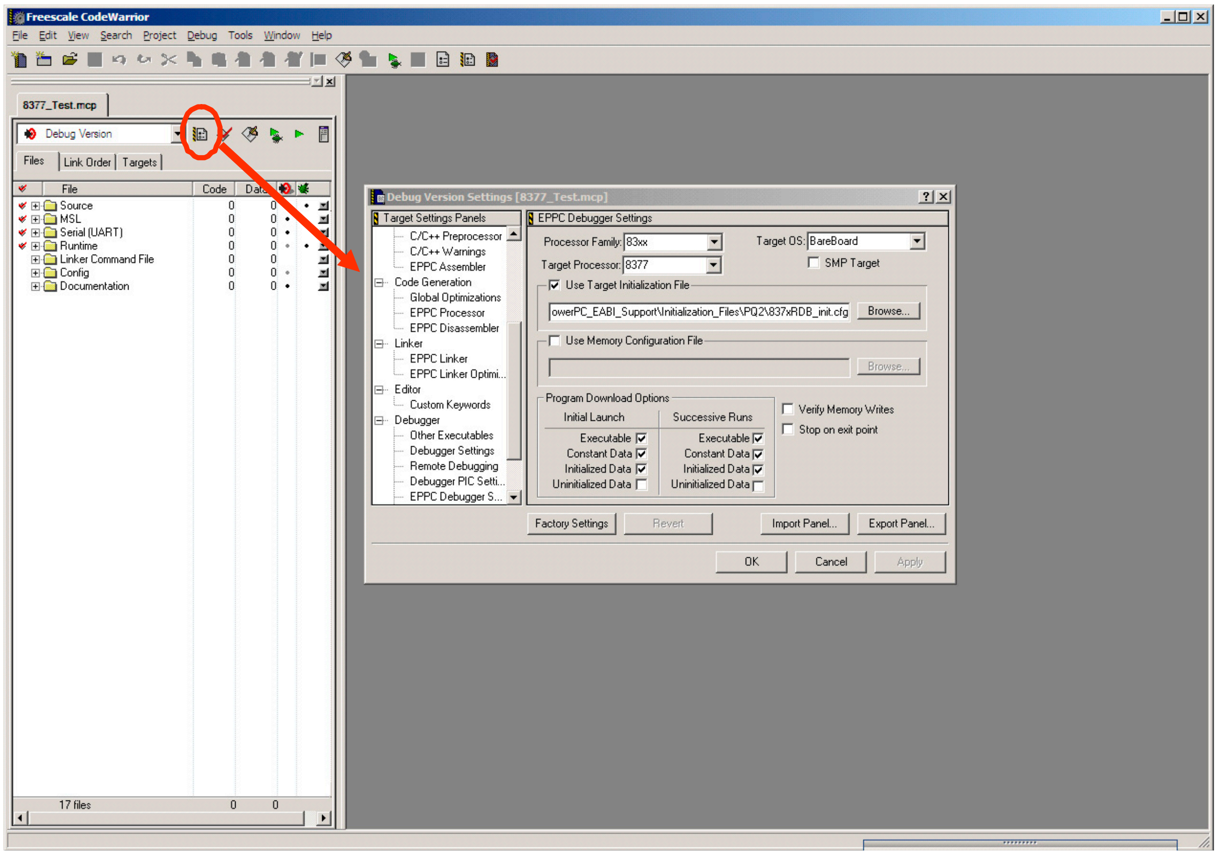
The last two wizard pages let you select the C or C++ programming language and the debug probe you will use to attach to your board. In this example, we use the USB TAP debug probe, which is provided with many of the PowerQUICC development boards.

At this point, you have created your project and are ready to start debugging.

1.2 Project Settings

Your project should now be displayed on the left side of the CodeWarrior IDE's main window. The actual project information is stored in an `.mcp` file in the location specified in the New Project Wizard. You can see detailed project settings by selecting the leftmost icon (see below) in the project window. Subsequent sections of this guide discuss key settings required for hardware debugging.

Figure 4. The Project Window and the Target Settings Window



EPPC Debugger Settings

Processor Family: 83xx
Target OS: BareBoard
Target Processor: 8377
☐ SMP Target
☑ Use Target Initialization File
owerPC_EABI_Support\Initialization_Files\PQ2\837xRDB_init.cfg Browse...
☐ Use Memory Configuration File
Browse...

Program Download Options

Initial Launch		Successive Runs	
Executable	☑	Executable	☑
Constant Data	☑	Constant Data	☑
Initialized Data	☑	Initialized Data	☑
Uninitialized Data	☐	Uninitialized Data	☐

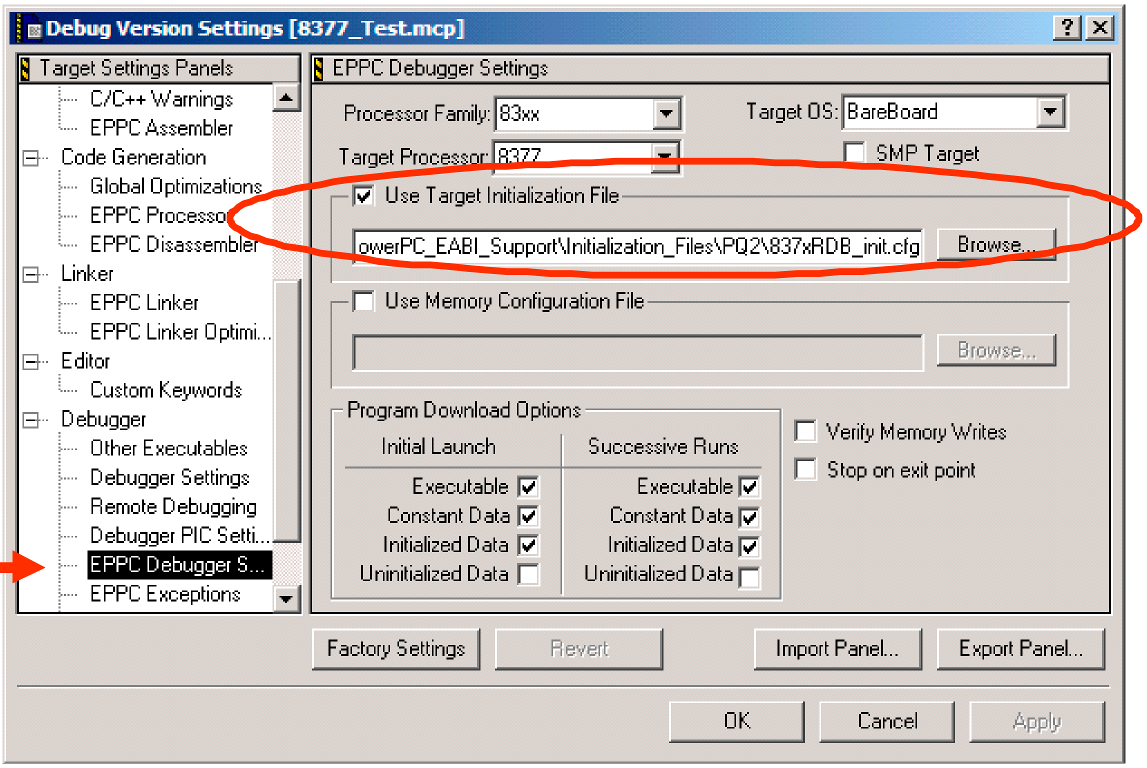
☐ Verify Memory Writes
☐ Stop on exit point

2 Using Target Initialization Files

2.1 Understanding the Target Initialization File (.cfg) File

One important setting in your project's configuration is the target initialization file, also known as the `.cfg` file. When connecting to a board via a USB TAP or similar JTAG-based COP interface, you have the option to download an initialization file containing commands that perform basic target device configuration. To use a `.cfg` file, you make settings in the EPPC Debugger Settings target settings panel. To display this panel, select EPPC Debugger Settings from the list of panels on the left side of the Target Settings window. (See Figure 5.)

