Application Note

AN2616
2/2004, Rev. 1

Getting Started with HCS08 and CodeWarrior Using C

By Stephen Pickering
8/16-bit Systems Group
East Kilbride, Scotland

Overview

The purpose of this application note is to demonstrate how easy it is to develop C code for an HCS08, using Metrowerks CodeWarrior®. A simple application is used in order to explain the techniques in developing an HCS08 application with CodeWarrior.

An important difference between the HC08 and the HCS08 is the inclusion of a suite of on-board hardware debugging facilities, designed to be used via the BDC (background debug controller). Code is debugged using the HCS08 BDM (background debug mode) pod, along with one of the demonstration/evaluation boards with an MC9S08GB60 device. The BDM pod used in the application is a P&E USB ML 12 (except where stated). Since BDM is an in-circuit debug methodology, the hardware could be the real application rather than a demo/evaluation board.

During the MC9S08GB60 launch, two boards (manufactured by Axiom) were made available: a low-cost demo board, and a more comprehensive evaluation board with LCD.

Although the document refers to the MC9S08GB60, the concepts apply to all the HCS08 devices, with the appropriate substitutions (for device name, header file names, etc.).

If CodeWarrior version 3 (or higher) is not installed, refer to **Metrowerks HCS08 CodeWarrior Development Tools on page 34**.

If you are using Metrowerks CodeWarrior and/or HCS08 for the first time, and you are unsure about writing C code in this environment, refer to **CodeWarrior C and HCS08 on page 37**, for a quick introduction to how the device specific features are used in the Metrowerks C programming language.

Contents

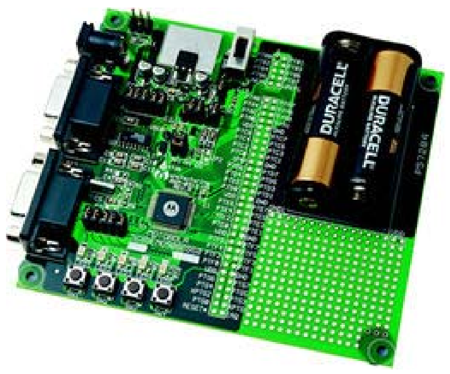
HCS08 Demonstration and Evaluation Boards

At the time of writing of this application note, the two Motorola HCS08 boards shown below are available for the HCS08. Both use the MC9S08GB60.

CAUTION: ***The version of Codewarrior supplied may not support your chosen device or connection method; check Metrowerks website for relevant service packs (see FAQs).***

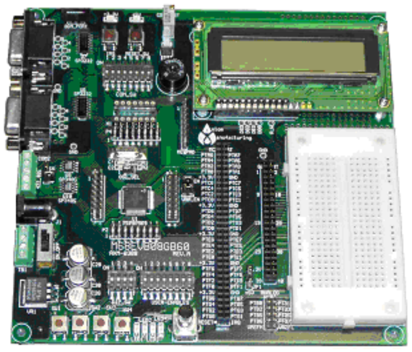
Using BDM cables: see FAQ for advice on using and how to update driver.

M68DEMO908GB60
Demonstration Board



- DB9 Serial Cable
- Documentation (CD)
- Metrowerks CodeWarrior for HC08 and HCS08
- Manual
- Batteries

M68EVB908GB60
MC9S08GB60 Evaluation Board (EVB)



- DB9 Serial Cable
- Support CD
- Metrowerks CodeWarrior for HC08 and HCS08
- Manuals
- Power supply

Introduction

The aim of this application note is to help the first time user of HCS08 and Metrowerks CodeWarrior C to be able to:

- understand the major differences between BDM and monitor mode

- create a project using CodeWarrior
- understand how to create and add files to a project
- connect the hardware (demo or evaluation board) via monitor mode or BDM
- run the code in the hardware
- add code to illuminate LED according to a switch and step through code observing its operation
- add interrupt handlers to a program
- initialize and use the PWM to flash an LED with 25:75 duty cycle and inverse depending upon a switch
- configure the FLL and on-board crystal for maximum performance.

In addition, the reader is provided with:

- an explanation of how Metrowerks defines HCS08 devices and how the device registers are used with C
- answers to commonly asked questions
- pointers and references to further information
- exercises in creating code.

As the exercises are not intended to be a lesson in typing, the appropriate files are included within the zip file AN2616SW.zip. The files main.c and M68DEMO908GB60.h are contained in the relevant sub directory within the zip file, and can simply be dragged from the zip file into the Sources directory of the CodeWarrior project at the appropriate point. For example, the directory 2.4 within the zip file contains the correct files for section 2.4. Alternatively, the code can be copied from an electronic copy (pdf) of the application note, and pasted into the relevant file using the CodeWarrior editor.

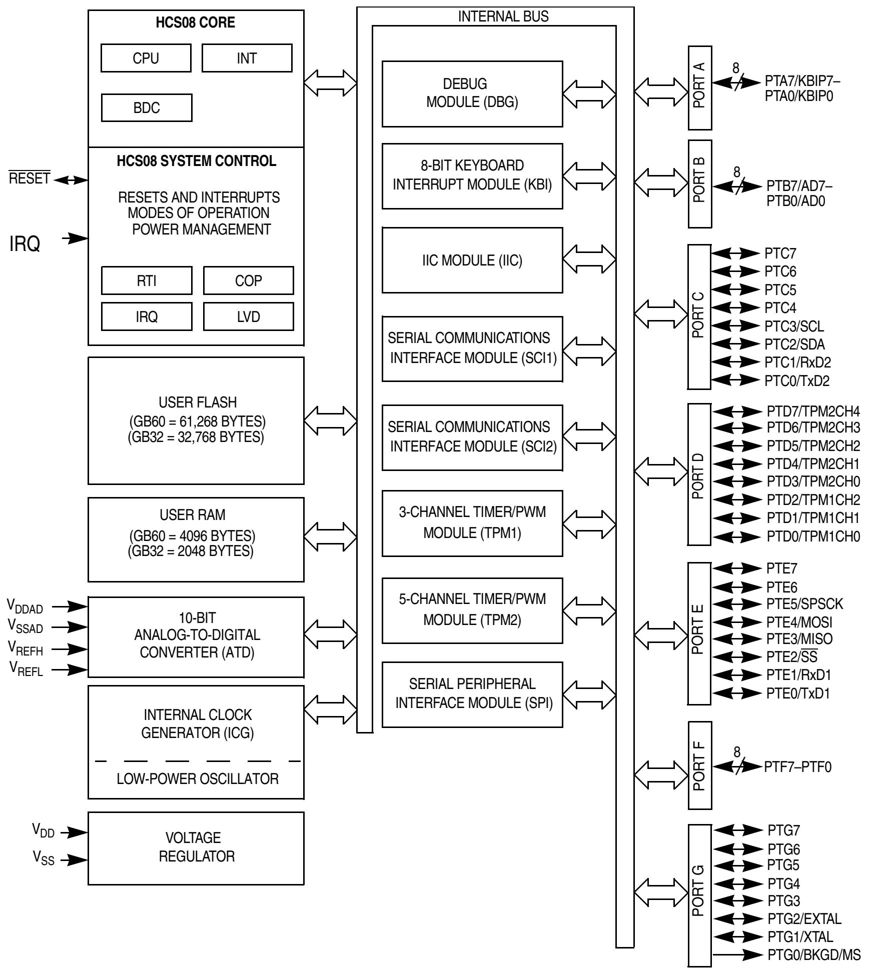
What is an HCS08?

The HCS08 is Motorola's latest range of 8-bit low-power, high-performance microcontrollers based on the HC08 core. The major differences from the HC08 are as follows.

- Core bus speed increased from 8 MHz to 20 MHz
- Hardware Background Debug Controller (BDC) and On-chip Debug Module (DBG), providing an additional two breakpoints in addition to the single breakpoint capability of the HC08, for improved debug and FLASH programming support
- Additional addressing modes for improved stack usage resulting in improved code density and performance

NOTE: *Instruction cycle timings have changed on some instructions/addressing modes due to design requirements to enable higher frequency operation and care should be exercised with time critical code that was used on an HC08.*

The following block diagram shows the major elements of the MC9S08GB60.



Background debug mode

Background debug mode (BDM) is a term used to refer to a mode of operation of the HCS08 where an in-circuit debugging technique is used. To use BDM, an interface typically referred to as a “BDM pod” is used to connect an external debugger, such as Metrowerks HCS08 True-Time Simulator and Real-Time Debugger, running on a PC, to the device being debugged.

A major advantage of Motorola’s BDM is its single pin operation, which allows all other pins to be used for the application (unlike, for example, JTAG, which requires four pins to operate).

The functional components of an HCS08 that provide this debugging capability are the BDC and DBG sections of the core.

- Background Debug Controller (BDC) — the module that controls access to the HCS08 core
- DeBuG module (DBG) — essentially a dual breakpoint controller and 8-word (16-bit) FIFO trace buffer

HCS08 serial monitor

As the HCS08 does not have a ROM-resident monitor like the HC08, Motorola has written a monitor; this is programmed into the HCS08 device and occupies about 1K of FLASH memory. The monitor enables debugging of an HCS08 through one of the on-board serial modules (SCI1) and provides a seamless integration with the CodeWarrior tools, providing almost all the functionality of BDM, albeit slightly slower.

NOTE: *The HCS08 monitor is not the same as or compatible with the HC08 monitor commonly referred to as MON08.*

BDM or monitor mode?

This application note was written with the use of a BDM pod (USB version), and verified using the HCS08 serial monitor mode and parallel BDM. Differences in operation between BDM and monitor mode are discussed, where appropriate.

The major advantages of using BDM over HCS08 serial monitor mode are as follows.

- Only one pin of the target device is used
- BDM does not use any peripherals
- BDM does not use any RAM or FLASH
- BDM cannot be locked out due to interrupts being disabled
- BDM can operate at any processor speed
- BDM can re-sync if processor speed changes during debug
- Device programming times are faster

Developing an Application with CodeWarrior

This section develops a simple application which illustrates the development process involved in using CodeWarrior.

The application uses the M68DEMO908GB60 board, and its five LEDs and four switches. When the complete application is built, the code will:

- initialize peripherals
- set up interrupt handler
- configure a PWM waveform
- turn off all LEDs

- pulse LED5 with 75% duty cycle off; set LED3 on
- set the processor speed with the FLL

and will then loop indefinitely around code which will:

- cause an interrupt if SW1 is pressed:
  - The Interrupt handler will set LED1 (and latch it) according to state of switch SW4: i.e. ON if switch is pressed; otherwise it will be switched OFF
- set or clear LED2 according to the state of switch SW2:
  - Pressing switch SW2 will set LED2, releasing SW2 will clear LED2
- check the state of switch SW4 when switch SW3 is pressed, and set/clear LED3 and LED4 accordingly:
  - If SW4 is pressed
    - set LED4, clear LED3 and Pulse LED5 with 75% duty cycle on
  - Otherwise
    - set LED3, clear LED4 and Pulse LED5 with 75% duty cycle off

This application will be developed incrementally, showing the steps involved in creating the application and explaining the steps taken.

NOTE: *Because of the differences between the demo board and the evaluation board, the buzzer on the evaluation board is used as an audible alternative to LED5 output on the demo board.*

Description of hardware used

The MC9S08GB60 evaluation board (M68EVB908GB60) or the low-cost demo board (M68DEMO908GB60) may be used. The application chosen for the demonstration will work on either board; however, setting up the boards to enable the switches, LEDs, and monitor mode will be different.

Setting up the demo board

To use the demo board with this application note, the LEDs and switches must be suitably configured as follows.

- The five jumpers on LED_EN must be installed to enable the LEDs
- To enable HCS08 serial monitor mode, the serial port SCI transmit and receive must be connected to the DB9 header, and power applied to the RS232 driver; this is accomplished by installing jumpers 1, 2 and 3 on COM_EN

Setting up the evaluation board

To use the evaluation board with this application note, the LEDs and switches must be suitably configured as follows.

- The USER-ENABLES dip switch 2 should be set ON; this connects the switches SW[1:4] on the evaluation board to PORTA[4:7]

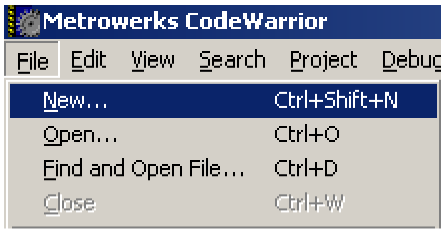
- The USER-ENABLES dip switches 5,6,7 and 8 should be set ON; this connects the LEDs on the evaluation board to PORTF[0:3]
- The COM_SW dip switch 8 should be set to ON; this enables the on-board buzzer
- To enable HCS08 serial monitor mode, the serial port SCI receive must be connected to the DB9 header via the RS232 driver; this is accomplished by setting switch 1 of COM_EN ON

Creating a new project CodeWarrior

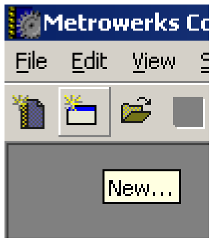
This section discusses the project types that can be created within CodeWarrior.

CodeWarrior has a project wizard that guides you through the process of creating a project.

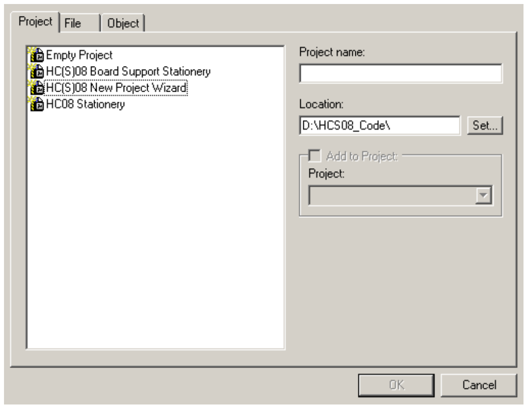
1. Create a new project by selecting “New…” from the File menu:



Or by pressing the New button:



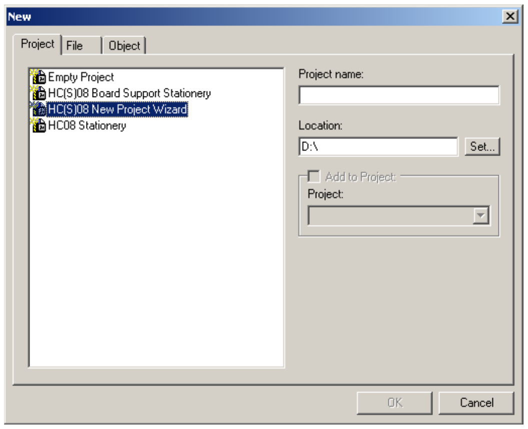
either of which will invoke the project creation dialog box shown below.



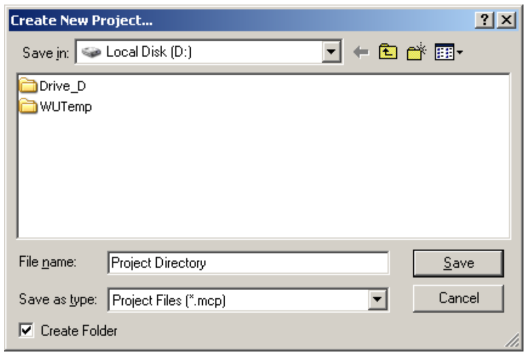
When creating a new project, the project wizard presents four main options:

Empty Project	Creates the very minimum of files required, essentially just creating the project file.
HC(S)08 Board Support Stationary	Provided as a quick start to use one of the demo boards, provides some skeleton code (or demo), and can be used to get accustomed quickly to CodeWarrior and a board.
HC(S)08 New Project Wizard	Used to choose the appropriate device on which to base the project, and is the most commonly used wizard.
HC08 Stationary	Provides the HC08 board support of the previous CodeWarrior for the HC08 and is not used for HCS08 devices or board.

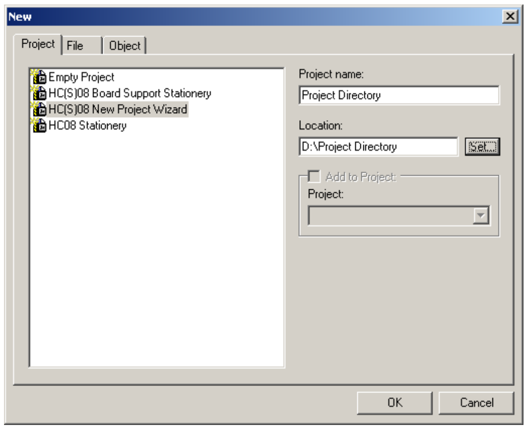
2. Select “HC(S)08 New Project Wizard”.



