Freescale Semiconductor
Application Note

Document Number: AN3465
Rev. 0, 06/2007

Migrating within the controller continuum

Flexis^{TM} S08 & ColdFire V1

by: Gabriel Sanchez, Eduardo Montañez,
Rafael Peralez and Manuel Davalos
RTAC Americas

As product complexity grows and the need to add new features accelerates, engineers face a challenging task of migrating applications from 8-bit to 32-bit microcontrollers (MCU). This application note describes how the Flexis family of MCUs eases this task. Although peripherals are identical in this family, there are differences between 8-bit and 32 bit architectures. This document provides a guide on how to develop an application where the same code can be used in both cores without changes.

This application note covers the architectural differences between 8-bit and 32-bit MCUs. It provides tips and tricks for writing C applications, ensuring code reuse and an easy migration. Finally, it details the enhancements in CodeWarrior for Microcontrollers V6.0, describing how the tool simplifies the porting experience between S08 and ColdFire V1.

Contents

1 Why migrate between 8-bit and 32-bit?

Before deciding to use 8-bit or 32-bit technology, the following reasons should be considered:

Table 1. 8-bit vs. 32-bit

8-bit	32-bit
Applications demand more performances.	Increases memory footprint and a wider portfolio of devices.
Devices tend to consume less power, where low power consumption is more critical than performance.	Provides access to high-end hardware devices and complex peripherals.
Packaging is an important advantage where space is critical.	More memory.
The core is not as complex.	The gap between 32-bit prices and 8-bit prices are getting smaller.
Tools tend to be more cost-effective.	

2 Common migration headaches

- A difficulty that can be found when migrating from applications between 8-bits and 32- bits is the use of different software tools. The time it takes to learn features and functionality can be significant, depending on how different the tools are. Derivatives and libraries might need updating when switching among devices. Hardware tools are commonly different; this implies extra cost and development time.
- The change in software implies the use of different documentation, register naming and of course peripherals. The configuration of peripherals can get as complicated as the complexity gap between an 8 and a 32-bit architecture.
- Simple factors like differences in debugging tools and debugging modules of both devices can impact the development.
- Regarding the pin-out, peripherals and supply pins might be located in different areas of the silicon. The supply voltage might be different and extra hardware might be required. The assignment of signals can vary creating conflict when building the new board.
- Different power modes can be found because they depend on the architecture.

All these differences lead a designer to re-write software, from peripherals to the variations in memory maps and changes in exception handling. These migration headaches could be the difference between a failing and a successful project.

3 Building using Freescale Controller Continuum

A Freescale Controller Continuum is an 8-bit to 32-bit compatible device that shares a common set of peripherals and tools.

It is important for a developer to know an array of products that range in performance and allow different options for different applications. Designs evolve and applications demand more performance and functionalities. Process technology improvements push costs down on 32-bit MCUs making them more affordable.

Migrating across continuum of performance and price options are not easy or quick when different bit architectures require re-coding and different tools.

Freescale is breaking bit boundaries to provide a simple and seamless path performance between 8-bit and 32-bit MCUs.

3.1 The Flexis Series of Microcontrollers

The controller continuum “Connection Point”

A number of factors contribute to a seamless transition from an 8-bit to a 32-bit architecture. The improvements start at the core and involve peripherals and packaging as well as development tools.

The ColdFire instruction set architecture (ISA) has changed since its initial implementation over a decade ago. It has changed from an optimization philosophy for 32-bit operands. It now expands and improves support for handling 8-bit and 16-bit operands. For the V1 class of applications, this is particularly important for users migrating from 8-bit and 16-bit applications. At the processor core level, it is important for the implementation of the instruction set to directly address some of the issues that involve efficient handling of 8-bit and 16-bit operands.

The V1 core microarchitecture has improved the V2 ColdFire regarding the 8-bit and 16-bit operands. In applications with data referenced to 8-bit and 16-bit operands, the instruction support and fast execution time for those instructions provides a considerable performance improvement.