Figure 5. The EPPC Debugger Settings Panel of the Target Settings Window



Processor Family: 83xx
Target OS: BareBoard
Target Processor: 8377
SMP Target: [ ]
Use Target Initialization File: [✓]
owerPC_EABI_Support\Initialization_Files\PQ2\837xRDB_init.cfg
Use Memory Configuration File: [ ]

Program Download Options

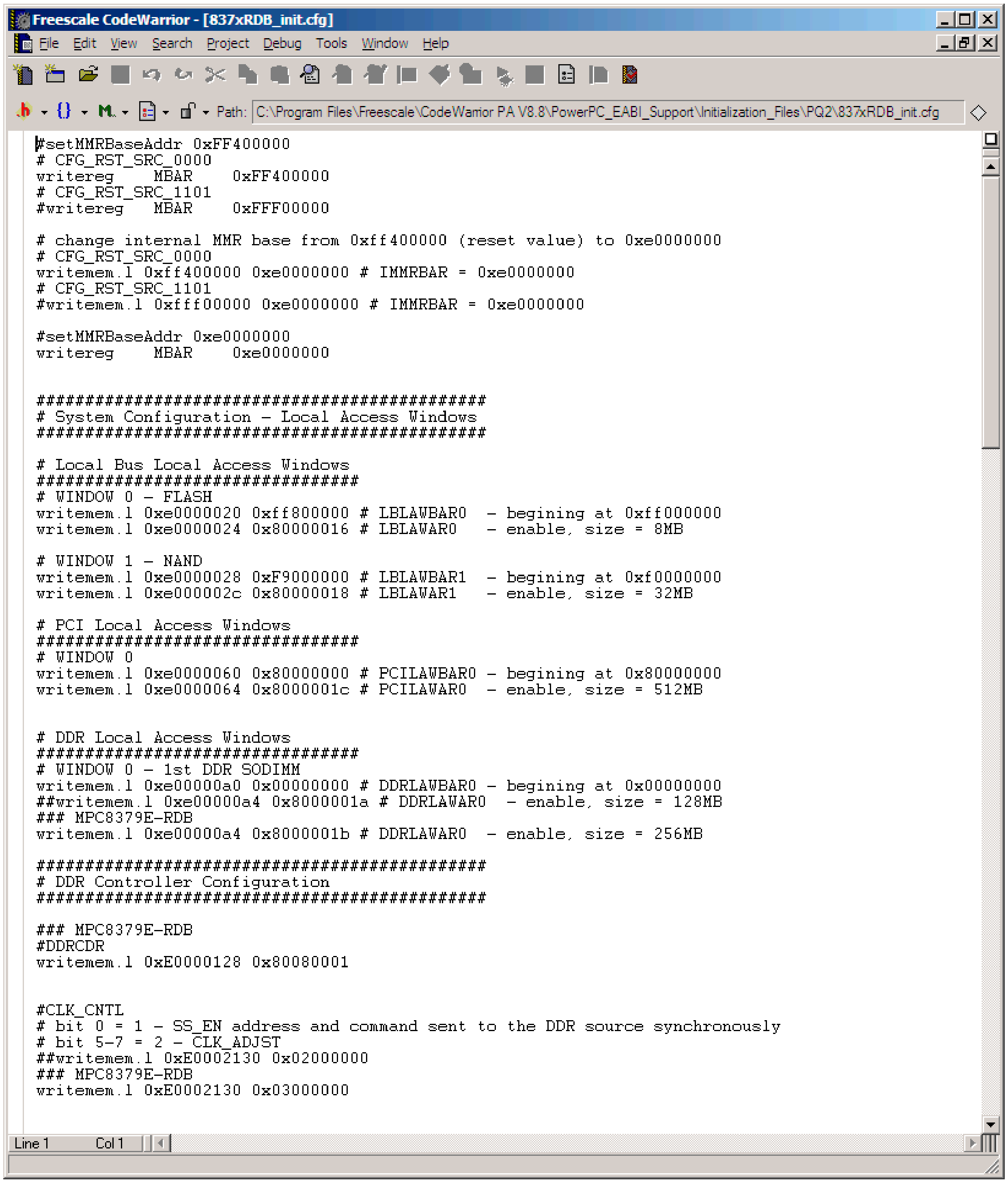
Initial Launch		Successive Runs	
Executable	✓	Executable	✓
Constant Data	✓	Constant Data	✓
Initialized Data	✓	Initialized Data	✓
Uninitialized Data		Uninitialized Data	

Verify Memory Writes: [ ]
Stop on exit point: [ ]

In the EPPC Debugger Settings panel, you can select whether to use a target initialization file, and also define the particular `.cfg` file to use. If you selected one of the Freescale development boards when you created your project, as we did with the 8377RDB, the correct `.cfg` file is automatically selected. Your CodeWarrior product includes a variety of `.cfg` files created for Freescale development and evaluation boards.

You can also modify an existing `.cfg` file or create one of you own that is customized for your board's settings. The `.cfg` file is a plain text file, which can set up basic device interfaces like the base address of the memory map, local access windows, and the DDR and local bus memory controller settings. Figure 6 shows a portion of the `.cfg` file for an 8377RDB board.

Figure 6. Example Target Initialization (.cfg) File for the 8377RDB Board


Freescale CodeWarrior - [837xRDB_init.cfg]
File Edit View Search Project Debug Tools Window Help
Path: C:\Program Files\Freescale\CodeWarrior PA V8.8\PowerPC_EABI_Support\Initialization_Files\PQ2\837xRDB_init.cfg
Line 1 Col 1

```
#setMMRBaseAddr 0xFF400000
# CFG_RST_SRC_0000
writereg    MBAR    0xFF400000
# CFG_RST_SRC_1101
#writereg   MBAR    0xFFF00000

# change internal MMR base from 0xff400000 (reset value) to 0xe0000000
# CFG_RST_SRC_0000
writemem.l 0xff400000 0xe0000000 # IMMRBAR = 0xe0000000
# CFG_RST_SRC_1101
#writemem.l 0xfff00000 0xe0000000 # IMMRBAR = 0xe0000000

#setMMRBaseAddr 0xe0000000
writereg    MBAR    0xe0000000


##############################################
# System Configuration - Local Access Windows
##############################################

# Local Bus Local Access Windows
#################################
# WINDOW 0 - FLASH
writemem.l 0xe0000020 0xff800000 # LBLAWBAR0  - begining at 0xff000000
writemem.l 0xe0000024 0x80000016 # LBLAWAR0   - enable, size = 8MB

# WINDOW 1 - NAND
writemem.l 0xe0000028 0xF9000000 # LBLAWBAR1  - begining at 0xf0000000
writemem.l 0xe000002c 0x80000018 # LBLAWAR1   - enable, size = 32MB

# PCI Local Access Windows
#################################
# WINDOW 0
writemem.l 0xe0000060 0x80000000 # PCILAWBAR0 - begining at 0x80000000
writemem.l 0xe0000064 0x8000001c # PCILAWAR0  - enable, size = 512MB


# DDR Local Access Windows
#################################
# WINDOW 0 - 1st DDR SODIMM
writemem.l 0xe00000a0 0x00000000 # DDRLAWBAR0 - begining at 0x00000000
##writemem.l 0xe00000a4 0x8000001a # DDRLAWAR0  - enable, size = 128MB
### MPC8379E-RDB
writemem.l 0xe00000a4 0x8000001b # DDRLAWAR0  - enable, size = 256MB

##############################################
# DDR Controller Configuration
##############################################

### MPC8379E-RDB
#DDRCDR
writemem.l 0xE0000128 0x80080001


#CLK_CNTL
# bit 0 = 1 - SS_EN address and command sent to the DDR source synchronously
# bit 5-7 = 2 - CLK_ADJST
##writemem.l 0xE0002130 0x02000000
### MPC8379E-RDB
writemem.l 0xE0002130 0x03000000
```