3. Select Set... button to set the “Location”. This allows you to select the parent directory for the project:



4. Once the root directory for the project has been located, enter the directory to be created (for example, “Project Directory”, as shown in the diagram above).
5. Select Save, and the new project directory / file name will be displayed.
6. In the “Project name:” field, you may enter the name of the project file to be created (by default it will be the same as the directory name):



New

Project | File | Object

- Empty Project
- HC(S)08 Board Support Stationery
- HC(S)08 New Project Wizard
- HC08 Stationery

Project name: Project Directory

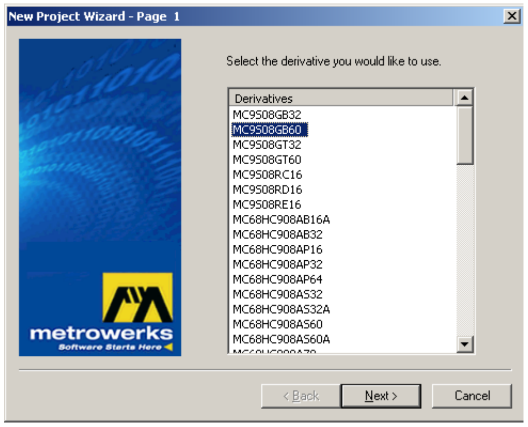
Location: D:\Project Directory Set...

Add to Project

Project:

OK Cancel

7. Select OK. A popup will appear which lists the possible devices:

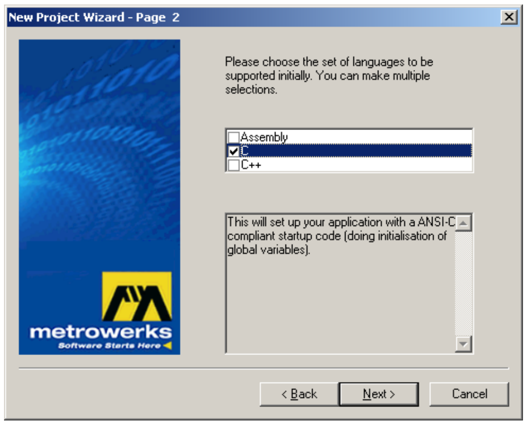


New Project Wizard - Page 1

Select the derivative you would like to use.

Derivatives
- MC9S08GB32
- MC9S08GB60
- MC9S08GT32
- MC9S08GT60
- MC9S08RC16
- MC9S08RD16
- MC9S08RE16
- MC68HC908AB16A
- MC68HC908AB32
- MC68HC908AP16
- MC68HC908AP32
- MC68HC908AP64
- MC68HC908AS32
- MC68HC908AS32A
- MC68HC908AS60
- MC68HC908AS60A

8. Select your desired device (MC9S08GB60 in this example), and press Next >, which will bring up the language dialog box:

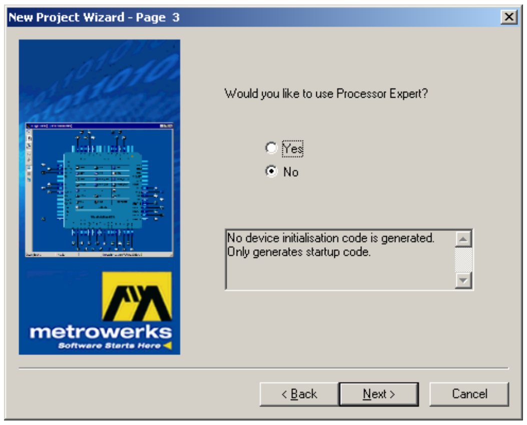

New Project Wizard - Page 2

Please choose the set of languages to be supported initially. You can make multiple selections.

- ☐ Assembly
- ☑ C
- ☐ C++

This will set up your application with a ANSI-C compliant startup code (doing initialisation of global variables).

9. Select C and press Next >. This will bring up the processor expert dialog box:


New Project Wizard - Page 3

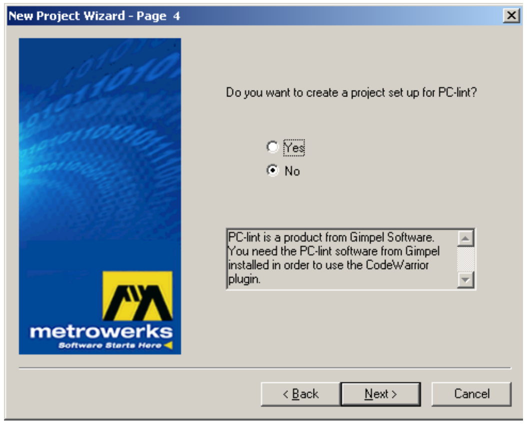
Would you like to use Processor Expert?

○ Yes [✓ No]

No device initialisation code is generated. Only generates startup code.

As this application note is about using C with the HCS08 and CodeWarrior, “Processor Expert” is not selected. If you wish to use “Processor Expert”, refer to the online help (and examples) before proceeding; otherwise:

10. Select Next >, and the PC-lint option will appear:

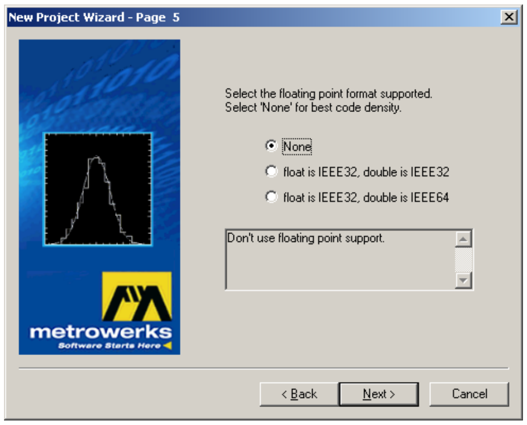

New Project Wizard - Page 4

Do you want to create a project set up for PC-lint?

Yes [✓ No]

PC-lint is a product from Gimpel Software. You need the PC-lint software from Gimpel installed in order to use the CodeWarrior plugin.

11. Unless you have purchased PC-lint and wish to use it, select “No” and press Next >, and the floating point support dialog box will appear:

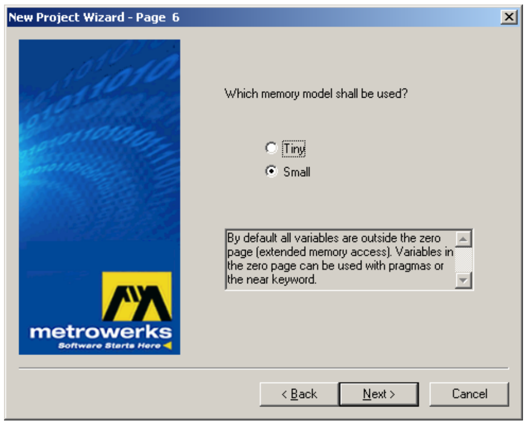

New Project Wizard - Page 5

Select the floating point format supported.
Select 'None' for best code density.

[✓ None] float is IEEE32, double is IEEE32 float is IEEE32, double is IEEE64

Don't use floating point support.

12. As the application will not use floating point support, select “None” and press Next >. The memory model selection will appear:

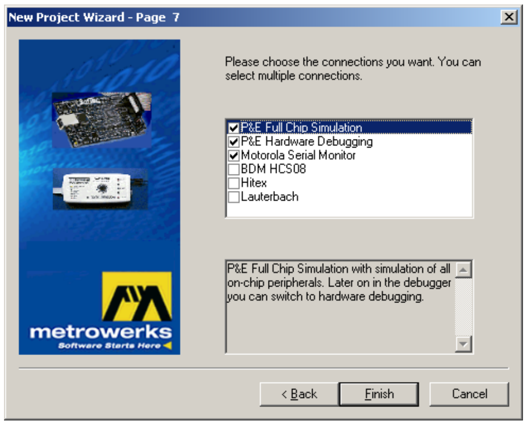


Which memory model shall be used?

Tiny [✓ Small]

By default all variables are outside the zero page (extended memory access). Variables in the zero page can be used with pragmas or the near keyword.

13. Select “Small” and press Next >. The connection dialog will appear:



Please choose the connections you want. You can select multiple connections.

- ✓ P&E Full Chip Simulation
- ✓ P&E Hardware Debugging
- ✓ Motorola Serial Monitor
- BDM HCS08
- Hitex
- Lauterbach

P&E Full Chip Simulation with simulation of all on-chip peripherals. Later on in the debugger you can switch to hardware debugging.

By selecting “P&E Full Chip Simulation”, “P&E Hardware Debugging” and “Motorola Serial Monitor”, it will be possible to use either simulation, the serial monitor within the HCS08 FLASH, or BDM to debug code.

NOTE: *There is no code overhead as a result of choosing multiple connection methods.*

Enabling all the likely connection methods makes switching between the different targets easier. For example, initially the development could use the simulator and swap to the HCS08 serial monitor or BDM method as required.

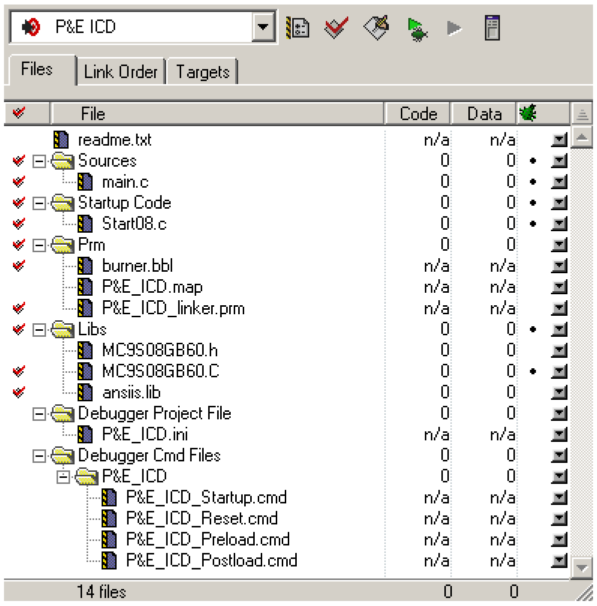
The supported target devices are listed in the following table.

Target		Comment
P&E Full Chip Simulation	P&E FCS	This option allows the on-chip peripherals to be emulated and software simulation of the peripherals.
P&E Hardware Debugging	P&E ICD	This option enables connection to the device via P&E hardware for HC08 and other P&E hardware such as USB HCS08/HCS12 Multilink.
Motorola Serial Monitor	Monitor	This option connects to the HCS08 device through the monitor ROM within the HCS08 device.
BDM HCS08	BDM	Legacy support only — not recommended for use (supports parallel BDM Multilink only). Superseded by P&E Hardware Debugging. Will be removed in future versions of CodeWarrior.
Hitex		Refer to manufacturer's detail.
Lauterbauch		Refer to manufacturer's detail.

Project creation is now complete!

CodeWarrior generated project

Once the project wizard has finished, a project, including a skeleton application, has been created. The project window will look something like:



✓	File	Code	Data	
	readme.txt	n/a	n/a	
✓	Sources	0	0	•
✓	main.c	0	0	•
✓	Startup Code	0	0	•
✓	Start08.c	0	0	•
✓	Prm	0	0	
✓	burner.bbl	n/a	n/a	
	P&E_ICD.map	n/a	n/a	
✓	P&E_ICD_linker.prm	n/a	n/a	
✓	Libs	0	0	•
	MC9S08GB60.h	0	0	
✓	MC9S08GB60.C	0	0	•
✓	ansiis.lib	0	0	
	Debugger Project File	0	0	
	P&E_ICD.ini	n/a	n/a	
	Debugger Cmd Files	0	0	
	P&E_ICD	0	0	
	P&E_ICD_Startup.cmd	n/a	n/a	
	P&E_ICD_Reset.cmd	n/a	n/a	
	P&E_ICD_Preload.cmd	n/a	n/a	
	P&E_ICD_Postload.cmd	n/a	n/a	
	14 files	0	0	

Details of the files and directories are discussed in section "4. CodeWarrior C and HCS08".

Initial code

The new project wizard creates enough code to actually load into a board and run/debug, the main program (main.c) being:

```
#include <hidef.h> /* for EnableInterrupts macro */
#include <MC9S08GB60.h> /* include peripheral declarations */

void main(void) {
  EnableInterrupts; /* enable interrupts */
  /* include your code here */
  for(;;) {
    __RESET_WATCHDOG(); /* kicks the dog */
  } /* loop forever */
}
```

Connecting the hardware

The preferred method of connecting the target device is via a BDM pod, but a standard RS232 serial connection to the monitor of the HCS08 can also be used.

The following sections discuss the BDM and HCS08 serial monitor connection.

Background debug mode (BDM)

To set up: connect the BDM pod to the PC with the appropriate cable supplied with the pod (for example, parallel cable, USB cable), connect the BDM pod to the target board, paying particular attention to ensure correct alignment of pin 1 of the BDM cable with the BDM connector on the board. Refer to the appropriate BDM and board for additional help.

Selecting a BDM

The BDM on the HCS08 evolved from the BDM used on the HCS12 (BDM Version 4).

The preferred method of connecting to an S08 device for code development or debugging is using one of the following BDM interfaces:

- M68MULTILINK12 (P&E BDM Multilink)
- M68MULTILINKS08 (NOT the similarly named M68MULTILINK08)
- P&E USB ML12 / P&E USB HCS08/HCS12 MULTILINK
- P&E Cyclone Pro

NOTE: *M68MULTILINK08 and M68MULTILINKS08 are NOT the same. The M68MULTILINK08 is a MON08 interface for use with HC08 devices; the M68MULTILINKS08 is a BDM pod for HCS08 that is also compatible with HC12 and HCS12.*

HCS08 serial monitor mode

To set up: connect the serial cable to the PC, and connect the other end to SCI1 on the target board. Refer to the appropriate board manual for additional help.

The HCS08 devices supplied with the demo and evaluation boards are pre-programmed with the HCS08 serial monitor. Refer to Motorola application note

"AN2140/D — Serial Monitor for MC9S08GB/GT" for a full description of its capability.

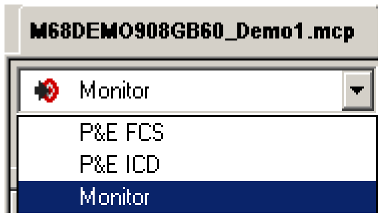
Running the code

The next two sub-sections discuss the use of HCS08 serial monitor mode or BDM as the method of connecting to the target device on either the evaluation or demo board. There are several differences in the dialog boxes and in the progression through the target connection; these differences are detailed in the following sections.

Using monitor

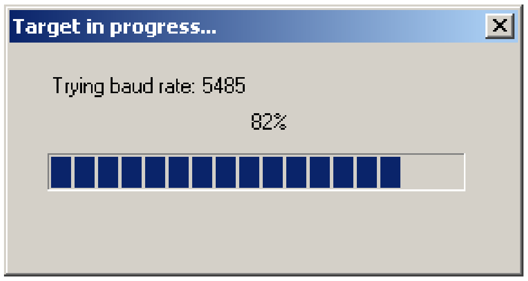
If you are using BDM, skip this section; go directly to **Using BDM on page 17**.

1. From the target connection pull-down, ensure that "Monitor" is selected:



2. Enable monitor mode on the target. (For the demo and evaluation boards, press switch 4, whilst applying power, or whilst pressing the reset switch on the evaluation board).

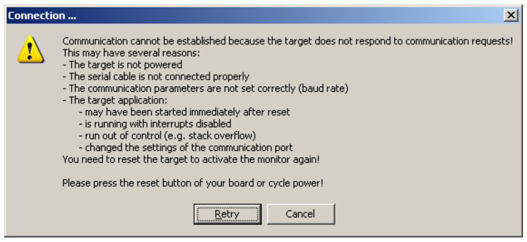
3. Select Debug ( ). This compiles and links the code, and invokes the True-Time simulator and Real-Time debugger (the HCS08 debugger). The debugger will then proceed to try to establish connection with the target device, by trying all the baud rates:


Target in progress...

Trying baud rate: 5485

82%

If the debugger fails to communicate with the device after going through all possible baud rates, a window suggesting possible causes will pop up:

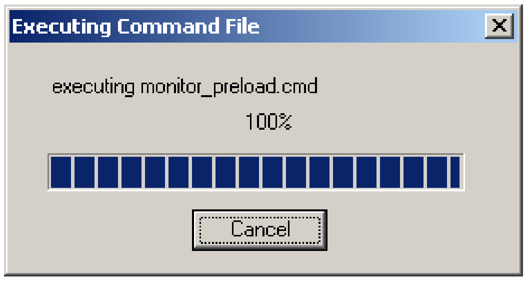

Connection ...