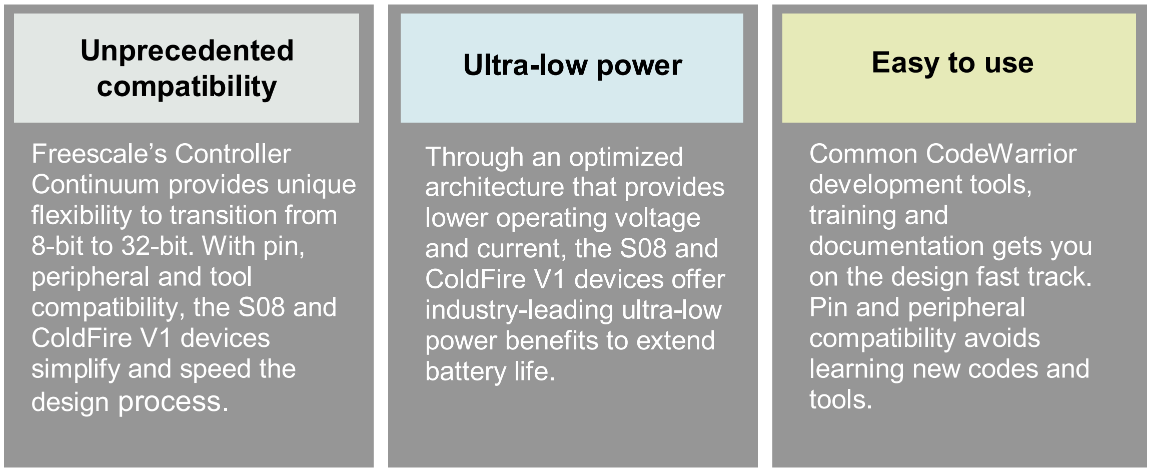
The continuum takes advantage of existing peripherals. Peripheral blocks connect to a common bus structure and complete reuse of the S08 is ensured. As a result, the 8-bit projects can be re-usable.

The Flexis series of microcontrollers have:

- Single development tool to ease migration between 8-bit (S08) and 32-bit (CFV1). The same CodeWarrior versions support both cores.
- Common peripheral set to preserve software investment between 8-bit and 32-bit.
- Practical pin compatibility wherever to maximize hardware reuse when moving between 8-bit and 32-bit.

4 Industry’s first 8-bit and 32-bit compatible MCUs

S08 and ColdFire V1are the industry’s ultra low power flexis devices first 8-bit and 32-bit pin, peripheral and tool compatible MCUs. They make the design process quick, easy and limitless.



Unprecedented compatibility

Freescale's Controller Continuum provides unique flexibility to transition from 8-bit to 32-bit. With pin, peripheral and tool compatibility, the S08 and ColdFire V1 devices simplify and speed the design process.

Ultra-low power

Through an optimized architecture that provides lower operating voltage and current, the S08 and ColdFire V1 devices offer industry-leading ultra-low power benefits to extend battery life.

Easy to use

Common CodeWarrior development tools, training and documentation gets you on the design fast track. Pin and peripheral compatibility avoids learning new codes and tools.

Figure 1. Benefits of Flexis Series Devices

Evident advantages can be found using Flexis. The Software re-investment and design cycle time is drastically reduced. Peripheral and tools compatibility makes the transition between 8-bit and 32-bit fast and simple.

Table 2 recommends when to use the S08 and ColdFire V1 separately and together.

Table 2. When is Necessary to Use S08, Both or ColdFire V1

Use S08 when:	Use *both* S08 & ColdFire V1 when:	Use ColdFire V1 when:
• Low power consumption required with minimal extended mathematical computation. • Low pin count or pin count options desired. • No application requirement for higher performance calculations or peripherals. • Greater cost sensitivity. • More headroom with 8-bit machine.	• You are designing for a portfolio of products of varying performance and price options.	• Immediate or near-term requirement for higher mathematical processing performance. • Immediate or near-term requirement for faster-throughput communication interfaces such as ethernet or USB. • Interested in achieving low-cost and low-overhead link to future 32-bit designs. • Need a path to higher flash densities (>128KB).

5 Hardware comparison

Most of the competitions´ MCUs do not have the flexibility to migrate an application from an 8-bit MCU to a 32-bit MCU. These hardware architectures are not sharing important things such as peripherals. Figure 2 shows the hardware architecture the S08 and ColdFire V1 share. This sharing makes the migrations an easy practice for applications engineers and developers.