Important: If you are connecting to a board that is already running and do not want to overwrite its configuration and setup, uncheck the "Use Target Initialization File" option. Otherwise, the existing board's device configuration will be overwritten by the commands in the `.cfg` file.

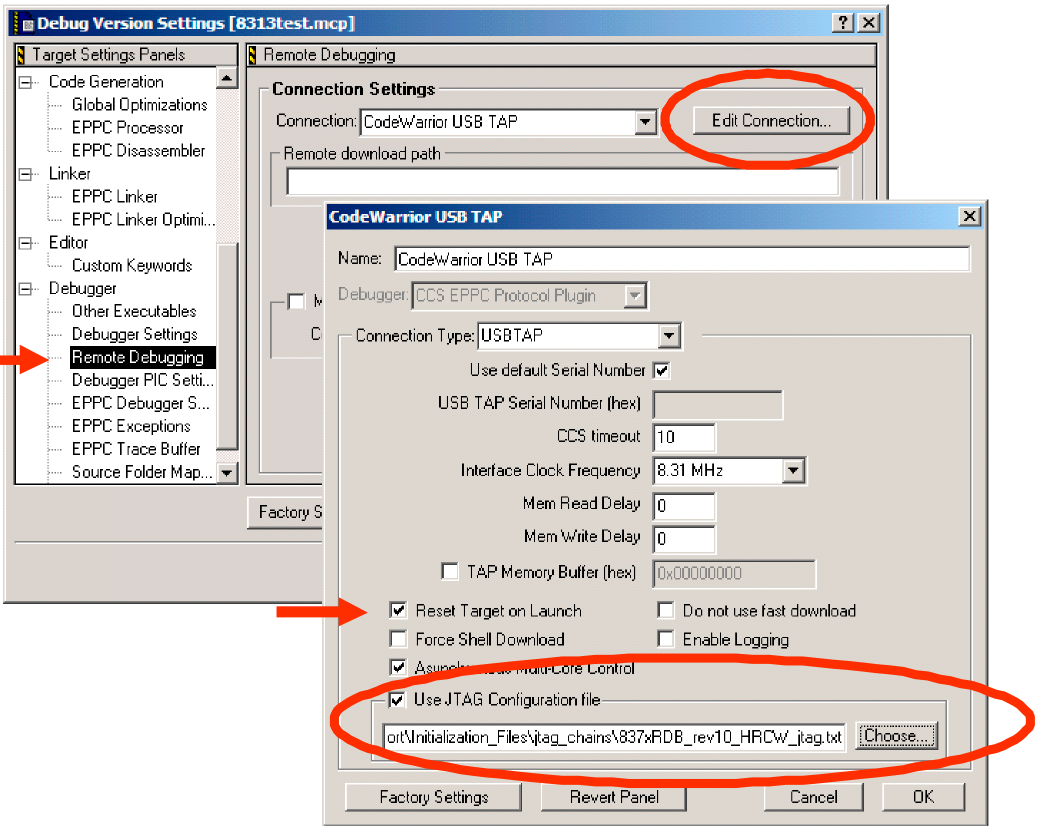
The `.cfg` file is useful in the initial stages of debugging a new board. This file provides an easy way to set up the memory controllers and ensure that accesses to DDR and flash memory are working properly. However, if you are debugging a problem with a board that is running software, you should probably disable the `.cfg` file, so you can view the issue using the actual software settings made by the board's BSP.

2.2 Configuring the Reset Configuration Word for the PQ2Pro

The PowerQUICC II Pro (MPC83xx) device reads a reset configuration word to initialize many important device settings. In many cases, this reset configuration word is stored in flash or some other non-volatile memory on the customer's board. However, you can also use your CodeWarrior tools to provide the reset configuration word information over the JTAG interface via a USB TAP. This feature is enabled via a JTAG initialization file and is especially useful for initial hardware debugging.

To select a JTAG initialization file, first display the Remote Debugging panel of the Target Settings window. (See Figure 7.) Next, click the Edit Connection button to display the CodeWarrior USB TAP dialog box. You must click "OK" when asked if you would like to continue in order to open this dialog box. As shown below, you can check the box next to "Use JTAG Configuration file", and then select the appropriate configuration file. Freescale provides JTAG configuration files for most of its PQ2 and PQ2Pro development boards as part of the CodeWarrior installation. You can also edit this text file to change the reset configuration word settings if necessary.

Figure 7. Remote Debugging Panel and CodeWarrior USB TAP Dialog Box



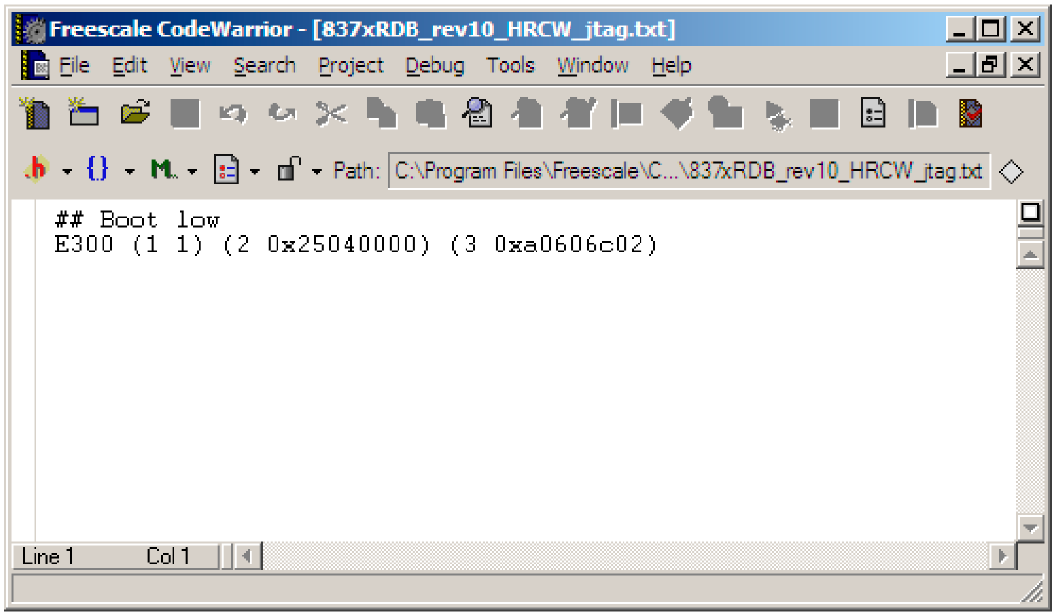
Connection: CodeWarrior USB TAP

Name: CodeWarrior USB TAP
Debugger: CCS EPPC Protocol Plugin
Connection Type: USBTAP
Use default Serial Number ☑
USB TAP Serial Number (hex)
CCS timeout 10
Interface Clock Frequency 8.31 MHz
Mem Read Delay 0
Mem Write Delay 0
☐ TAP Memory Buffer (hex) 0x00000000
☑ Reset Target on Launch ☐ Do not use fast download
☐ Force Shell Download ☐ Enable Logging
☑ Asynchronous Multi-Core Control
☑ Use JTAG Configuration file
ort\Initialization_Files\jtag_chains\837xRDB_rev10_HRCW_jtag.txt

Note that you should also make sure that the box beside “Reset Target on Launch” is checked, so that the device is reset when you connect to the board. This is necessary to ensure that the reset configuration settings from the specified JTAG configuration file are applied.