Communication cannot be established because the target does not respond to communication requests!
This may have several reasons:
- The target is not powered
- The serial cable is not connected properly
- The communication parameters are not set correctly (baud rate)
- The target application:
  - may have been started immediately after reset
  - is running with interrupts disabled
  - run out of control (e.g. stack overflow)
  - changed the settings of the communication port
You need to reset the target to activate the monitor again!

Please press the reset button of your board or cycle power!

Power the board off and then on again (holding down switch SW4), and press the Retry button. Communication should now be established.

Once communications have been established between the debugger and the device, the debugger will pop up a few dialog windows as it prepares the device, ready for debugging; the first dialog box is the monitor preload:


Executing Command File

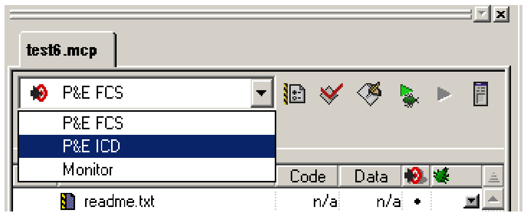
executing monitor_preload.cmd
100%

This will be followed by the FLASH erase/programming and the monitor post load commands.

Skip the next section; go directly to **Debugging — communications established on page 20**.

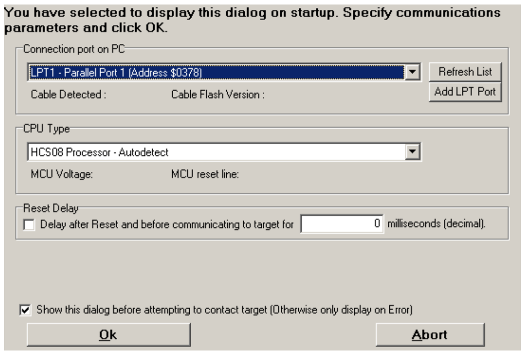
Using BDM

1. From the target connection pull-down, ensure that "P&E ICD" is selected:



2. Select Debug ( ). This compiles and links the code, and invokes the True-Time simulator and Real-Time debugger (the HCS08 debugger). As this is the first time the code has been run, the debugger does not

know which BDM device is to be used to connect to the target, so it pops up a dialog box for configuration (defaulting to use LPT1):


You have selected to display this dialog on startup. Specify communications parameters and click OK.

Connection port on PC: LPT1 - Parallel Port 1 (Address $0378)

Cable Detected : Cable Flash Version :

CPU Type: HCS08 Processor - Autodetect

MCU Voltage: MCU reset line:

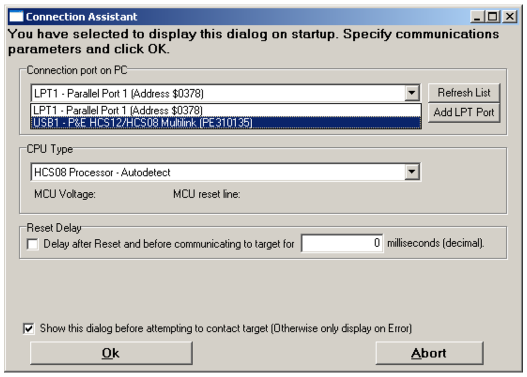
Reset Delay: ☐ Delay after Reset and before communicating to target for 0 milliseconds (decimal).

☑ Show this dialog before attempting to contact target (Otherwise only display on Error)

CodeWarrior knows the CPU Type is an HCS08, because that is the project type we created.

The debugger will list the pods it can see on the pull-down list. If the pod being used is not shown, ensure it is connected correctly, then press the “Refresh List” button. The pod should now appear on the list.

Ensure the relevant pod is highlighted; in this example, P&E USB ML 12 is being used:


Connection Assistant

You have selected to display this dialog on startup. Specify communications parameters and click OK.

Connection port on PC: LPT1 - Parallel Port 1 (Address $0378)

LPT1 - Parallel Port 1 (Address $0378)
USB1 - P&E HCS12/HCS08 Multilink (PE310135)

CPU Type: HCS08 Processor - Autodetect

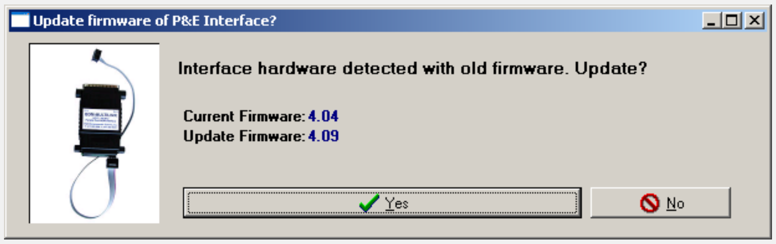
MCU Voltage: MCU reset line:

Reset Delay: ☐ Delay after Reset and before communicating to target for 0 milliseconds (decimal).

☑ Show this dialog before attempting to contact target (Otherwise only display on Error)

3. Select Ok.

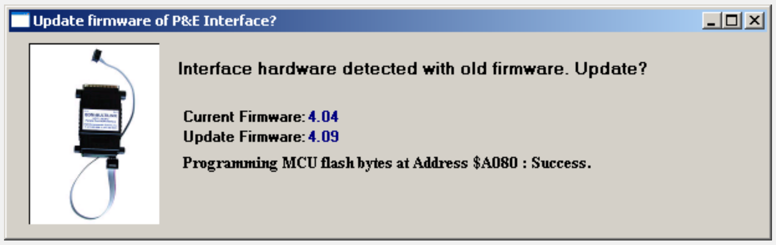
The BDM firmware version will be checked, and, if the pod has old firmware, a dialog box will appear, allowing it to be upgraded:


Update firmware of P&E Interface?

Interface hardware detected with old firmware. Update?

Current Firmware: 4.04
Update Firmware: 4.09

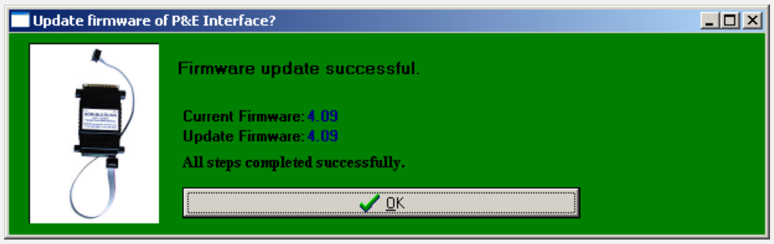
Select ✓ Yes, and a dialog pops up showing the reprogramming of the pod:


Update firmware of P&E Interface?

Interface hardware detected with old firmware. Update?

Current Firmware: 4.04
Update Firmware: 4.09
Programming MCU flash bytes at Address $A080 : Success.

Select Ok.

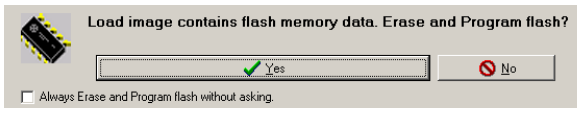

Update firmware of P&E Interface?

Firmware update successful.

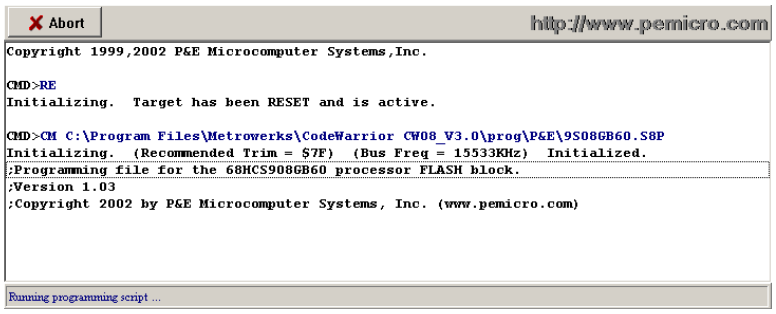
Current Firmware: 4.09
Update Firmware: 4.09
All steps completed successfully.

NOTE: *This will occur only with BDM pods that have old firmware. Once upgraded, the prompt will not recur for a pod that has been updated (unless CodeWarrior is updated and new code is available).*

Next, a dialog pops up asking to erase and program FLASH:


Load image contains flash memory data. Erase and Program flash?

4. Select ✓ Yes to proceed. Another window pops up showing the erasure and programming of the FLASH:



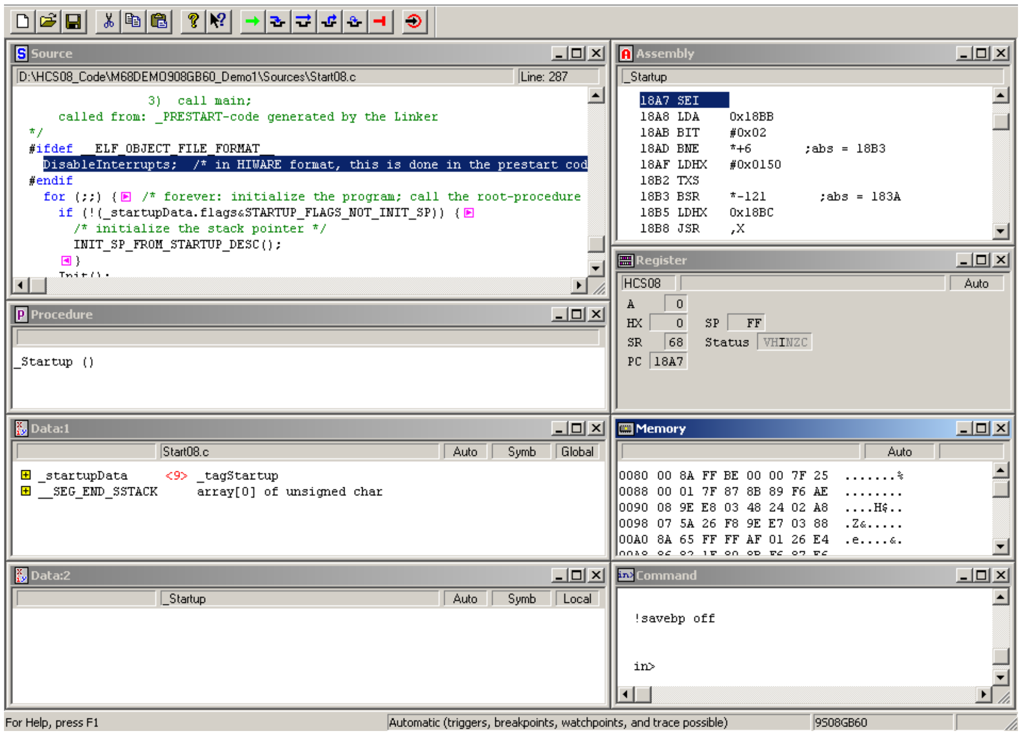
```
Copyright 1999,2002 P&E Microcomputer Systems,Inc.

CMD>RE
Initializing.  Target has been RESET and is active.

CMD>CM C:\Program Files\Metrowerks\CodeWarrior CW08_V3.0\prog\P&E\9S08GB60.S8P
Initializing.  (Recommended Trim = $7F)  (Bus Freq = 15533KHz)  Initialized.
;Programming file for the 68HCS908GB60 processor FLASH block.
;Version 1.03
;Copyright 2002 by P&E Microcomputer Systems, Inc. (www.pemicro.com)
```

Running programming script ...

At this point, the code has been programmed in the device, and the debugger shows the following windows.

Debugging — communications established

Once communication (BDM or Monitor) has been established with the device, the debugger will show its debug window:



D:\HCS08_Code\M68DEMO908GB60_Demo1\Sources\Start08.c Line: 287

```
                 3)  call main;
     called from: _PRESTART-code generated by the Linker
  */
  #ifdef __ELF_OBJECT_FILE_FORMAT__
    DisableInterrupts;  /* in HIWARE format, this is done in the prestart cod
  #endif
    for (;;) {  /* forever: initialize the program; call the root-procedure
      if (!(_startupData.flags&STARTUP_FLAGS_NOT_INIT_SP)) {
        /* initialize the stack pointer */
        INIT_SP_FROM_STARTUP_DESC();
      }
```

```
_Startup ()
```

Start08.c Auto Symb Global

```
_startupData      <9>  _tagStartup
__SEG_END_SSTACK       array[0] of unsigned char
```

_Startup Auto Symb Local

_Startup

```
18A7 SEI
18A8 LDA    0x18BB
18AB BIT    #0x02
18AD BNE    *+6        ;abs = 18B3
18AF LDHX   #0x0150
18B2 TXS
18B3 BSR    *-121        ;abs = 183A
18B5 LDHX   0x18BC
18B8 JSR    ,X
```

HCS08 Auto

```
A      0
HX     0   SP     FF
SR    68   Status VHINZC
PC  18A7
```

Auto

```
0080  00 8A FF BE 00 00 7F 25   .......%
0088  00 01 7F 87 8B 89 F6 AE   ........
0090  08 9E E8 03 48 24 02 A8   ....H$..
0098  07 5A 26 F8 9E E7 03 88   .Z&.....
00A0  8A 65 FF FF AF 01 26 E4   .e....&.
```

```
!savebp off

in>
```

For Help, press F1 | Automatic (triggers, breakpoints, watchpoints, and trace possible) | 9S08GB60

NOTE: *If the debugger does not erase the FLASH, it is almost certainly operating in simulation mode. Refer to the Frequently Asked Questions section for information on how to select the relevant in-circuit debug mode.*

Apart from a few exceptions, debugging will be the same whether the monitor or BDM is used. The notable difference is that, when debugging using the HCS08 serial monitor, loss of communication between the debugger and the device is possible due to the fact that the monitor is code within the FLASH, interrupt driven and susceptible to errors in the user code, for example, disabling interrupts or user code runaway.

The monitor is in protected FLASH but, if it is erased, it will have to be reprogrammed. Refer to Frequently Asked Questions for more information on reprogramming the monitor.

Buttons of interest from the debugger window are:

RUN	Run code from current pc location until either halted or reset
STEP INTO	Execute one C statement; enter into functions
STEP OVER	Execute one C statement or complete function
STEP OUT	Execute remaining C statements in function
Assembly Step	Step one assembly language instruction
HALT	Stop the executing program
RESET	Hardware reset

At this point, running the application will appear to have no effect, as there is no code that actually does anything. The following sections add code that can be run, and that performs real functions.

NOTE: *For a detailed discussion of the HCS08 on-chip debug features using the CodeWarrior interface, refer to application note, “AN2596 — Using the HCS08 family On-Chip Debug System.”*

Adding files to the project

Multiple files can be added to a project to allow a program to be split into logical sections for easier development. As an example, a header file defining the switches and LEDs for the demo and evaluation boards will be added to the project to demonstrate the process.

Header file for demo board

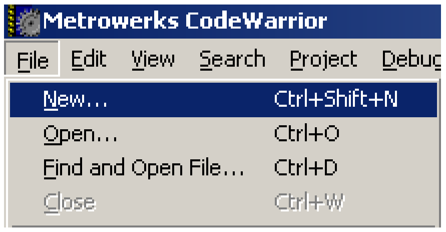
A small header file is created to give more meaningful names to the LEDs and switches on the demo board for use in the application. Another benefit of using an application specific header file is that code will be easier to modify for different devices or applications, if meaningful names are used rather than the port or pin names; for example, in a real application, LED2 may be the STOP LED.

NOTE: *The header will work with the demo board and the evaluation board, as only a subset of the evaluation board is used.*

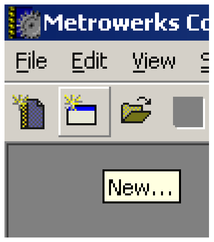
Add a file to a project

In order to add a new file to a project, the file must exist; therefore, a new file must be created. To do this:

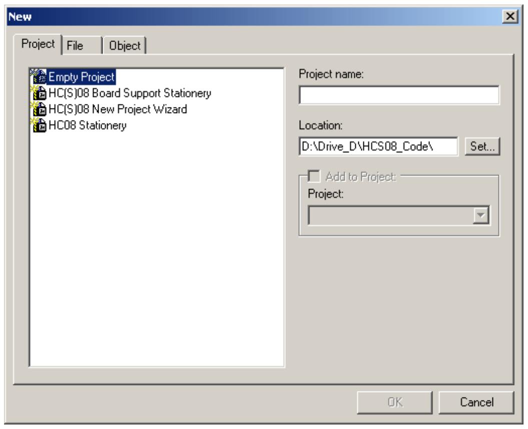
1. From the File menu, select "New…":



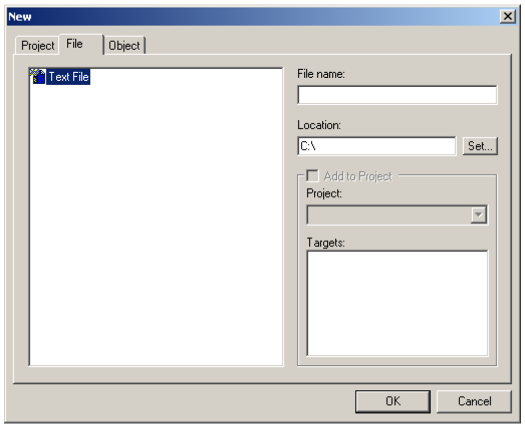
or press the New button:



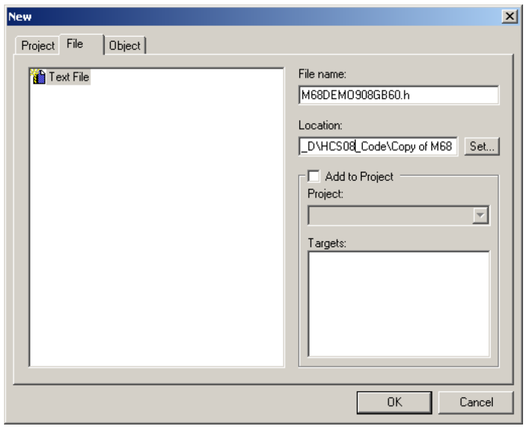
The new dialog window will appear:



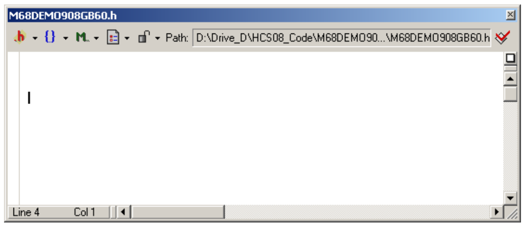
2. Select the “File” tab. There will be one option — create a file:



3. A dialog box will appear. Select Set... button, navigate to the Sources directory within the project directory, and enter the name of the file to create (for example, M68DEMO908GB60.h):



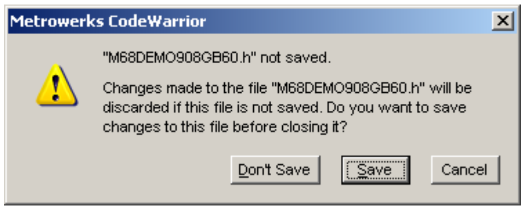
4. Select Ok. This will create an empty text file in the appropriate directory and bring up a text editor window for the file:



5. Enter the following text into the file:

```
/* File: M68DEMO908GB60.h*/

/* include peripheral declarations */
#include <MC9S08GB60.h>

/*define value for led's when on and off*/
#define ON 0
#define OFF 1

/*define value for switches when up (not pressed) and down (pressed)*/
#define UP 1
#define DOWN 0

/*define led's*/
#define LED1 PTFD_PTFD0
#define LED2 PTFD_PTFD1
#define LED3 PTFD_PTFD2
#define LED4 PTFD_PTFD3
#define LED5 PTFD_PTFD4

/*define switches*/
#define SW1 PTAD_PTAD4
#define SW2 PTAD_PTAD5
#define SW3 PTAD_PTAD6
#define SW4 PTAD_PTAD7
```

6. After entering the text, select the close window (☒). This will bring up the save dialog:



Metrowerks CodeWarrior

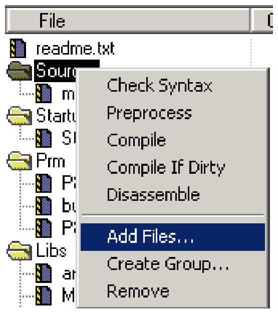
"M68DEMO908GB60.h" not saved.

Changes made to the file "M68DEMO908GB60.h" will be discarded if this file is not saved. Do you want to save changes to this file before closing it?

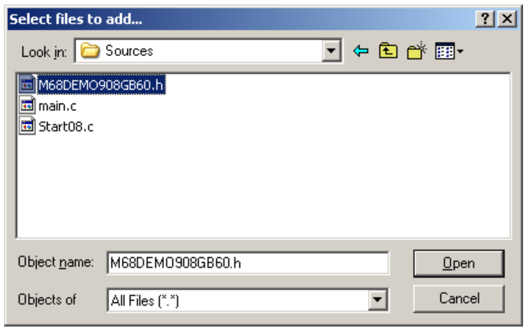
7. Press Save to save the changes made to the file.

Add file to project

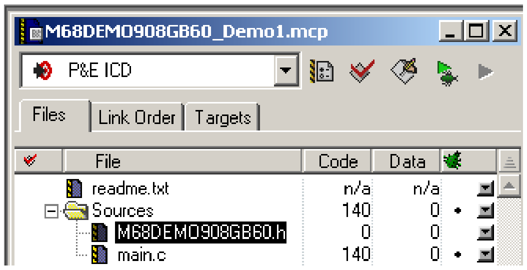
1. To add the header file to the project, select the desired directory (for example, "Sources") in the project manager and press the right mouse button. The following dialog will appear:



2. Select "Add Files…". This will pop up the file dialog window, which is used to locate the file:

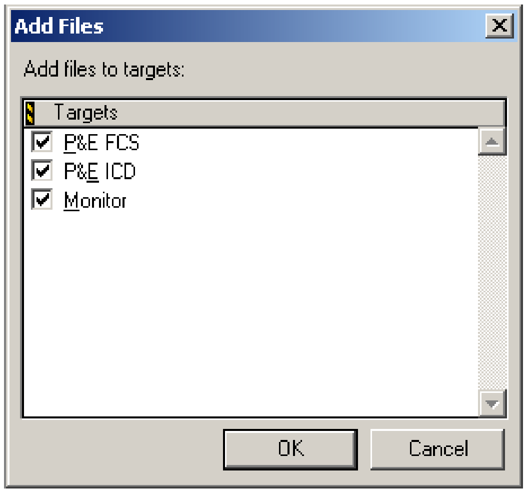


Select Open to add the file to the project and associate it with the "Sources" folder:



File	Code	Data
readme.txt	n/a	n/a
Sources	140	0
M68DEMO908GB60.h	0	0
main.c	140	0

A popup will appear showing a list of available targets that the file can be added to. Select all:



- [✓] P&E FCS
- [✓] P&E ICD
- [✓] Monitor

Illuminate LED2 if SW2 is pressed

Let's add some code to light up LED2 if SW2 is pressed. Double-click on "main.c" in the project manager to invoke the editor.

First, add an include statement to use our header file for the demo board:

```
#include "M68DEMO908GB60.h"
```

Next, configure PORTF for output (LEDs) and PORTD for input (switches):

```
  PTADD = 0;    //initialize as input (Data Direction Register)
  PTAPE = 0xf0; //Pullups on upper 4 bits

  /*initialize bits 0-3 of Port F as outputs (connected to led's)*/
  PTFDD = 0x0f;
  LED1 = OFF;
  LED2 = OFF;
  LED3 = OFF;
  LED4 = OFF;
  LED5 = OFF;
```

Code to turn on LED2 depending upon the state of SW2 would be:

```
LED2 = SW2;
```

To edit the main.c file, simply double-click on the file name within the project window; this will invoke the file editor. By adding this code within the main program, as illustrated below, LED2 will be turned on for as long as SW2 is pressed. The resulting program is:

```
#include <hidef.h> /* for EnableInterrupts macro */
#include <MC9S08GB60.h> /* include peripheral declarations */

#include "M68DEMO908GB60.h"


void main(void) {
  EnableInterrupts; /* enable interrupts */
  /* include your code here */


  PTADD = 0;    //initialize as input (Data Direction Register)
  PTAPE = 0xf0; //Pullups on upper 4 bits

  /*initialize bits 0-3 of Port F as outputs (connected to led's)*/
  PTFDD = 0x0f;
  LED1 = OFF;
  LED2 = OFF;
  LED3 = OFF;
  LED4 = OFF;
  LED5 = OFF;


  for(;;) {
    __RESET_WATCHDOG(); /* kicks the dog */

    LED2 = SW2;

  } /* loop forever */
}
```

Run the code

- In CodeWarrior, press the Debug button (). This compiles the program and produces an executable file, which invokes the debugger. The debugger erases the FLASH and downloads the code to the device.
- In the debugger, press the RUN button (); the application will run.
- The running application can be confirmed by pressing SW2 and seeing LED2 light.
- Press the STOP button (); LED2 will maintain its state prior to the stop being pressed.
- Press the STEP OVER button () to advance the code; the debugger will advance one C instruction with each press.
- With the debugger pointing to the statement LED=SW2, press SW2 and press the STEP OVER button () once; LED2 will be turned on.
- Press the RUN button () to restart the program; this will cause LED2 to be on when SW2 is pressed

Use PWM to flash LED

Adding the following code to the program will configure the PWM and will flash LED5 (or buzzer on evaluation board), using LED3 and LED4 to indicate the duty cycle selected by switch SW4, when changed by the operation of switch SW3.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include <MC9S08GB60.h> /* include peripheral declarations */
#include "M68DEMO908GB60.h"


#define PRESCALAR 7
#define MODULUS 32768
#define DUTY75 (MODULUS-(MODULUS/4))
#define DUTY25 (MODULUS/4)


void main(void) {
  EnableInterrupts; /* enable interrupts */
  /* include your code here */

  PTADD = 0;    //initialize as input (Data Direction Register)
  PTAPE = 0xf0; //Pullups on upper 4 bits

  /*initialize bits 0-3 of Port F as outputs (connected to led's)*/
  PTFDD = 0x0f;
  LED1 = OFF;
  LED2 = OFF;
  LED3 = OFF;
  LED4 = OFF;
  LED5 = OFF;


  /*Initialize timer TPM1 channel, assumes not touched since reset!*/
  TPM1SC_CLKSA = 1;/*Select BUS clock*/
  TPM1SC_CLKSB = 0;
  TPM1SC_PS = PRESCALAR;/*clock source divided by prescalar*/
  TPM1MOD = MODULUS;/*set Counter modulus*/
  /*configure PWM mode and pulse*/
  TPM1C0SC_MS0B = 1;  /*MS0B=1, MS0A=0; << Edge align PWM*/
  TPM1C0SC_ELS0A = 1; /*Select low as true*/



  TPM1C0V = DUTY25;/*select final divider (duty cycle)*/
  LED4 = ON;


  for(;;) {
    __RESET_WATCHDOG(); /* kicks the dog */
    LED2 = SW2;


    if(SW3==DOWN){
      /*Switch pressed*/
      if(SW4==DOWN){/**/
        TPM1C0V = DUTY75;/**/
        LED3 = ON;/**/
        LED4 = OFF;/**/
      }else{
       TPM1C0V = DUTY25;/**/
        LED3 = OFF;/**/
        LED4 = ON;/**/
      }
    }


  } /* loop forever */
}
```

Invoke the debugger

- In CodeWarrior, press the Debug button ( ). This compiles the program and produces an executable file, which invokes the debugger. The debugger erases the FLASH and downloads the code to the device.
- In the debugger, press the RUN button ( ); the application will run.
- LED3 will be illuminated and LED5 (or buzzer on evaluation board) will be pulsed.
- Pressing SW3 whilst holding down SW4 will turn off LED3, light LED4, and change the duty cycle of LED5 (buzzer).
- Pressing SW3 will revert to LED3 illuminated, and the duty cycle of LED5 (buzzer) will revert back to the original duty cycle.
- LED2 will still function according to the state of SW2.

Add interrupt on SW1

CodeWarrior supports several ways of incorporating interrupts. Refer to the Frequently Asked Questions section for a discussion of other methods.

As the vector for the keyboard interrupt is 22, add a macro to define Vkeyboard in the header file M68DEMO908GB60.h:

```
#define Vkeyboard 22
```

The method chosen is the interrupt keyword with vector number. This method has the advantage that the interrupt routine is self-declaring, and only one file is involved in its declaration:

```
interrupt Vkeyboard void intSW1(){
    LED1 = SW4;
    KBISC_KBACK = 1;/*acknowledge interrupt*/
}
```

The code to initialize the interrupts is:

```
  KBIPE_KBIPE4 = 1;
  KBISC_KBIE = 1;
```

Add this code to the main program:

```
#include <hidef.h> /* for EnableInterrupts macro */
#include <MC9S08GB60.h> /* include peripheral declarations */
#include "M68DEMO908GB60.h"

#define PRESCALAR 7
#define MODULUS 32768
#define DUTY75 (MODULUS-(MODULUS/4))
#define DUTY25 (MODULUS/4)


interrupt Vkeyboard void intSW1(){
    LED1 = SW4;
    KBISC_KBACK = 1;/*acknowledge interrupt*/
}


void main(void) {
  EnableInterrupts; /* enable interrupts */
  /* include your code here */

  PTADD = 0;    //initialize as input (Data Direction Register)
  PTAPE = 0xf0; //Pullups on upper 4 bits

  /*initialize bits 0-3 of Port F as outputs (connected to led's)*/
  PTFDD = 0x0f;
  LED1 = OFF;
  LED2 = OFF;
  LED3 = OFF;
  LED4 = OFF;
  LED5 = OFF;

  /*Initialize timer TPM1 channel, assumes not touched since reset!*/
  TPM1SC_CLKSA = 1;/*Select BUS clock*/
  TPM1SC_CLKSB = 0;
  TPM1SC_PS = PRESCALAR;/*clock source divided by prescalar*/
  TPM1MOD = MODULUS;/*set Counter modulus*/
  /*configure PWM mode and pulse*/
  TPM1C0SC_MS0B = 1;  /*MS0B=1, MS0A=0; << Edge align PWM*/
  TPM1C0SC_ELS0A = 1; /*Select low as true*/

  TPM1C0V = DUTY25;/*select final divider (duty cycle)*/
  LED4 = ON;


  KBIPE_KBIPE4 = 1;
  KBISC_KBIE = 1;


  for(;;) {
    __RESET_WATCHDOG(); /* kicks the dog */
    LED2 = SW2;

    if(SW3==DOWN){
      /*Switch pressed*/
      if(SW4==DOWN){/**/
        TPM1C0V = DUTY75;/**/
        LED3 = ON;/**/
        LED4 = OFF;/**/
      }else{
       TPM1C0V = DUTY25;/**/
        LED3 = OFF;/**/
        LED4 = ON;/**/
      }
    }

  } /* loop forever */
}
```

NOTE: *Do not forget to add the #define statement to the header file M68DEMO908GB60.h.*

Invoke the debugger

- In CodeWarrior, press the Debug button ( ). This compiles the program and produces an executable file, which invokes the debugger. The debugger erases the FLASH and downloads the code to the device.
- In the debugger, press RUN ( ); the application will run.
- Pressing SW1 will cause an interrupt, which will read SW4 and set LED1 accordingly.
- LED2, LED3, LED4 and LED5 (buzzer) will still function as before.

Set clock frequency

CAUTION: *An important consideration in setting the FLL is that, if the HCS08 serial monitor is being used for debugging, then changing the processor speed from that set by the monitor will cause CodeWarrior to lose control of the device, unless the SCI speed is also adjusted to take account of the change in processing speed. BDM operation is unaffected by changes in processor speed, as it either has a separate fixed frequency clock (as MC9S08GB60), or the BDM pod has the ability to re synchronize in the event of the device changing frequency.*

The Internal Clock Generator (ICG) used within the MC9S08GB60 has an FLL that allows a frequency higher than the reference source (for example, crystal or internal oscillator) to be generated. This enables the processor frequency to be optimized in terms of power consumption and performance.

The demo board includes a 32.768 kHz crystal, while the evaluation board has a 4 MHz crystal on board. The crystals allow the device to operate at a maximum frequency of 18.87 MHz for the demo board (32.768 kHz) and 20 MHz for the evaluation board (4 MHz).

The formula for calculating the bus frequency is:

$$\textbf{Bus frequency} = ((f_{IRG} \div 7) \times P \times N \div R) \div 2$$

Where

- P = 1 or 64 (high or low frequency range)
- N[0:7] = {4, 6, 8, 10, 12, 14, 16, or 18}
- R[0:7] = 1, 2, 4, 8, 16, 32, 64, or 128

Code to configure the clock in C would be:

```
  /*configure Internal Clock Generator [ICG]*/
  /*MFD[]={4,6,8,10,12,14,16,18}*/
  ICGC2_MFD = 7;            /*32KHz crystal, demo board.
                              For 4MHz crystal (eval board):
                                   ICGC2_MFD = 3
                            */