You can modify the reset configuration word file as desired to test different reset configuration options. The file is a text file (Figure 8) with the values for the high and low reset configuration word included in parenthesis.

Figure 8. Example JTAG Configuration File


Freescale CodeWarrior - [837xRDB_rev10_HRCW_jtag.txt]
File Edit View Search Project Debug Tools Window Help
Path: C:\Program Files\Freescale\C...\837xRDB_rev10_HRCW_jtag.txt
Line 1
Col 1

```
## Boot low
E300 (1 1) (2 0x25040000) (3 0xa0606c02)
```

3 CodeWarrior Board Connection Options

The CodeWarrior IDE offers multiple options for connecting to a board via a JTAG debug probe. Choosing the correct option is critical because you can easily overwrite settings on the board that you wish to maintain. Three of the most common connection options are covered below, along with the main actions the debugger takes when started under each option.

3.1 Project > Debug

To use this option, either select “Debug” from the IDE’s “Project” menu or click the green triangle with a bug icon under it in the project window. This option is the most intrusive to the system and causes the debugger to perform these tasks:

- Reset the target
- Load ELF / DWARF symbols
- Load the target initialization file (`.cfg`)
- Download project code
- Halt the target

If you want to attach to a board that is already running software in order to debug and observe, this is not the best connection option. You can disable the “Reset Target on Launch” option and/or the Target Initialization File option by modifying your projects debugger settings; however, even with both these options disabled, the debugger still downloads program code to the board.

3.2 Debug > Connect

To use this option. select “Connect” from the IDE’s “Debug” menu. This option is similar to the option discussed above, except that project code is *not* downloaded to the board. This option causes the debugger to perform these tasks:

- Reset the target
- Load the target initialization file (`.cfg`)
- Halt the target

Once again, you can disable “Reset Target on Launch” and/or the use of a target initialization file by modifying your project’s debugger settings.

3.3 Debug > Attach to Process

To use this option, select “Attach to Process” from the IDE’s “Debug” menu. Use this option if you want to attach to a running board with minimal interruption. This option causes the debugger to perform these tasks:

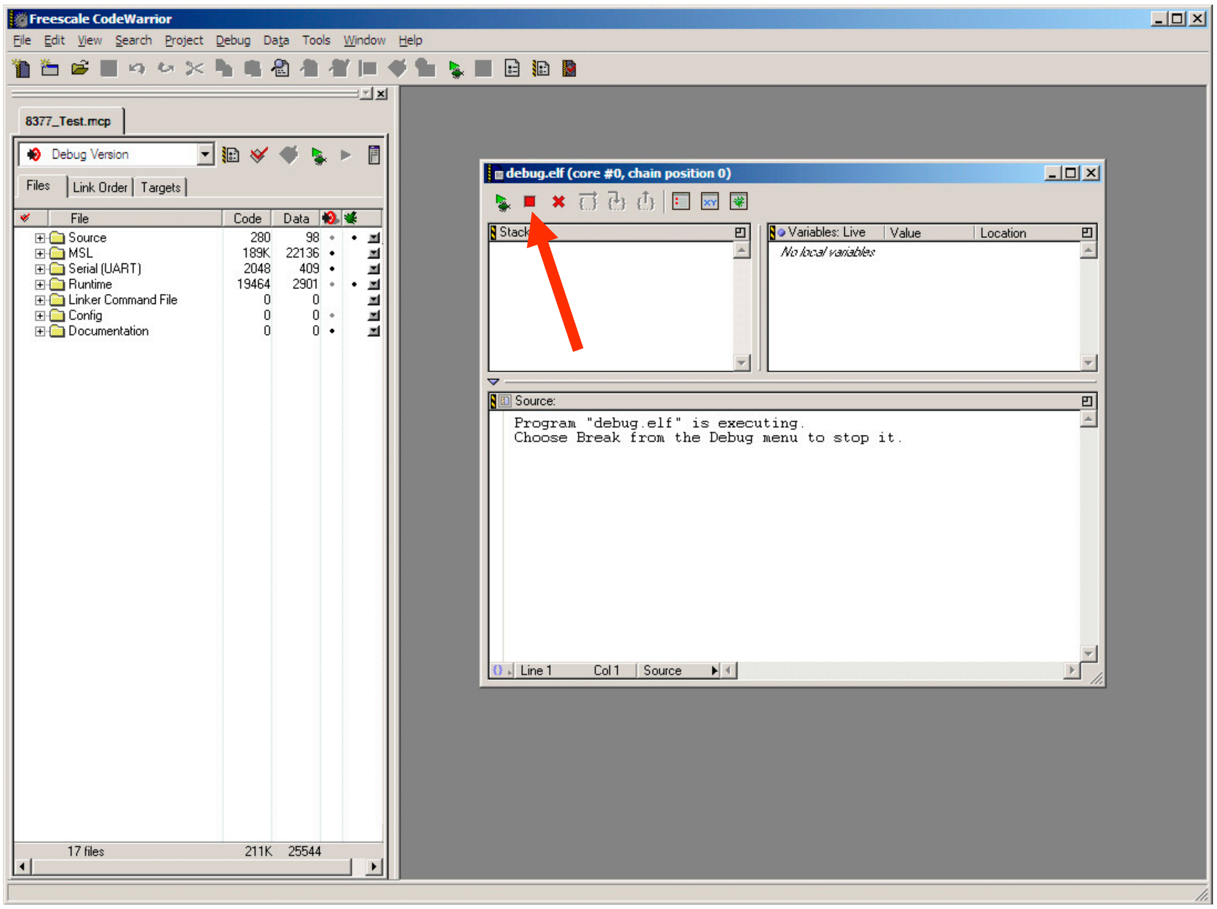
- Load ELF / DWARF symbols
- Leave the target running

Note that once the debugger is attached, you must halt the target before the debugger can access memory locations or registers.

4 Displaying Memory and Registers

Once you are connected to the target, most debugging involves displaying internal register settings or memory locations in the memory map of the processor. Before the debugger can display memory, the processor must be halted. If the processor is running, click the red box in the Debugger window (Figure 9) to break and halt execution. Do not click the red “X” because this button kills the thread and disconnects the debugger from the target.