  ICGC2_RFD = 0;            /* RFD[]={1,2,4,8,16,32,64,128}*/

  ICGC1 = 0b00111000;       /*32KHz crystal, demo board.
                              For 4MHz crystal (eval board):
                                    ICGC1 = 0b01111000;
                            */

  while((ICGS1_LOCK==0)||(ICGS1_ERCS==0)){
    /*Ensure COP doesn't reset device whilst waiting for clock lock*/
    __RESET_WATCHDOG(); /* kicks the dog */
  }
  ICGC2_LOCRE = 1; /*enable reset if clock fails*/
```

The “while” loop ensures that the external clock has been selected (ICGS1_ERCS) and that the FLL has locked to the desired frequency (ICGS1_LOCK).

The final program is:

```
#include <hidef.h> /* for EnableInterrupts macro */
#include <MC9S08GB60.h> /* include peripheral declarations */
#include "M68DEMO908GB60.h"

#define PRESCALAR 7
#define MODULUS 32768
#define DUTY75 (MODULUS-(MODULUS/4))
#define DUTY25 (MODULUS/4)

interrupt Vkeyboard void intSW1(){
    LED1 = SW4;
    KBISC_KBACK = 1;/*acknowledge interrupt*/
}

void main(void) {
  EnableInterrupts; /* enable interrupts */
  /* include your code here */

  PTADD = 0;    //initialize as input (Data Direction Register)
  PTAPE = 0xf0; //Pullups on upper 4 bits

  /*initialize bits 0-3 of Port F as outputs (connected to led's)*/
  PTFDD = 0x0f;
  LED1 = OFF;
  LED2 = OFF;
  LED3 = OFF;
  LED4 = OFF;
  LED5 = OFF;

  /*Initialize timer TPM1 channel, assumes not touched since reset!*/
  TPM1SC_CLKSA = 1;/*Select BUS clock*/
  TPM1SC_CLKSB = 0;
  TPM1SC_PS = PRESCALAR;/*clock source divided by prescalar*/
  TPM1MOD = MODULUS;/*set Counter modulus*/
  /*configure PWM mode and pulse*/
  TPM1C0SC_MS0B = 1;  /*MS0B=1, MS0A=0; << Edge align PWM*/
  TPM1C0SC_ELS0A = 1; /*Select low as true*/

  TPM1C0V = DUTY25;/*select final divider (duty cycle)*/
  LED4 = ON;

  KBIPE_KBIPE4 = 1;
  KBISC_KBIE = 1;
  /*configure Internal Clock Generator [ICG]*/
  /*MFD[]={4,6,8,10,12,14,16,18}*/
  ICGC2_MFD = 7;             /*32KHz crystal, demo board.
                               For 4MHz crystal (eval board):
                                    ICGC2_MFD = 3
                             */

  ICGC2_RFD = 0;             /* RFD[]={1,2,4,8,16,32,64,128}*/

  ICGC1 = 0b00111000;        /*32KHz crystal, demo board.
                               For 4MHz crystal (eval board):
                                     ICGC1 = 0b01111000;
                             */

  while((ICGS1_LOCK==0)||(ICGS1_ERCS==0)){
    /*Ensure COP doesn't reset device whilst waiting for clock lock*/
    __RESET_WATCHDOG(); /* kicks the dog */
  }
  ICGC2_LOCRE = 1; /*enable reset if clock fails*/


  for(;;) {
    __RESET_WATCHDOG(); /* kicks the dog */
    LED2 = SW2;

    if(SW3==DOWN){
      /*Switch pressed*/
      if(SW4==DOWN){/**/
        TPM1C0V = DUTY75;/**/
        LED3 = ON;/**/
        LED4 = OFF;/**/
      }else{
       TPM1C0V = DUTY25;/**/
        LED3 = OFF;/**/
        LED4 = ON;/**/
      }
    }

  } /* loop forever */
}
```

NOTE: *It is important to set the RANGE, REFS and CLKS in the ICGC1 register together as a single byte store (for example, STA or MOV instruction), and not by read-modify-write instructions (BSET or BCLR), as attempting to set these bits individually may result in the clock mode being locked after the first write to the ICGC1 register.*

For more information on setting system clock, refer to the relevant data sheet and to Application Note “AN2494 — Configuring the System and Peripheral Clocks in the MC9S08GB/GT “.

Application complete

The program will operate in exactly the same manner as in the previous section. The only noticeable difference will be the frequency of LED5 (buzzer in the case of the evaluation board), as the bus speed of the device has been set to the maximum using the on-board crystals.

Metrowerks HCS08 CodeWarrior Development Tools

Metrowerks CodeWarrior

Additional HC(S)08 help included with CodeWarrior

After CodeWarrior has been installed, assuming a “typical” or a “full” installation was performed), there will be some HC(S)08 specific documentation that will assist in fine-tuning your application.

The default location of Metrowerks CodeWarrior is:

- C:\Program Files\Metrowerks\CodeWarrior CW08_V3.0

Directories and files of interest:

- Release_Notes\HC08\CW_Tools\HC08
  - This directory contains release notes for various sections of CodeWarrior, for example, linker, compiler, etc.
- CodeWarrior Help
  - This directory contains the generic CodeWarrior and is target independent.
- CodeWarrior Manuals
  - In this directory, the documentation is available as either on-screen help or standalone pdfs and are held within the relevant sub-directories.
    - hc08_manuals.pdf
    - HC08_Processor_Expert_User_Guide.pdf
    - Manual_Assembler_HC08.pdf
    - Manual_Compiler_HC08.pdf
    - Manual_Engine_HC08.pdf
    - Manual_ICD_HCS08.pdf
    - Manual_Mon08.pdf
    - Manual_True-Time_Simulator_HC08.pdf

Metrowerks

Metrowerks produces an integrated development environment (IDE) for the HC08 and HCS08. To obtain the best from the HCS08, the latest version should be used. The HC08 version will generate code that will work on an HC08, but it will not use any of the additional addressing modes on a couple of instructions; this can have a significant impact on both code size and performance.

Metrowerks offers a free version of the Metrowerks HCS08 IDE, which is initially limited to 1K of code generated from C. Upon registering the compiler, a license will be provided which will allow the compiler to generate up to 4K of code.

Metrowerks have other licenses which will allow 32K or 64K of code, upon payment of the relevant license fee.

More information about the Metrowerks HC(S)08 CodeWarrior development Studio is available from the following website.

http://www.metrowerks.com/MW/Develop/Embedded/HC08/Default.htm

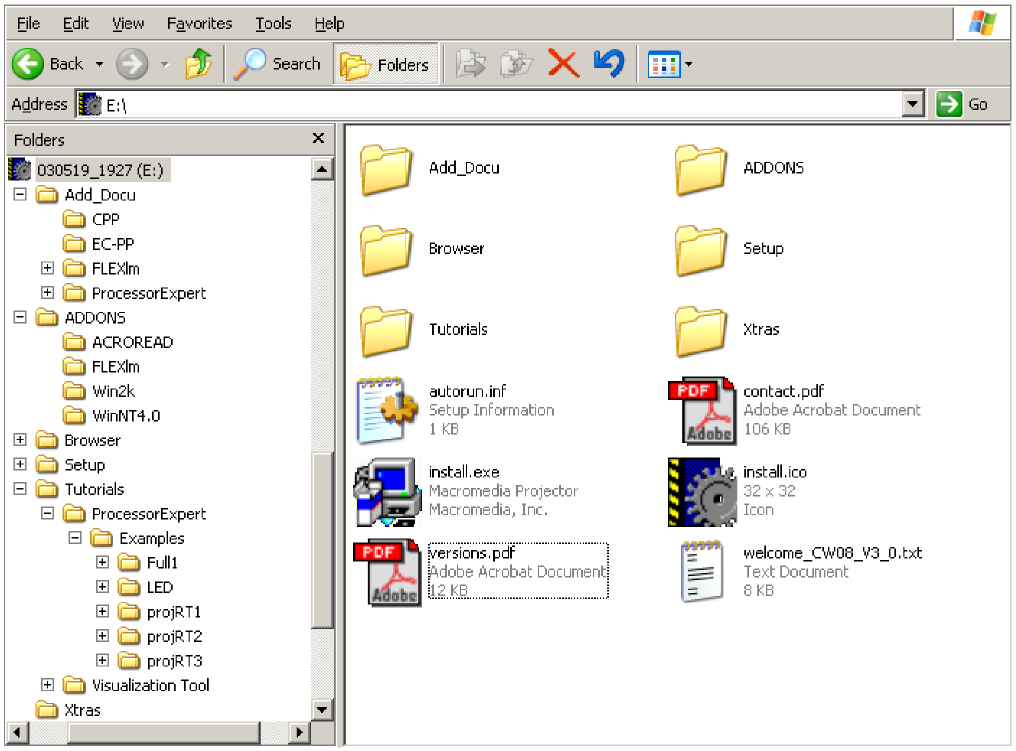
Metrowerks CodeWarrior for HC(S)08

CodeWarrior is an integrated development environment that provides a graphical user interface (GUI) to code development. Code is developed as a project where all the files, configuration information, and debugging information (for example, hardware) required to generate/debug a program are stored.

CodeWarrior includes a project manager (Project window) which lists all the files required to compile the code and invoke the various development activities such as editing, compiling, or running and debugging the application using the simulator or actual hardware.

What's on the CD

Opening the CD in Windows Explorer will result in something like the following:



```
030519_1927 (E:)
  Add_Docu
    CPP
    EC-PP
    FLEXlm
    ProcessorExpert
  ADDONS
    ACROREAD
    FLEXlm
    Win2k
    WinNT4.0
  Browser
  Setup
  Tutorials
    ProcessorExpert
      Examples
        Full1
        LED
        projRT1
        projRT2
        projRT3
    Visualization Tool
  Xtras
```

Add_Docu	ADDONS
Browser	Setup
Tutorials	Xtras
autorun.inf Setup Information 1 KB	contact.pdf Adobe Acrobat Document 106 KB
install.exe Macromedia Projector Macromedia, Inc.	install.ico 32 x 32 Icon
versions.pdf Adobe Acrobat Document 12 KB	welcome_CW08_V3_0.txt Text Document 8 KB

System requirements for CodeWarrior

Metrowerks CodeWarrior is compatible with the following Microsoft products.

- Windows 98 (including SE)
- Windows ME
- Windows XP
- Windows NT 4
- Windows 2000

NOTE: *Some versions of Windows may require installation of appropriate patches or service packs.*

NOTE: *There may be additional restrictions on supported operating systems, due to debugging support; for example, USB Multilink requires Windows 2000 or XP, and is not supported with other versions of Windows.*

Installing CodeWarrior

On inserting the CD into a Windows PC, the CodeWarrior installer will normally start automatically. (If this is not the case, locate the install.exe file on the CD and invoke it). Follow the on-screen instructions and refer to the booklet supplied with the CD for more help. (For the web download version, a “quick start” document is available on the web.)

CodeWarrior license

By default, Metrowerks installs a code size limited version of CodeWarrior, which is limited to generating runable code of up to 1K byte in size. By registering, you can obtain a license file from CodeWarrior Metrowerks by E-mail that will remove the code size limit on the assembler and linker; the C compiler code generator limit will be increased from 1K to 4K. (Registration is free; a valid E-mail address is required in order to send the license details.)

A 30-day evaluation license to the Standard or Professional Edition with an unlimited code size can be requested during the registration process; refer to the welcome text file included on the CD for details of how to request this.

NOTE: *A registered version of CodeWarrior will have a 4K code size limit for C and an unlimited code size assembler.*

CodeWarrior updates

The boards and devices supported by Metrowerks CodeWarrior will be those available when released. There may be patches available for CodeWarrior to support newer boards and/or devices; check Metrowerks web support if the board or device is not supported by your version of CodeWarrior:

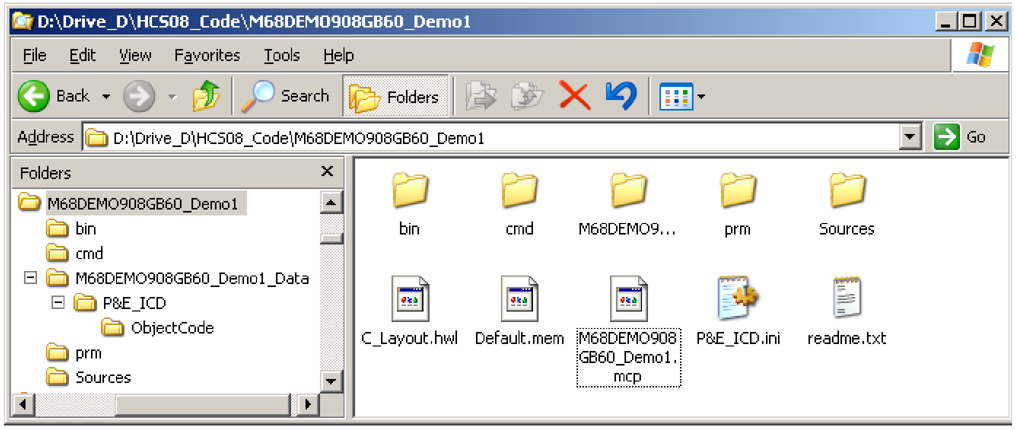
http://www.metrowerks.com/MW/download/default.asp

CodeWarrior “projects”

A “project” is simply a file (for example, M68DEMO908GB60_Demo1.mcp) that contains all the information required to compile and debug an application. Information in the project file typically includes:

- Directory location for sources, binaries, and other files
- Compiler settings for each source file
- Debugging details

CodeWarrior normally places the files within the directory of the project file or subdirectories; for example:



CodeWarrior C and HCS08

This section explains how the HCS08 devices are defined within CodeWarrior C, and how they are used.

In order to be able to write C code, for HCS08 devices, that utilizes the hardware, it is necessary to define the registers of the modules and their absolute addresses. Metrowerks provides the register definitions and address space mapping for each device within two files: the device header file and the device definition file.

The device header file (MC9S08GB60.h) contains data declarations and definitions used to reference the device registers. Registers are allocated and mapped into the device's address space in the device definition file (MC9S08GB60.c), using the definitions from the header file.

A program normally consists of the device definition/mapping file (MC9S08GB60.c), the user code (for example, main.c), and a startup file (startup.c), which initializes the runtime environment. The startup code, device and header files are normally included automatically within the project by the "CodeWarrior project wizard", along with a default main program (main.c).

To access peripherals in a device, ensure that the device declaration file (MC9S08GB60.h) is included within the source file; for example:

```
#include <MC9S08GB60.h>
int mycode(){
}
```

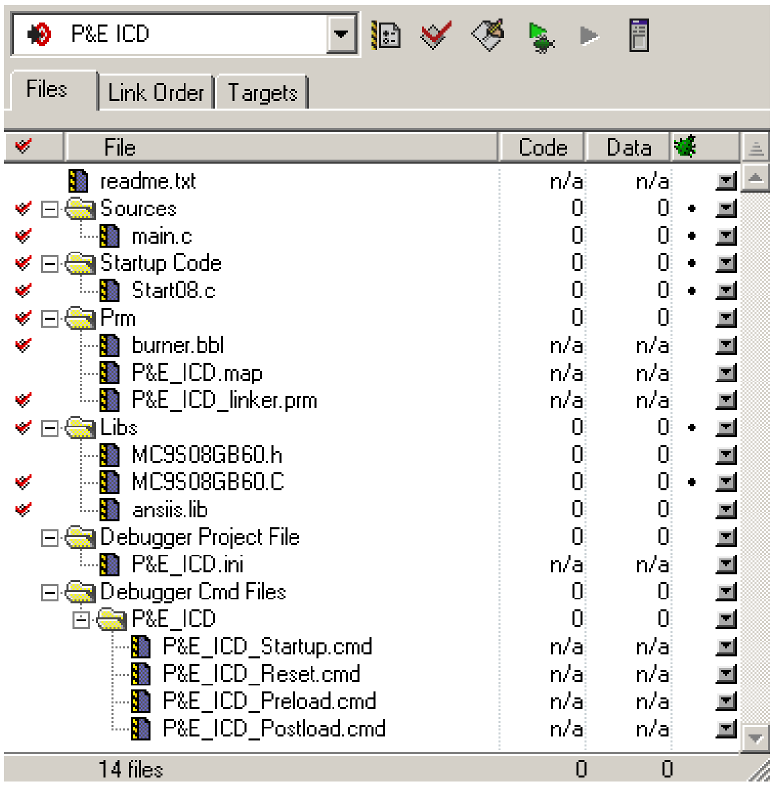
Data types

CodeWarrior C for the HCS08 supports all the normal C types. Also, the types byte (1 byte), word (2 bytes), dword (4 bytes), and dlong (8 bytes) are defined unsigned as follows:

```
/* Types definition */
typedef unsigned char byte;
typedef unsigned int word;
typedef unsigned long dword;
typedef unsigned long dlong[2];
```

What a project consists of

A newly created project looks like:



✓	File	Code	Data	
	readme.txt	n/a	n/a	
✓	Sources	0	0	•
✓	main.c	0	0	•
✓	Startup Code	0	0	•
✓	Start08.c	0	0	•
✓	Prm	0	0	
✓	burner.bbl	n/a	n/a	
	P&E_ICD.map	n/a	n/a	
✓	P&E_ICD_linker.prm	n/a	n/a	
✓	Libs	0	0	•
	MC9S08GB60.h	0	0	
✓	MC9S08GB60.C	0	0	•
✓	ansiis.lib	0	0	
	Debugger Project File	0	0	
	P&E_ICD.ini	n/a	n/a	
	Debugger Cmd Files	0	0	
	P&E_ICD	0	0	
	P&E_ICD_Startup.cmd	n/a	n/a	
	P&E_ICD_Reset.cmd	n/a	n/a	
	P&E_ICD_Preload.cmd	n/a	n/a	
	P&E_ICD_Postload.cmd	n/a	n/a	
	14 files	0	0	

with the following files and directories created automatically:

- readme.txt
  - Initially contains a brief overview of the project structure, details of on-line help, and how to contact Metrowerks
- Sources/
  - Contains the users source code, sample main.c provided by wizard with project creation
- Startup Code/
  - Start.c - C/C++ startup code which initializes the C library and invokes the user code (main function)
- Prm/
  - burner.bbl - details of how to generate the required S-Record for the debugger
  - *.prm - details of how to link code/data segments
  - *.map - generated by the linker
- Libs/
  - Required library files (ANSI library)
  - Device header and device file
- Debugger Project File/
  - Contains an *.ini file for the debugger — essentially a project file for the debugger
- Debugger Cmd Files/
  - Contains sub-folders for each target connection method, along with command files

Using CodeWarrior device definitions

This section describes the method used by Metrowerks CodeWarrior in defining device registers, the mapping of these registers to memory, and how to use this information within a program.

All examples within this section refer to the MC9S08GB60. Substitute the appropriate device as required.

The names given to the device registers and bit names within the registers are defined by the device definition files. The two files associated with the MC9S08GB60 are:

- MC9S08GB60.c <- device file
- MC9S08GB60.h <- device definition/mapping file

To fully understand programming a particular device, it is essential to have the correct data sheet for the device, for example, the MC9S08GB60 Data Sheet (MC9S08GB60/D).

Device file (MC9S08GB60.c)

This file defines ALL registers within the device. All registers are named as per the register names in the relevant data sheet:

```
volatile <register>STR _<register>;
```

Where <register> is the name (in capital letters) of the register, as defined in the relevant data book (for example, MC9S08GB/D). All registers are defined as structure types, where the structure name is the same as the register name with STR appended. For example:

```
volatile KBIPESTR _KBIPE;
```

_KBIPE is the actual register. The structures are defined in the header file.

NOTE: *For simplicity, the following examples will use the macro definitions of the registers.*

Header file (MC9S08GB60.h)

This file makes available the register definitions for any file that requires access to the registers. It also maps the registers into the device's memory map.

The header file contains macros that allow the register to be referenced without the underscore. For example:

```
#define KBIPE _KBIPE
```

All registers can be referred to by their real definition, in this case "_KBIPE" or via a macro, in this case "KBIPE".

Register and bit definitions

The header files supplied with CodeWarrior facilitate access to register bits through structures and do not include mask-based access to the register bits.

The following example shows the definition of the DBGC register utilizing a structure.

```
/*** DBGC - Debug Control Register ***/
typedef union {
  byte Byte;
  struct {
    byte RWBEN   :1;     /* Enable R/W for Comparator B */
    byte RWB     :1;     /* R/W Comparison Value for Comparator B */
    byte RWAEN   :1;     /* Enable R/W for Comparator A */
    byte RWA     :1;     /* R/W Comparison Value for Comparator A */
    byte BRKEN   :1;     /*  Break Enable */
    byte TAG     :1;     /* Tag/Force Select */
    byte ARM     :1;     /* Arm Control */
    byte DBGEN   :1;     /* Debug Module Enable */
  } Bits;
} DBGCSTR;
extern volatile DBGCSTR _DBGC @0x00001816;
```

The ":1" is used to indicate that a single bit is required and CodeWarrior C compiler will pack together into a single byte.

To access RWB would require the following code.

```
DBGC.Bits.RWB = 1;       /*Set RWB bit of DBGC */
DBGC.Bits.RWB = 0;       /*Clear RWB bit of DBGC */
```

To access RWB using the Metrowerks predefined macros would require the following code.

```
DBGC_RWB = 1;       /*Set RWB bit of DBGC */
DBGC_RWB = 0;       /*Clear RWB bit of DBGC */
```

The Metrowerks CodeWarrior C Compiler will generate bit set/clear instructions for page 0 registers and memory; otherwise, it will generate bit mask operations (I =, & =) for other addresses. Using bit structures with HCS08 for I/O registers residing in page 0 is very efficient, requiring a single instruction for set, clear and test/branch.

How device registers and bits are used

To use a register, simply use its name as defined within the data sheet; for example:

- TPM2C0SC

To reference a bit within a register, concatenate the register name and the bit name with an underscore between. For example:

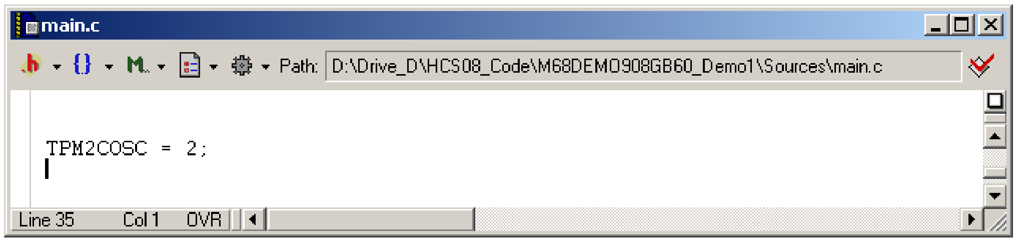
- TPM2C0SC_MS0B

NOTE: *Register and bit names MUST be in capital letters.*

How device registers are defined and used

In order to be able to program a device, it is necessary to understand the relationship between the device definition/mapping file and the device header file, and the register definitions contained within these files.

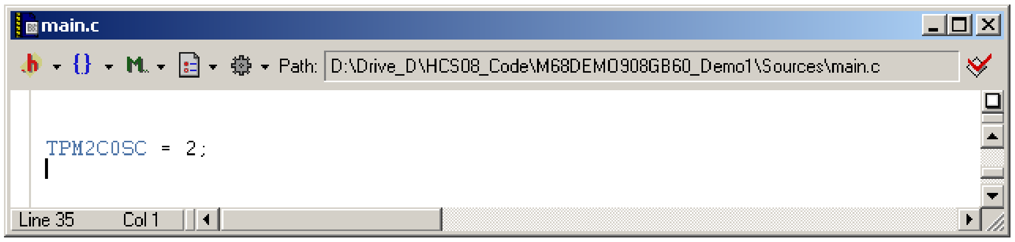
For example, the register TPM2C0SC:


main.c
Path: D:\Drive_D\HCS08_Code\M68DEMO908GB60_Demo1\Sources\main.c
Line 35
Col 1
OVR

```
TPM2COSC = 2;
```

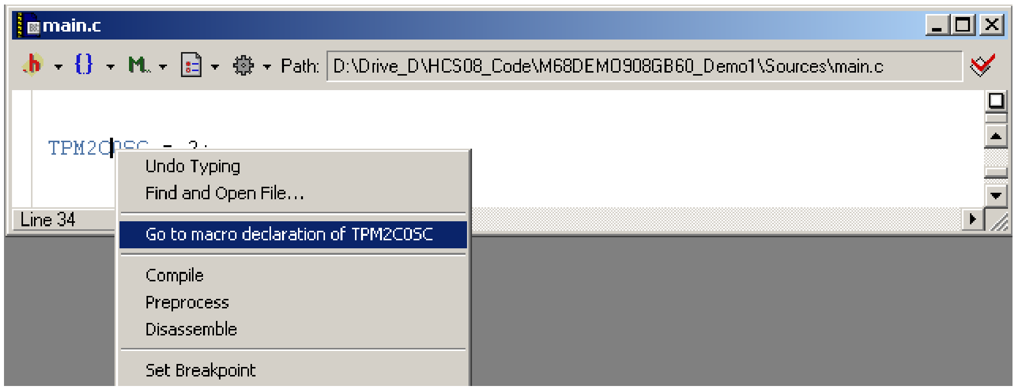
As a letter “O” was typed instead of a zero, the variable is undefined and so appears in black lettering.

After correcting, the register name appears in light blue:

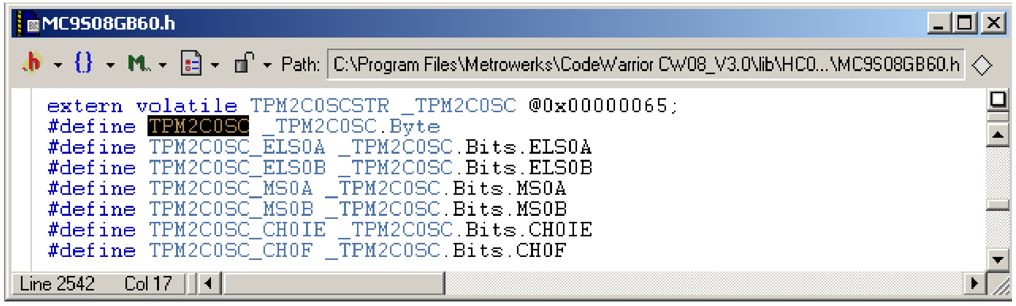

main.c
Path: D:\Drive_D\HCS08_Code\M68DEMO908GB60_Demo1\Sources\main.c
Line 35
Col 1

```
TPM2C0SC = 2;
```

To determine the definition of the variable TPM2C0SC, place the cursor over the variable name and right-click the mouse. A popup will appear, which will include an option to go to the variable declaration “Go to macro declaration of TPM2C0SC”:



This will bring up a window showing the section of code that defines the variable:



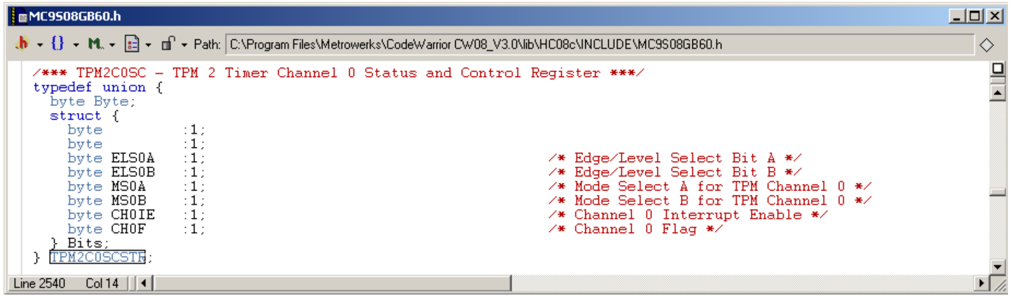
MC9S08GB60.h

Path: C:\Program Files\Metrowerks\CodeWarrior CW08_V3.0\lib\HC0...\MC9S08GB60.h

```
extern volatile TPM2C0SCSTR _TPM2C0SC @0x00000065;
#define TPM2C0SC _TPM2C0SC.Byte
#define TPM2C0SC_ELS0A _TPM2C0SC.Bits.ELS0A
#define TPM2C0SC_ELS0B _TPM2C0SC.Bits.ELS0B
#define TPM2C0SC_MS0A _TPM2C0SC.Bits.MS0A
#define TPM2C0SC_MS0B _TPM2C0SC.Bits.MS0B
#define TPM2C0SC_CH0IE _TPM2C0SC.Bits.CH0IE
#define TPM2C0SC_CH0F _TPM2C0SC.Bits.CH0F
```

Line 2542 Col 17

As can be seen in the example above, the variable is actually a macro definition to a structure element “Byte” of the variable _TPM2C0SC. The variable _TPM2C0SC is, in fact, defined as being a structure of type TPMC0SCSTR and at absolute address $0065, as shown in the figure above. The definition of structure TPMC0SCSTR is:



MC9S08GB60.h

Path: C:\Program Files\Metrowerks\CodeWarrior CW08_V3.0\lib\HC08c\INCLUDE\MC9S08GB60.h

```
/*** TPM2C0SC - TPM 2 Timer Channel 0 Status and Control Register ***/
typedef union {
  byte Byte;
  struct {
    byte        :1;
    byte        :1;
    byte ELS0A  :1;                                    /* Edge/Level Select Bit A */
    byte ELS0B  :1;                                    /* Edge/Level Select Bit B */
    byte MS0A   :1;                                    /* Mode Select A for TPM Channel 0 */
    byte MS0B   :1;                                    /* Mode Select B for TPM Channel 0 */
    byte CH0IE  :1;                                    /* Channel 0 Interrupt Enable */
    byte CH0F   :1;                                    /* Channel 0 Flag */
  } Bits;
} TPM2C0SCSTR;
```

Line 2540 Col 14

Summary: From the example of TPM2C0SC, it can be seen that the way registers are defined within CodeWarrior is:

A register, as defined in a data sheet (for example, TPM2C0SC), is a macro that refers to a byte within a structure, and is mapped to the relevant address for the device in question.

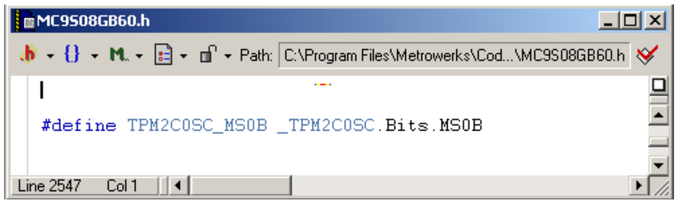
Metrowerks CodeWarrior header files for HCS08 use bit structures as opposed to masks. All code will utilize the CodeWarrior structures.

How device register bit(s) are defined / used

The method used for defining register bits follows a similar method as used for the actual registers, except that, in the case of register bits, it is necessary to specify both the register name and the bit name as defined in the data sheet, for example, MS0B and TPM2C0SC both need to be specified. The way in which the bit name and register name are used is to concatenate their names together with an underscore between; for example:

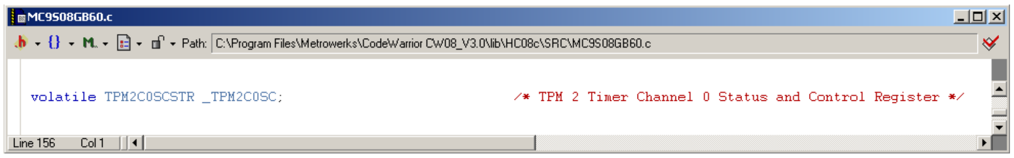
- TPM2C0SC_MS0B

The definition of the bit can be found by right-clicking on the appropriate variables. For example:



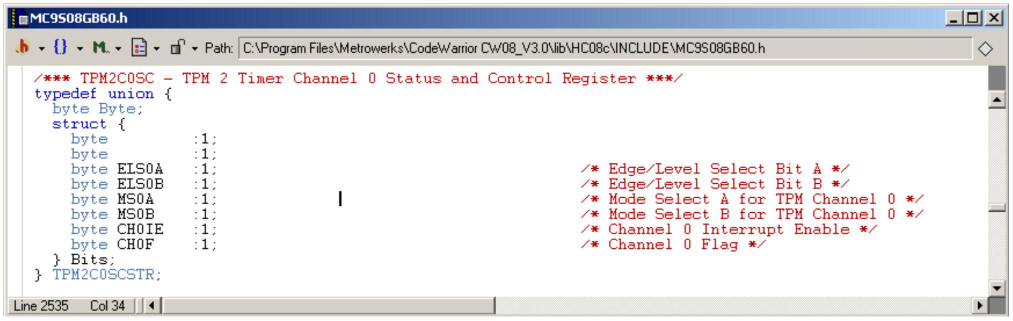
```
#define TPM2C0SC_MS0B _TPM2C0SC.Bits.MS0B
```

The definition of _TPM2C0SC:


MC9S08GB60.c
Path: C:\Program Files\Metrowerks\CodeWarrior CW08_V3.0\lib\HC08c\SRC\MC9S08GB60.c
Line 156 Col 1