Figure 9. The Break Button of the Debugger Window


Freescale CodeWarrior
File Edit View Search Project Debug Data Tools Window Help
8377_Test.mcp
Debug Version
Files Link Order Targets
debug.elf (core #0, chain position 0)
Stack
Variables: Live Value Location
No local variables
Source:
Line 1 Col 1 Source

File	Code	Data
Source	280	98
MSL	189K	22136
Serial (UART)	2048	409
Runtime	19464	2901
Linker Command File	0	0
Config	0	0
Documentation	0	0
17 files	211K	25544

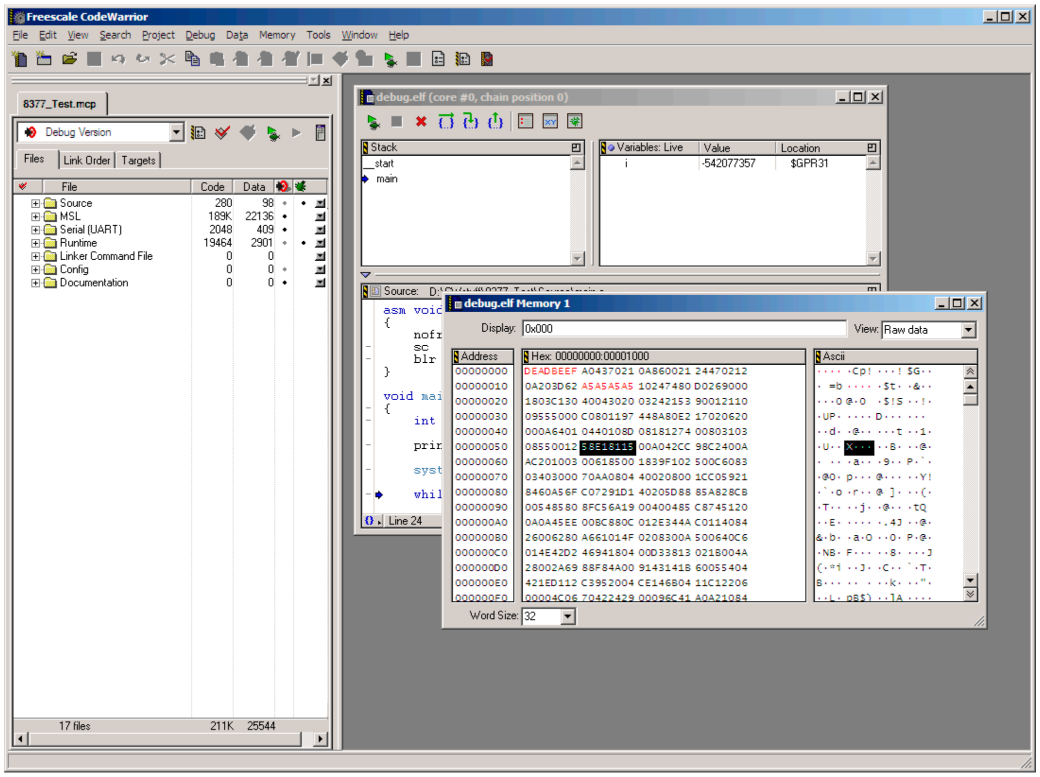
```
Program "debug.elf" is executing.
Choose Break from the Debug menu to stop it.
```

4.1 Displaying Memory

Once the processor is halted, the easiest way to display memory is by selecting “View Memory” from the IDE’s “Data” menu. The Memory window opens. This window can display a range of addresses along with the contents of these addresses. You can type any starting address into the text box labeled “Display” to change the range of addresses shown. Also, you can write a new value to any memory location displayed by typing over the value shown. Changed memory values are highlighted in red. (See Figure 10.)

Also note that your CodeWarrior project must have the same memory offsets defined as the software on the board being debugged. If the CSSRBAR or IMMR values are not set correctly between the `.cfg` file and the board under test, the register display will not show correct values.

Figure 10. The Memory Window



8377_Test.mcp

Debug Version

File	Code	Data
Source	280	98
MSL	189K	22136
Serial (UART)	2048	409
Runtime	19464	2901
Linker Command File	0	0
Config	0	0
Documentation	0	0
17 files	211K	25544

debug.elf (core #0, chain position 0)

Stack: __start, main

Variables: Live	Value	Location
i	-542077357	$GPR31

debug.elf Memory 1

Display: 0x000 View: Raw data

```
Address   Hex: 00000000:00001000
00000000  DEADBEEF A0437021 0A860021 24470212
00000010  0A203D62 A5A5A5A5 10247480 D0269000
00000020  1803C130 40043020 03242153 90012110
00000030  09555000 C0801197 448A80E2 17020620
00000040  000A6401 0440108D 08181274 00803103
00000050  08550012 58E18115 00A042CC 98C2400A
00000060  AC201003 00618500 1839F102 500C6083
00000070  03403000 70AA0804 40020800 1CC05921
00000080  8460A56F C07291D1 40205D88 85A828CB
00000090  00548580 8FC56A19 00400485 C8745120
000000A0  0A0A45EE 00BC880C 012E344A C0114084
000000B0  26006280 A661014F 0208300A 500640C6
000000C0  014E42D2 46941804 00D33813 021B004A
000000D0  28002A69 88F84A00 9143141B 60055404
000000E0  421ED112 C3952004 CE146B04 11C12206
000000F0  00004C06 70422429 00096C41 A0A21084
```