```
volatile TPM2C0SCSTR _TPM2C0SC;                    /* TPM 2 Timer Channel 0 Status and Control Register */
```

Definition of type TPM2C0SCSTR is a structure:


MC9S08GB60.h
Path: C:\Program Files\Metrowerks\CodeWarrior CW08_V3.0\lib\HC08c\INCLUDE\MC9S08GB60.h
Line 2535 Col 34

```
/*** TPM2C0SC - TPM 2 Timer Channel 0 Status and Control Register ***/
typedef union {
  byte Byte;
  struct {
    byte         :1;
    byte         :1;
    byte ELS0A   :1;                                  /* Edge/Level Select Bit A */
    byte ELS0B   :1;                                  /* Edge/Level Select Bit B */
    byte MS0A    :1;                                  /* Mode Select A for TPM Channel 0 */
    byte MS0B    :1;                                  /* Mode Select B for TPM Channel 0 */
    byte CH0IE   :1;                                  /* Channel 0 Interrupt Enable */
    byte CH0F    :1;                                  /* Channel 0 Flag */
  } Bits;
} TPM2C0SCSTR;
```

Now MS0B is one bit of a byte in the structure Bits of the structure TPM2C0SCTR, so we can use:

- _TPM2C0SC.Bits.MS0B

or

- TPM2C0SC.Bits.MS0B

To reference it, alternatively, we can use one of the predefined macros:

- TPM2C0SC_MS0B

Both examples refer to the Mode select B of TPM channel 0.

Summary: A register bit is defined by a macro as being its register and bit name concatenated together with an underscore between the register and bit names.

Register names used with multiple peripherals

Some devices have multiple peripherals; for example, the MC9S08GB60 has two SCIs. It may be necessary to check the naming of the peripherals in the device and header files but, in general, they should be defined as per the data sheet.

For example, the MC9S08GB60 has multiple timers (timer 1 and timer 2), which have multiple channels (three on timer 1 and five on timer 2). As an example, the status channel for a channel is defined as:

TPM**x**C**n**SC

– CHnF - flag
– CHnIE - interrupt enable
– MSnB - mode select B
– MSnA - mode select A
– ELSnB:ELSnA - Edge/Level select bits

Where

– x = the timer
– n = the channel

Therefore, to reference the flag bit of channel 2 on timer 1, the macro defined in the header file is:

TPM1C2SC_CH2F

Frequently Asked Questions

This section identifies the main issues that a newcomer to CodeWarrior may experience, and explains how to proceed.

Where can I get the most up to date documentation

The most up to date documentation for Codewarrior is available on Metrowerks web site. For the HC08 it can be found at:

http://www.metrowerks.com/MW/Support/dev_resources/HC08.htm

Device and/or target isn't supported by Metrowerks

Check Metrowerks for availability of patches to support device and/or target.

http://www.metrowerks.com/MW/download/default.asp

In the “Updates and Patches” section, select “Codewarrior for Motorola HC08”, press “select”, and a list of available patches will appear. Download the appropriate patch and install (a reboot of the PC will be required).

USB BDM doesn't work with Metrowerks HC08 Codewarrior v3.0

Codewarrior version 3.0 was shipped before P&E USB BDM was available. To add USB BDM functionality it is necessary to install a service pack. The service pack is available from Metrowerks site at:

ftp://ftp.metrowerks.com/pub/updates/
CWHC08/HC08V3_0_USB_MULTILINK_SP.exe

Tips on using a parallel BDM pod?

General suggestions for successfully using a Multilink BDM connector.

- Upgrade to the latest version of CodeWarrior
- Ensure that the BDM pod is using the latest firmware (http://www.pemicro.com)
- Ensure that the parallel port BDM hardware is at the latest revision. With the exception of Rev.A, all Multilinks can be upgraded to latest spec (currently Rev.D) (http://www.pemicro.com)
- Ensure that the parallel port is configured as a standard port in computer’s BIOS. The BIOS settings for the Parallel port should be SPP, Normal, Standard, Output Only, Unidirectional or AT. Try to avoid ECP, EPP or PS/2 Bidirectional
- Limit the BDM cable length between MultiLink and target
- Do not protect the FLASH during code development and debugging

P&E parallel Multilink BDM & laptop

Some laptops ship with a 3v parallel port and may not work reliably with the P&E parallel Multilink BDM. To overcome this the Multilink should be powered by an external 5v supply. Connect an external 5v center negative power supply to the optional power jack of the P&E Multilink BDM.

Monitor mode is not working?

There are several ways this can occur:

- Monitor erased
  - reprogram
- Clock speed is incorrectly assumed within monitor
  - reprogram device with different clock setup
  - change crystal to 32.768 kHz or 4 MHz, according to version of monitor in device

Other than changing the crystal, a BDM will be required to reprogram the monitor code or alter the device's clock frequency used by the monitor.

How do I reprogram the HCS08 monitor?

If the HCS08 monitor is erased or corrupted, it will be necessary to download the HCS08 monitor code to the device utilizing a BDM; there is no other way.

Refer to application note "AN2140/D — Serial Monitor for MC9S08GB/GT" for a description of the monitor and the Metrowerks project files required for re-programming the monitor.

How can I program small batches of HCS08 devices without tying up a PC?

An HCS08 can be programmed only using the serial monitor mode or BDM. P&E make a BDM pod that can be connected to the PC via usb, parallel or ethernet, which can also be used as a standalone programmer. Refer to P&E's web site for product details and ordering information on the CYCLONE PRO

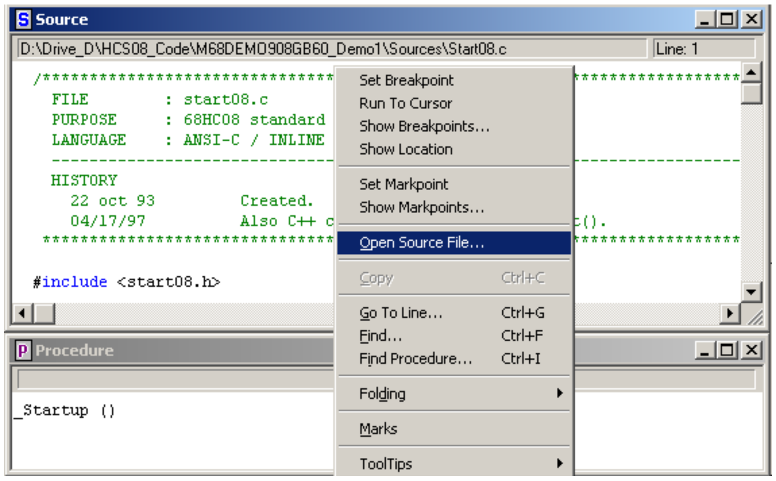
http://www.pemicro.com/products/68hc08/mon08/cyclone_pro/cyclone_pro.html

Code in FLASH only works when BDM is powered?

Disconnect the BDM pod, as it is interfering with the devices normal operation.

Debugger not showing the source code of main.c?

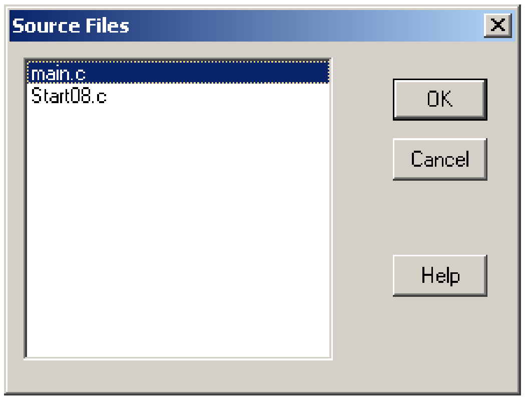
To show the source code for main.c in the debugger, simply right-click over the source code window:



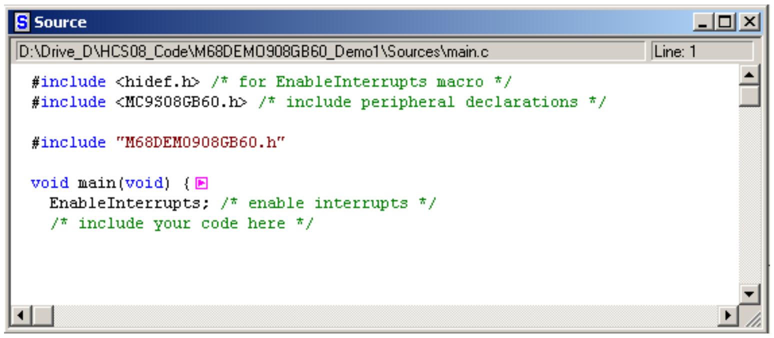
```
D:\Drive_D\HCS08_Code\M68DEMO908GB60_Demo1\Sources\Start08.c        Line: 1

/*********************************
  FILE         : start08.c
  PURPOSE      : 68HC08 standard
  LANGUAGE     : ANSI-C / INLINE
  ----------------------------------
  HISTORY
    22 oct 93          Created.
    04/17/97           Also C++ c
 *********************************

#include <start08.h>
```

```
_Startup ()
```

and select "Open Source File". A dialog box will pop up, allowing selection of the correct source file (for example, main.c):



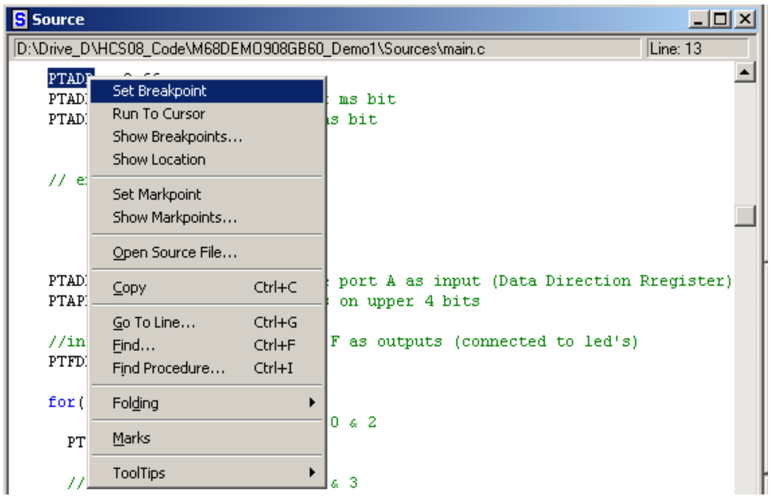
After selecting the correct file, the source window will appear:



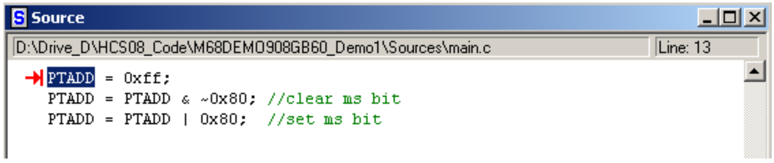
```
D:\Drive_D\HCS08_Code\M68DEMO908GB60_Demo1\Sources\main.c        Line: 1

#include <hidef.h> /* for EnableInterrupts macro */
#include <MC9S08GB60.h> /* include peripheral declarations */

#include "M68DEMO908GB60.h"

void main(void) {
  EnableInterrupts; /* enable interrupts */
  /* include your code here */
```

How do I set a breakpoint in the debugger?

To set a break point whilst in the debugger, select some text where the break point is required, or hold the mouse over the relevant code and press the right mouse button. A popup dialog box will appear; the first option is to set a break point:



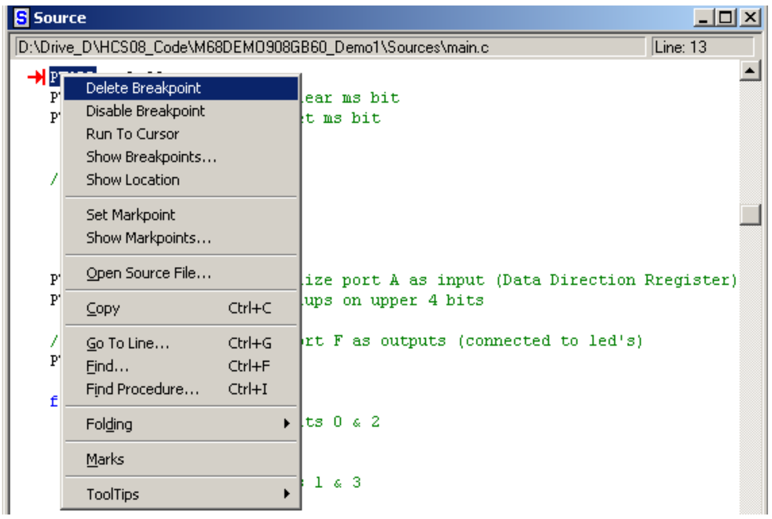
The break point will be shown in the source window with a red arrow:



```
PTADD = 0xff;
PTADD = PTADD & ~0x80; //clear ms bit
PTADD = PTADD | 0x80;  //set ms bit
```

To remove a breakpoint, simply select (or place the mouse over) the break point you wish to remove and right click the mouse.

A popup dialog will appear, the first option being to delete the break point:

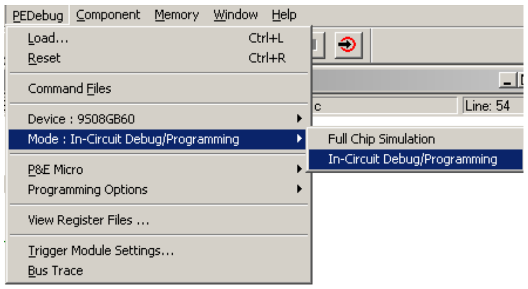


Debugging does not seem to use the hardware/select in-circuit debug

The most likely reason for this is that the debugger is using the simulator or the wrong hardware. A may be caused by the debugger not finding a BDM pod when it started and it defaulted to the software simulator.

Ensure the correct target is set.

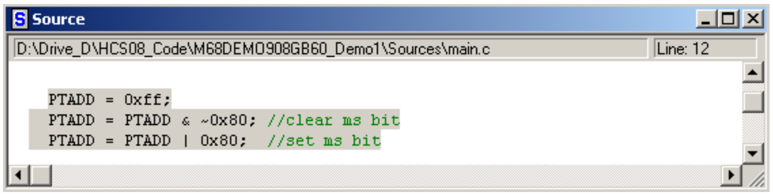
To set in-circuit debug/programming whilst in "Full Chip Simulation" mode, select "In-Circuit Debug/Programming" via the PEDebug pull-down menu:



The debugger will revert to the stage after it was invoked from the project manager and should erase and reprogram the device.

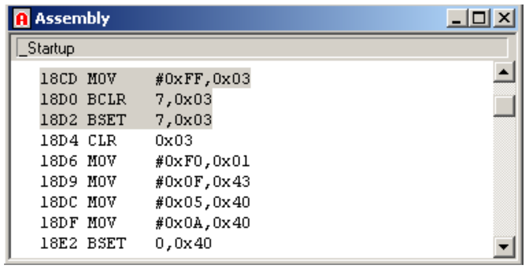
How can I see the assembler code generated for C statements?

When reviewing C code in order to optimize, it is helpful to see exactly what code is generated by the compiler for a C statement. The easiest way to do this is to highlight the relevant C code:



```
PTADD = 0xff;
PTADD = PTADD & ~0x80; //clear ms bit
PTADD = PTADD | 0x80;  //set ms bit
```

Next, drag the C code to the assembler window (hold the mouse down over the selected text, move the cursor over the assembler window, and release) and the assembly language statements for the C code will be highlighted:



```
18CD MOV   #0xFF,0x03
18D0 BCLR  7,0x03
18D2 BSET  7,0x03
18D4 CLR   0x03
18D6 MOV   #0xF0,0x01
18D9 MOV   #0x0F,0x43
18DC MOV   #0x05,0x40
18DF MOV   #0x0A,0x40
18E2 BSET  0,0x40
```

NOTE: *Due to the optimization performed by the compiler, some code may not be highlighted.*

What are all the interrupt vector numbers for the MC9S08GB60?

The following define statements include ALL the interrupt vectors of the MC9S08GB60.