Word Size: 32

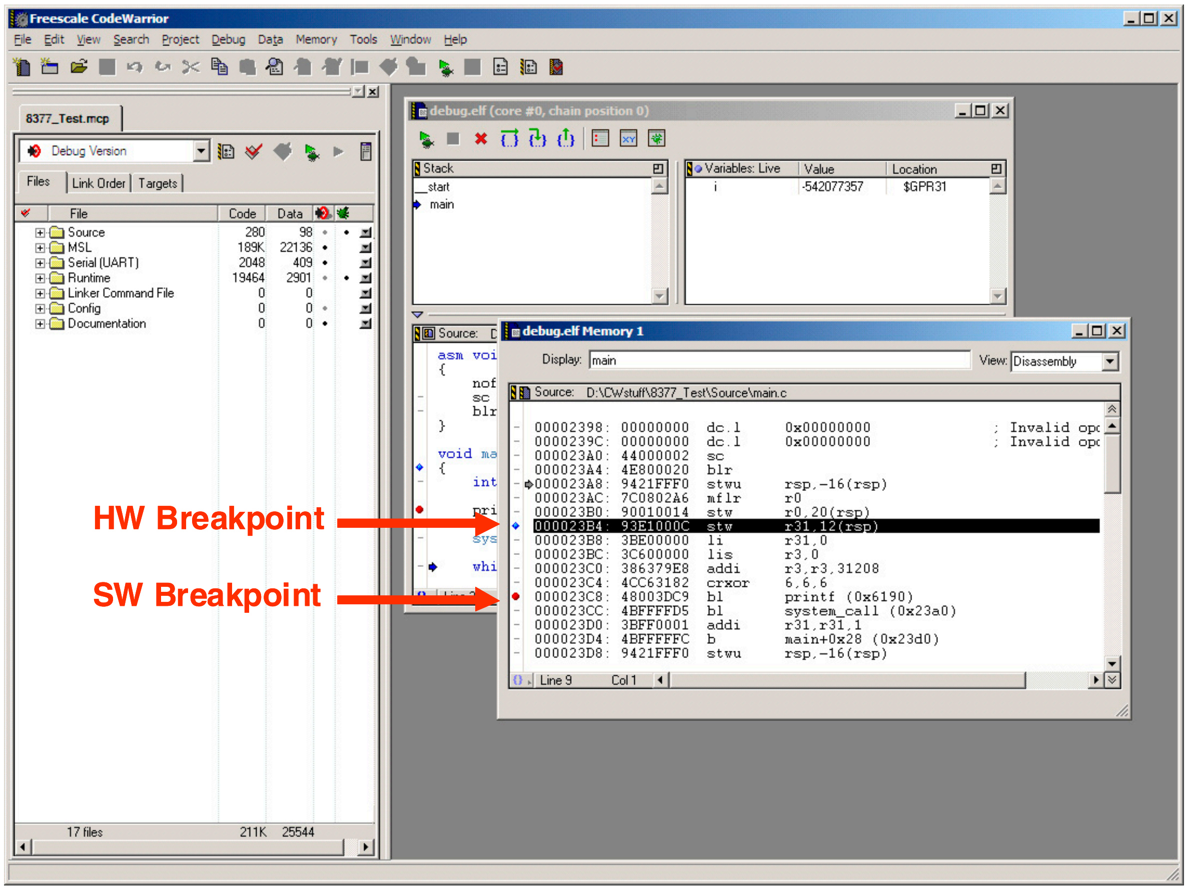
4.2 Setting Breakpoints and Watchpoints

You can also use the Memory window to set breakpoints and watchpoints. To set a breakpoint from this window, change the “View” selection to “Disassembly,” “Source” or “Mixed.” In these views, the Memory window displays the assembly language, C language, or mixed assembly/C language statements at the specified address range. Just as you can set breakpoints in the Debugger window, you can set breakpoints in the Memory window by right-clicking on a statement and selecting “Set Breakpoint” from the context menu that appears.

Note that there are two types of breakpoints: hardware and software. There are a limited number of hardware breakpoints available — the number varies depends on the processor you are using. Software breakpoints modify the code at the location of the breakpoint, while hardware breakpoints use built-in debug functions of the processor core and therefore do not modify the code. If you are setting a breakpoint in flash memory, the CodeWarrior debugger automatically uses a hardware breakpoint because code in flash memory cannot be easily modified.

The debugger shows a hardware breakpoint as a blue diamond, while software breakpoints (which are the default if you select “Set Breakpoint”) are shown as a red circle. (See Figure 11.)

Figure 11. Display of Hardware and Software Breakpoints


Freescale CodeWarrior
File Edit View Search Project Debug Data Memory Tools Window Help
8377_Test.mcp
Debug Version
Files | Link Order | Targets
debug.elf (core #0, chain position 0)
Stack
Variables: Live
debug.elf Memory 1
HW Breakpoint
SW Breakpoint

Display: main   View: Disassembly

Source: D:\CWstuff\8377_Test\Source\main.c

```
00002398: 00000000  dc.l   0x00000000          ; Invalid opc
0000239C: 00000000  dc.l   0x00000000          ; Invalid opc
000023A0: 44000002  sc
000023A4: 4E800020  blr
000023A8: 9421FFF0  stwu   rsp,-16(rsp)
000023AC: 7C0802A6  mflr   r0
000023B0: 90010014  stw    r0,20(rsp)
000023B4: 93E1000C  stw    r31,12(rsp)
000023B8: 3BE00000  li     r31,0
000023BC: 3C600000  lis    r3,0
000023C0: 386379E8  addi   r3,r3,31208
000023C4: 4CC63182  crxor  6,6,6
000023C8: 48003DC9  bl     printf (0x6190)
000023CC: 4BFFFFD5  bl     system_call (0x23a0)
000023D0: 3BFF0001  addi   r31,r31,1
000023D4: 4BFFFFFC  b      main+0x28 (0x23d0)
000023D8: 9421FFF0  stwu   rsp,-16(rsp)
```

File	Code	Data
Source	280	98
MSL	189K	22136
Serial (UART)	2048	409
Runtime	19464	2901
Linker Command File	0	0
Config	0	0
Documentation	0	0
17 files	211K	25544

Variables: Live	Value	Location
i	-542077357	$GPR31

You can also set watchpoints from the Memory window. Watchpoints stop execution if the value of any of the memory locations selected changes. To set a watchpoint, highlight the values in the Memory window that you want to monitor, then right-click and select “Set Watchpoint” from the context menu that appears. Alternatively, you can select “Set Watchpoint” from the IDE’s “Debug” menu.

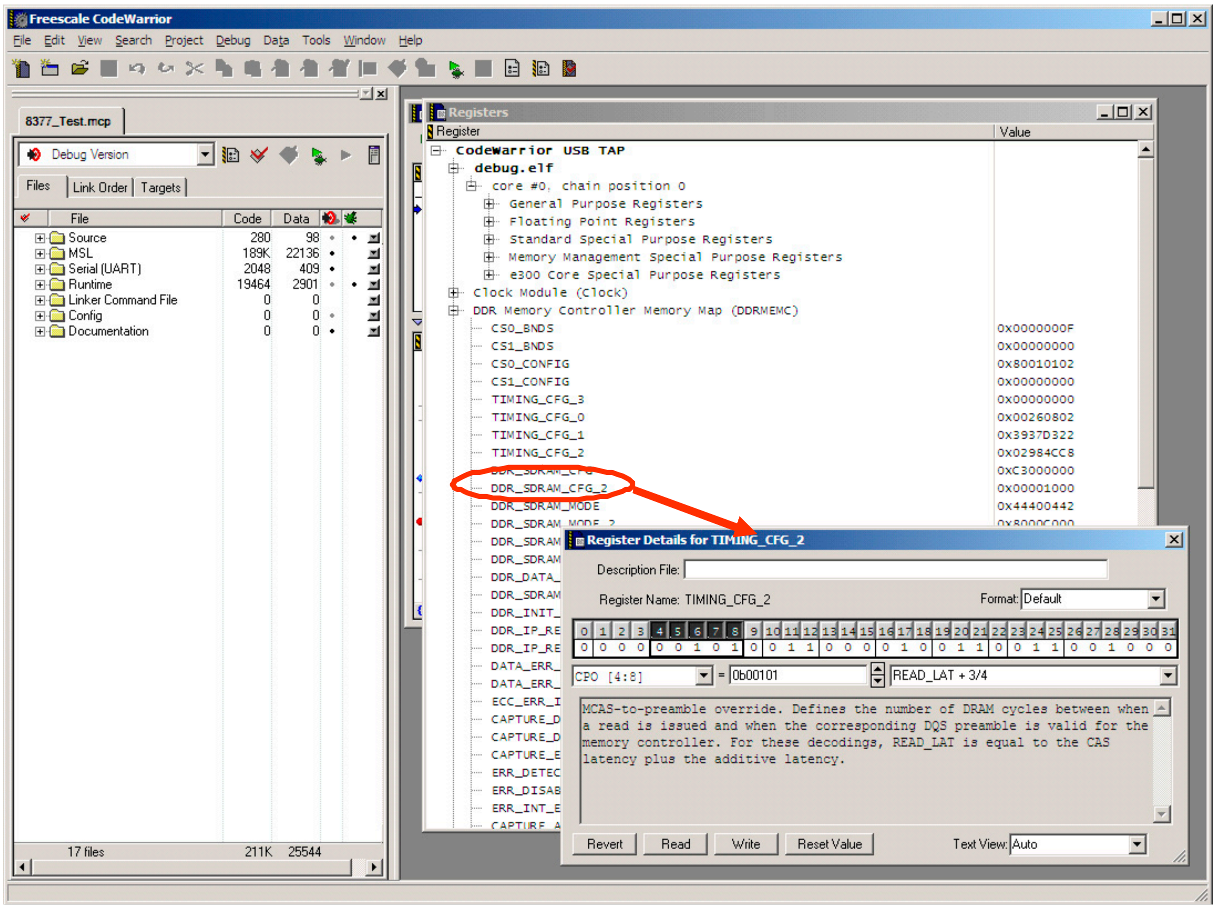
4.3 Displaying Registers

The CodeWarrior debugger has several useful features for displaying the target processor’s internal registers. Although the Memory window can display many of the memory-mapped registers, the Register window includes register names and breaks them into categories. To display the Register window, select “Registers” from the IDE’s “View” menu. Once open, this window displays all of the processor’s internal registers, organized by category. (See Figure 12.)

For example, you can find the settings for the DDR memory controller by expanding the “DDR Memory Controller Memory Map” heading. All of the associated DDR registers are shown along with their current values. As with the Memory window, you can change the displayed register values by selecting and overwriting the values shown. Any changed values are highlighted in red.

Another useful feature is the “Register Details” window, which provides a detailed description of each of the bit fields of the selected register. To display this window, right-click on a register and select “Register Details” from the context menu that appears. Once the Register Details window is open, you can select individual fields to display a more complete explanation. You can also use this window to change values in a register.

Figure 12. The Register Window and the Register Details Window


Freescale CodeWarrior
File Edit View Search Project Debug Data Tools Window Help
8377_Test.mcp
Debug Version
Files Link Order Targets
Registers
Register Details for TIMING_CFG_2
Revert Read Write Reset Value

File	Code	Data
Source	280	98
MSL	189K	22136
Serial (UART)	2048	409
Runtime	19464	2901
Linker Command File	0	0
Config	0	0
Documentation	0	0
17 files	211K	25544

Register	Value
CodeWarrior USB TAP	
debug.elf	
core #0, chain position 0	
General Purpose Registers	
Floating Point Registers	
Standard Special Purpose Registers	
Memory Management Special Purpose Registers	
e300 Core Special Purpose Registers	
Clock Module (Clock)	
DDR Memory Controller Memory Map (DDRMEMC)	
CS0_BNDS	0x0000000F
CS1_BNDS	0x00000000
CS0_CONFIG	0x80010102
CS1_CONFIG	0x00000000
TIMING_CFG_3	0x00000000
TIMING_CFG_0	0x00260802
TIMING_CFG_1	0x3937D322
TIMING_CFG_2	0x02984CC8
DDR_SDRAM_CFG	0xC3000000
DDR_SDRAM_CFG_2	0x00001000
DDR_SDRAM_MODE	0x44400442
DDR_SDRAM_MODE_2	0x8000C000

Description File:

Register Name: TIMING_CFG_2 Format: Default

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0	1	0	0	1	1	0	0	1	1	0	0	1	0	0	0

CPO [4:8] = 0b00101 READ_LAT + 3/4

```
MCAS-to-preamble override. Defines the number of DRAM cycles between when
a read is issued and when the corresponding DQS preamble is valid for the
memory controller. For these decodings, READ_LAT is equal to the CAS
latency plus the additive latency.
```

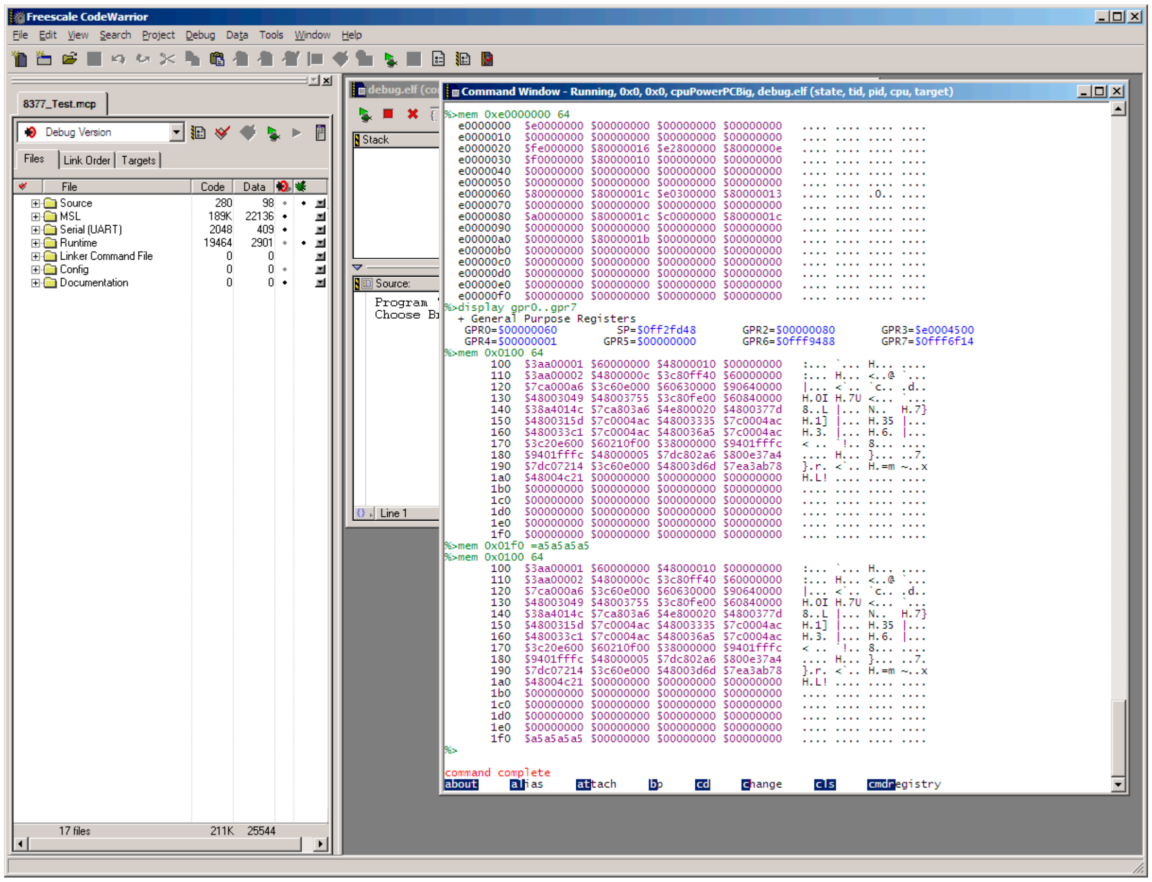
Text View: Auto

5 Using the Command Window

The Command window provides another useful method for communicating with a target device. To display this window, select “Command Window” from the IDE’s “View” menu. (See Figure 13.) The debugger must be connected to the target before you can use the Command window to access the target.

From the Command window, you can enter a variety of commands at the prompt. Type “help” to display a complete list of commands and usage options. “Mem” is a commonly used command for reading and writing memory. You can also use the “Display” command to display register and memory contents.