```
#define Vreset      0
#define Vswi        1
#define Virq        2
#define Vlvd        3
#define Vicg        4
#define Vtpm1ch0    5
#define Vtpm1ch1    6
#define Vtpm1ch2    7
#define Vtpm1ovf    8
#define Vtpm2ch0    9
#define Vtpm2ch1   10
#define Vtpm2ch2   11
#define Vtpm2ch3   12
#define Vtpm2ch4   13
#define Vtpm2ovf   14
#define Vspi       15
#define Vsci1err   16
#define Vsci1rx    17
#define Vsci1tx    18
#define Vsci2err   19
#define Vsci2rx    20
#define Vsci2tx    21
#define Vkeyboard  22
#define Vatd       23
#define Viic       24
#define Vrti       25
```

Where are header files located?

Basically, there are two types of header files used within C:

- System - placed within angled brackets (< and >)
- User - placed within double quotes (")

System header files are located within the Metrowerks CodeWarrior directories; for the HCS08, they are located in the directory “CodeWarrior CW08_V3.0\lib\HC08c\include”.

User header files are normally located within a sub-directory of the project directory (for example, Sources or headers). User headers are made available to the user by adding them to the project.

Should I use bit fields or masks for bit manipulation?

Metrowerks CodeWarrior will use bit field instructions (bit set, clear, test/branch) for bit field data or simple mask operations on data within page 0.

For memory other than page 0, the compiler translates bit field operations to bit mask operations.

How does the compiler use page 0?

Most registers are defined within page 0, and the compiler will utilize direct addressing wherever possible when using the peripheral registers. The linker files generated for a device do not use page 0 by default (unless the device only has RAM in page 0). The default will allocate variables in extended memory.

How can I force a variable to reside in page 0?

To instruct the linker to allocate storage for variables in page 0, enclose the declarations within the following #pragma statements in the file where the variable is declared (for example, main.c):

```
#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
/*Page 0 data declarations go here*/
#pragma DATA_SEG DEFAULT
```

For example:

```
#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
byte p0i, p0j;
#pragma DATA_SEG DEFAULT
```

The first #pragma statement instructs the compiler to allocate data from this point on in the short segment (page 0), MY_ZEROPAGE is the default data segment used within the linker file.

The second #pragma statement instructs the compiler to return to the default data section for allocation of subsequent variables.

NOTE: *Segment names assume default project and linker configuration used.*

How do I disable the watchdog?

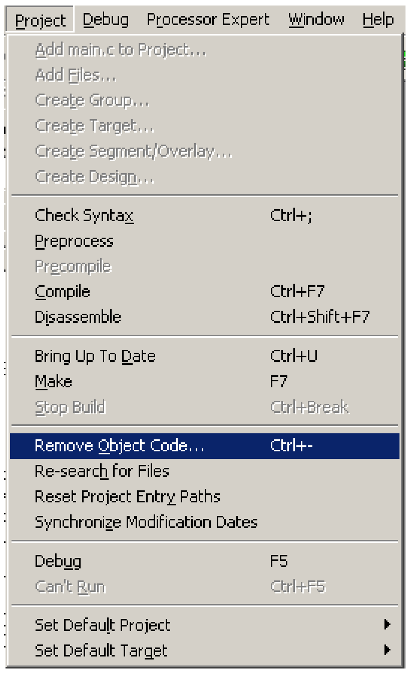
It may be necessary to disable the watchdog whilst debugging code, or an application may not require it.

To disable the watchdog, simply clear the COPE bit in the SOPT register as follows:

```
SOPT_COPE = 0;
```

Problems with variable, structure, or type definition?

Occasionally, CodeWarrior will not have the correct definition for a variable, a structure type or a macro definition, for example. This may be due to CodeWarrior's cached definition; it can normally be rectified by clearing all object code from a project and rebuilding:



How do I add interrupt handlers?

CodeWarrior supports a number of ways of incorporating interrupts, the most common methods being

- Use a #pragma TRAP_PROC prior to the interrupt routine and add to the vector table in the linker.prm file. For example:

```
#pragma TRAP_PROC
void intSW1(void){

}
```

or

- Use the keyword “interrupt” and add to the vector table in the linker.prm file. For example:

```
interrupt void intSW1(void){

}
```

and:

- Add the vector table entry to the linker.prm file. For example:

```
VECTOR ADDRESS 0xFFD2 intSW1
```

- Use the keyword "interrupt" and specify the interrupt vector number in the definition of the interrupt routine. This does not require any modification to the linker.prm file. For example:

```
interrupt 22 void intSW1(void){
}
```

This method has the advantage of defining an interrupt handler and its vector in a single file, removing the need to maintain two files.

How can I use the assembler within C?

Refer to section "High Level Online Assembler for Motorola HC08" in the document "Manual_Compiler_HC08.pdf" included with CodeWarrior.

How are interrupt vectors redirected?

The serial monitor implements a redirection of the vector table in an unprotected area of FLASH; this is discussed in the HCS08 serial monitor documentation.

How do I use masks?

A mask-based definition for a register would look something like:

```
/*** DBGC - Debug Control Register ***/
extern volatile byte _DBGC @0x00001816;
#define RWBEN   0x01;  /* Enable R/W for Comparator B */
#define RWB     0x02;  /* R/W Comparison Value for Comparator B */
#define RWAEN   0x04;  /* Enable R/W for Comparator A */
#define RWA     0x08;  /* R/W Comparison Value for Comparator A */
#define BRKEN   0x10;  /* Break Enable */
#define TAG     0x20;  /* Tag/Force Select */
#define ARM     0x40;  /* Arm Control */
#define DBGEN   0x80;  /* Debug Module Enable */
```

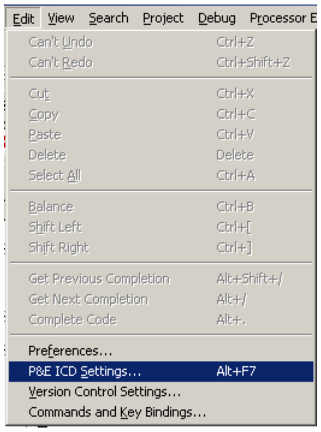
To access RWB would require the following code:

```
DBGC = DBGC | RWB;       /*Set RWB bit of DBGC */
DBGC = DBGC & ~RWB;      /*Clear RWB bit of DBGC */
```

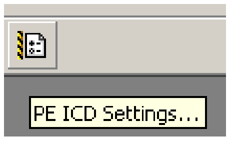
NOTE: *CodeWarrior header files implement structures, NOT bit masks. If you wish to use bit masks, thet must be defined manually.*

How do I set the compiler options?

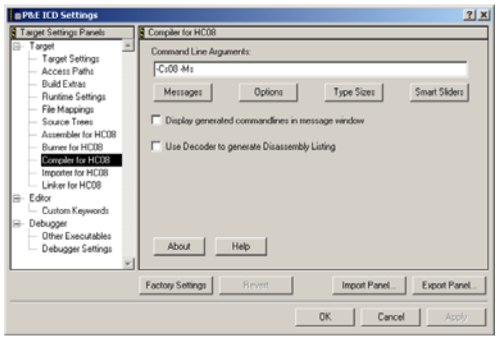
In order to set compiler options, select “Settings” from the Edit menu:



Or press the Settings button:



A dialog box will appear, allowing access to the compiler options, either by simply entering them as command line arguments:

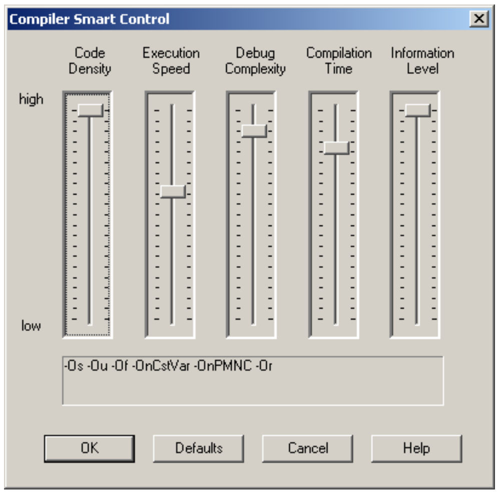


Command Line Arguments: -Cs08 -Ms

☐ Display generated commandlines in message window

☐ Use Decoder to generate Disassembly Listing

or by using one of the sub-option windows; for example, “Smart sliders”:



-Os -Ou -Of -OnCstVar -OnPMNC -Or

This provides a graphical front end to the selection of the compiler switches.

Big Endian or Little Endian?

The Endianness of a processor refers to the order in which it stores multiple byte values in memory. Big Endian processors store the most significant byte at the lowest address, whereas Little Endian processors store the least significant byte at the lowest address. This can cause issues if it is not taken into account.

For a more thorough discussion of Endianness refer to:

http://www.wikipedia.org/wiki/Endianness

The HCS08 (like all 68HCxx) is a Big Endian processor.

Endianness is also used to express the order of bits within a byte, and is typically used within serial communications. Serial communications may expect data least significant bit (Little Endian) or most significant bit (Big Endian) first. RS232 expects bits to be sent in Little Endian format. The SCI on an HCS08 transmits data in Little Endian format; the SPI on an HCS08 is selectable as Big Endian or Little Endian. An example of a Big Endian serial protocol is MIL-STD-1553B, where data transmissions are most significant bit first.

NOTE: *If connecting external peripherals, check the Endianness to ensure compatibility.*

Check core and compiler for Big Endianness

The following piece of code checks the processor and compiler for Big Endianness.

First, create a union/structure to allow word or byte access to a union/structure for a 16-bit integer:

```
// Declare union to access word as word or two bytes
typedef union {
  word w;
  struct {
    byte h;
    byte l;
  } bytes;
} TEMP;
```

Next, define required variables, one using the union/structure, an error counter, and a byte pointer:

```
TEMP t;       //declare temporary variable using union
byte err;     //error count
byte *p;      //pointer to access bytes within word
```

Clear the error count, and set the word so that the upper and lower bytes are unique:

```
err=0;          //clear error count
t.w = 0x55aa;   //set word so high byte and low byte are different
```

Check using pointers to access the individual bytes of the word in the correct order:

```
p=(byte *)&t.w;            //set pointer to address of word
if(t.bytes.h!=*p)err|=1;  //should be pointing to the high byte
p++;                       //increment pointer to next byte
if(t.bytes.l!=*p)err|=2;  //should be pointing to the low byte
```

Check that, using union/structure, access to the individual bytes of the word is correct:

```
if(t.bytes.h!=0x55)err|=4;  //check high byte accessed as structure
if(t.bytes.l!=0xaa)err|=8;  //check low byte accessed as structure
```

The results should be 0.

Is Linux/Unix support available?

Cosmic supports Linux, Solaris and HP/UX as well as supporting Windows:

Platform	Part No	Description
Linux	CLXH08	C Cross Compiler, Assembler, Linker and IDEA package targeting Motorola's 68HC08 microcontroller
SUN Solaris	CSSH08	C Cross Compiler, Assembler, Linker and IDEA package targeting Motorola's 68HC08 microcontroller
SUN Solaris	ZSSH08SIM	ZAP Debugger Simulator for 68HC08
HP-UX	CHPH08	C Cross Compiler, Assembler, Linker and IDEA package targeting Motorola's 68HC08 microcontroller
HP-UX	ZHPH08SIM	ZAP Debugger Simulator for 68HC08

References

The following publications may be of interest. They are available on Motorola's web site at:

http://www.motorola.com/

AN1752/D	Data Structures for 8-Bit Microcontrollers
AN1837/D	Non-Volatile Memory Technology Overview
AN2093/D	Creating Efficient C Code for the MC68HC08
AN2111/D	A Coding Standard for HCS08 Assembly Language
AN2140/D	Serial Monitor for MC9S08GB/GT
AN2342/D	Opto Isolation Circuits For In Circuit Debugging of 68HC9(S)12 and 68HC908 Microcontrollers
AN2438/D	ADC Definitions and Specifications
AN2493/D	AN2493/D: MC9S08GB/GT Low Power Modes
AN2494/D	Configuring the System and Peripheral Clocks in the MC9S08GB/GT
AN2496/D	Calibrating the MC9S08GB/GT Internal Clock Generator (ICG)
AN2497/D	HCS08 Background Debug Mode versus HC08 Monitor Mode
M68EVB908GB60	Development board for Motorola MC9S08GB60
M68DEMO908GB60	Demonstration Board for Motorola MC9S08GB60
MC9S08GB60/D	MC9S08GB60, MC9S08GT60, MC9S08GB32, MC9S08GT32 Data Sheet
HCS08RMv1/D	HCS08 Family Reference Manual

MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. YOU ASSUME THE ENTIRE RISK ARISING OUT OF THE USE OR PERFORMANCE OF THE SOFTWARE, OR ANY SYSTEMS YOU DESIGN USING THE SOFTWARE (IF ANY). NOTHING IN THIS AGREEMENT MAY BE CONSTRUED AS A WARRANTY OR REPRESENTATION BY MOTOROLA THAT THE SOFTWARE OR ANY DERIVATIVE WORK DEVELOPED WITH OR INCORPORATING THE SOFTWARE WILL BE FREE FROM INFRINGEMENT OF THE INTELLECTUAL PROPERTY RIGHTS OF THIRD PARTIES.

INDEMNITY. You agree to fully defend and indemnify Motorola from any and all claims, liabilities, and costs (including reasonable attorney's fees) related to (1) your use (including your sublicensee's use, if permitted) of the Software or (2) your violation of the terms and conditions of this Agreement.

LIMITATION OF LIABILITY. IN NO EVENT WILL MOTOROLA BE LIABLE, WHETHER IN CONTRACT, TORT, OR OTHERWISE, FOR ANY INCIDENTAL, SPECIAL, INDIRECT, CONSEQUENTIAL OR PUNITIVE DAMAGES, INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR ANY LOSS OF USE, LOSS OF TIME, INCONVENIENCE, COMMERCIAL LOSS, OR LOST PROFITS, SAVINGS, OR REVENUES TO THE FULL EXTENT SUCH MAY BE DISCLAIMED BY LAW.

COMPLIANCE WITH LAWS; EXPORT RESTRICTIONS. You must use the Software in accordance with all applicable U.S. laws, regulations and statutes. You agree that neither you nor your licensees (if any) intend to or will, directly or indirectly, export or transmit the Software to any country in violation of U.S. export restrictions.

GOVERNMENT USE. Use of the Software and any corresponding documentation, if any, is provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(l) and (2) of the Commercial Computer Software--Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Motorola, Inc., 6501 William Cannon Drive West, Austin, TX, 78735.

HIGH RISK ACTIVITIES. You acknowledge that the Software is not fault tolerant and is not designed, manufactured or intended by Motorola for incorporation into products intended for use or resale in on-line control equipment in hazardous, dangerous to life or potentially life-threatening environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems, in which the failure of products could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). You specifically represent and warrant that you will not use the Software or any derivative work of the Software for High Risk Activities.

CHOICE OF LAW; VENUE; LIMITATIONS. You agree that the statutes and laws of the United States and the State of Texas, USA, without regard to conflicts of laws principles, will apply to all matters relating to this Agreement or the Software, and you agree that any litigation will be subject to the exclusive jurisdiction of the state or federal courts in Texas, USA. You agree that regardless of any statute or law to the contrary, any claim or cause of action arising out of or related to this Agreement or the Software must be filed within one (1) year after such claim or cause of action arose or be forever barred.

PRODUCT LABELING. You are not authorized to use any Motorola trademarks, brand names, or logos.

ENTIRE AGREEMENT. This Agreement constitutes the entire agreement between you and Motorola regarding the subject matter of this Agreement, and supersedes all prior communications, negotiations, understandings, agreements or representations, either written or oral, if any. This Agreement may only be amended in written form, executed by you and Motorola.

SEVERABILITY. If any provision of this Agreement is held for any reason to be invalid or unenforceable, then the remaining provisions of this Agreement will be unimpaired and, unless a modification or replacement of the invalid or unenforceable provision is further held to deprive you or Motorola of a material benefit, in which case the Agreement will immediately terminate, the invalid or unenforceable provision will be replaced with a provision that is valid and enforceable and that comes closest to the intention underlying the invalid or unenforceable provision.

NO WAIVER. The waiver by Motorola of any breach of any provision of this Agreement will not operate or be construed as a waiver of any other or a subsequent breach of the same or a different provision.

Trademarks

- Motorola and the Motorola logo are registered trademarks of Motorola, Inc.
- Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the U.S. and other countries.
- UNIX is a registered trademark of Open Group in the US and other countries.
- P&E is a trademark of P&E Microcomputer Systems, Inc.
- CodeWarrior® is a registered trademark of MetroWerks, a wholly owned subsidiary of Motorola, Inc.

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:
Motorola Literature Distribution
P.O. Box 5405, Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

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

ASIA/PACIFIC:
Motorola Semiconductors H.K. Ltd.
Silicon Harbour Centre
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
852-26668334

HOME PAGE:
http://motorola.com/semiconductors

MOTOROLA