Figure 13. The Command Window



Command Window - Running, 0x0, 0x0, cpuPowerPCBig, debug.elf (state, tid, pid, cpu, target)

```
%>mem 0xe0000000 64
  e0000000  $e0000000 $00000000 $00000000 $00000000   .... .... .... ....
  e0000010  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
  e0000020  $fe000000 $80000016 $e2800000 $8000000e   .... .... .... ....
  e0000030  $f0000000 $80000010 $00000000 $00000000   .... .... .... ....
  e0000040  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
  e0000050  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
  e0000060  $80000000 $8000001c $e0300000 $80000013   .... .... .0.. ....
  e0000070  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
  e0000080  $a0000000 $8000001c $c0000000 $8000001c   .... .... .... ....
  e0000090  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
  e00000a0  $00000000 $8000001b $00000000 $00000000   .... .... .... ....
  e00000b0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
  e00000c0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
  e00000d0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
  e00000e0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
  e00000f0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
%>display gpr0..gpr7
  + General Purpose Registers
   GPR0=$00000060         SP=$0ff2fd48       GPR2=$00000080       GPR3=$e0004500
   GPR4=$00000001       GPR5=$00000000       GPR6=$0fff9488       GPR7=$0fff6f14
%>mem 0x0100 64
       100  $3aa00001 $60000000 $48000010 $00000000   :... `... H... ....
       110  $3aa00002 $4800000c $3c80ff40 $60000000   :... H... <..@ `...
       120  $7ca000a6 $3c60e000 $60630000 $90640000   |... <`.. `c.. .d..
       130  $48003049 $48003755 $3c80fe00 $60840000   H.0I H.7U <... `...
       140  $38a4014c $7ca803a6 $4e800020 $4800377d   8..L |... N..  H.7}
       150  $4800315d $7c0004ac $48003335 $7c0004ac   H.1] |... H.35 |...
       160  $480033c1 $7c0004ac $480036a5 $7c0004ac   H.3. |... H.6. |...
       170  $3c20e600 $60210f00 $38000000 $9401fffc   < .. `!.. 8... ....
       180  $9401fffc $48000005 $7dc802a6 $800e37a4   .... H... }... ..7.
       190  $7dc07214 $3c60e000 $48003d6d $7ea3ab78   }.r. <`.. H.=m ~..x
       1a0  $48004c21 $00000000 $00000000 $00000000   H.L! .... .... ....
       1b0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
       1c0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
       1d0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
       1e0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
       1f0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
%>mem 0x01f0 =a5a5a5a5
%>mem 0x0100 64
       100  $3aa00001 $60000000 $48000010 $00000000   :... `... H... ....
       110  $3aa00002 $4800000c $3c80ff40 $60000000   :... H... <..@ `...
       120  $7ca000a6 $3c60e000 $60630000 $90640000   |... <`.. `c.. .d..
       130  $48003049 $48003755 $3c80fe00 $60840000   H.0I H.7U <... `...
       140  $38a4014c $7ca803a6 $4e800020 $4800377d   8..L |... N..  H.7}
       150  $4800315d $7c0004ac $48003335 $7c0004ac   H.1] |... H.35 |...
       160  $480033c1 $7c0004ac $480036a5 $7c0004ac   H.3. |... H.6. |...
       170  $3c20e600 $60210f00 $38000000 $9401fffc   < .. `!.. 8... ....
       180  $9401fffc $48000005 $7dc802a6 $800e37a4   .... H... }... ..7.
       190  $7dc07214 $3c60e000 $48003d6d $7ea3ab78   }.r. <`.. H.=m ~..x
       1a0  $48004c21 $00000000 $00000000 $00000000   H.L! .... .... ....
       1b0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
       1c0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
       1d0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
       1e0  $00000000 $00000000 $00000000 $00000000   .... .... .... ....
       1f0  $a5a5a5a5 $00000000 $00000000 $00000000   .... .... .... ....
%>

command complete
about     alias     attach     bp     cd     change     cls     cmdregistry
```

File	Code	Data
Source	280	98
MSL	189K	22136
Serial (UART)	2048	409
Runtime	19464	2901
Linker Command File	0	0
Config	0	0
Documentation	0	0
17 files	211K	25544

There are several advantages to using the Command window in a debug session:

- If you use the Command window to read and write memory, the target does not have to be halted. Instead, the debugger briefly halts the target to perform a memory access, and then immediately resumes execution of the target. In contrast, if you use the Memory window or the Register window, the target must be halted by clicking the break button.
- The Command window displays a history of the commands you issue and their results. Further, you can capture this history in a log file using the “log” command, thereby preserving the results for analysis. This feature is extremely useful for capturing memory dumps that reflect the state of a device while debugging. You can then send these dumps to others for discussion and analysis.

- You can perform a simple access to the target via the Command window, which is a good way to ensure that the board is still alive and that the JTAG connection is still in place. With the CodeWarrior IDE alone, sometimes the target gets reset, but the open thread in the IDE is not aware that the JTAG connection has been lost until you attempt to communicate with the board. You may be waiting for a breakpoint to be hit, while the connection to the target has actually been lost. Trying a simple access via the Command window lets you verify that the connection is still valid.
- With the Command window, you can specify and perform memory reads and writes as 8-, 16-, or 32-bit accesses.

6 An Example Debug Session

Let’s use an example to show how your CodeWarrior tools can be used to debug a customer board.

Problem: A production MPC8347-based board is randomly resetting at certain temperatures in the customer system. After the problem occurs, the COP port cannot be used to reliably access the processor. Attempts to reproduce the problem on a standalone board, outside of the production shelf, have been unsuccessful.

Strategy: We need to have the CodeWarrior debugger attached to the board via the USB TAP when the problem occurs. The hypothesis is that a machine check condition is occurring on the board, which is causing the MPC8347 to enter a checkstop condition and reset. To trigger on the occurrence of the machine check, we must change the settings so that a machine check condition causes an interrupt rather than resetting the board. Then, we can set a breakpoint within the machine check interrupt routine at offset 0x200. Once we hit the breakpoint, we can display memory and register settings to determine the cause of the error. In our setup, the MPC8347-based is board plugged into the customer shelf with the USB TAP attached. The shelf must be in a temperature chamber to recreate the issue, while the computer running the CodeWarrior IDE is outside of the chamber, but communicates with the board via the USB TAP.

Debug Steps

1. Use the EPPC New Project Wizard to create a simple MPC8347 project.
2. Ensure that the USB TAP is properly connected to the customer board.
3. Ensure that the customer board is running properly.
4. Attach to the MPC8347 device using the “Attach to Process” option.

   A Debugger window opens, but code continues to execute on the customer board.
5. Click the red square in the Debugger window.

   The processor halts.
6. Select Data > View Memory to display the Memory window.
7. Display the memory at location 0x200 (the machine check interrupt service routine).
8. Select “Disassembly” in the Memory window so you can see the assembly language code.
9. Set a breakpoint in the interrupt service routine at a point **after** SRR0 and SRR1 have been saved.

   Because the breakpoint itself causes an interrupt, interrupting before the return address and context are saved is not recoverable.
10. Click “Run” (the green triangle) to resume execution.

    At this point, everything is set to capture the machine check interrupt if it occurs.
11. In the IDE, open the Command window and read a memory location periodically to ensure that you are still connected and that the processor is still alive.

    With this type of problem, it is possible that the board could reset without any indication to the CodeWarrior debugger that the connection was lost.

    Assuming the hypothesis regarding the machine check is correct, the breakpoint will eventually be hit and the processor will halt.

12. Once halted, examine register settings and memory to identify the cause of the problem.
13. If needed, use the Command window along with the "log" command to capture a memory or register dump for later analysis.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

***For Literature Requests Only*:**
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.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.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. CodeWarrior™ is a trademark or registered trademark of Freescale Semiconductor, Inc. StarCore® is a registered trademark of Freescale Semiconductor, Inc. in the United States and/or other countries. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2008-2009. All rights reserved.

Document Number: AN3830
Revision 0
02/2009