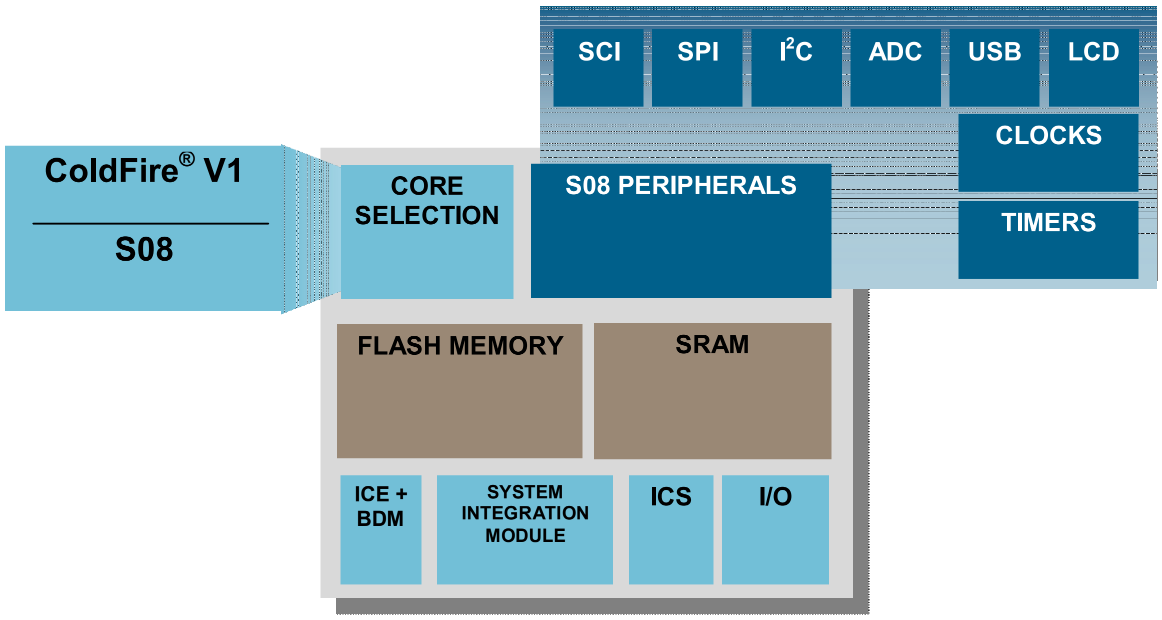


Figure 2. S08 and ColdFire V1 Hardware Sharing

It is impressive that both cores are sharing the same peripherals. The BDM and ICS blocks are common to both cores and it is possible to select from S08 or CF V1 using the same software compiler tool, CodeWarrior 6.0 and the same programmer tool, BDM.

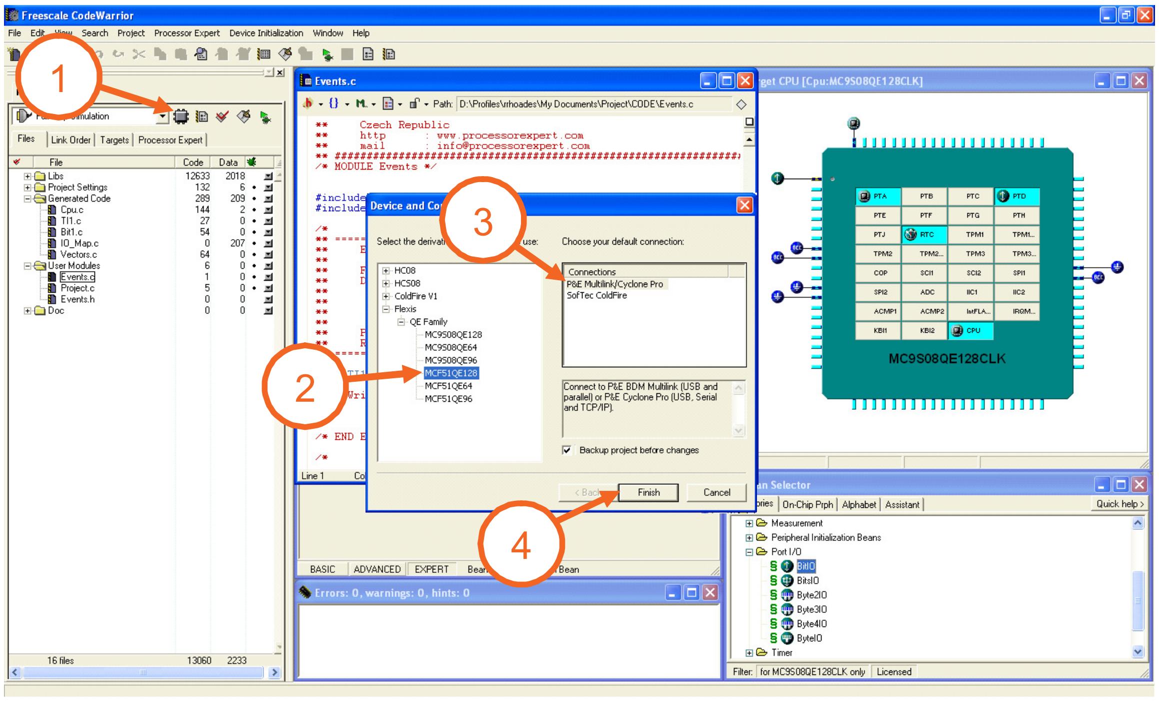
6 MCU change wizard

The MCU change wizard in CodeWarrior 6.0 allows flexibility and changes cores with very few clicks.

A project can change from an S08 to a V1 core in four simple steps.

1. Choose the option Change MCU/Connections.
2. Select the correct ColdFire V1 MCU to migrate the application.
3. Select the connection mode P&E Multilink (BDM).
4. Click on the finish button.

Figure 3 shows how to change from an S08 core to a ColdFire V1 core.



Path: D:\Profiles\rrhoades\My Documents\Project\CODE\Events.c

```
**     Czech Republic
**     http       : www.processorexpert.com
**     mail       : info@processorexpert.com
** ###################################################################
/* MODULE Events */
```

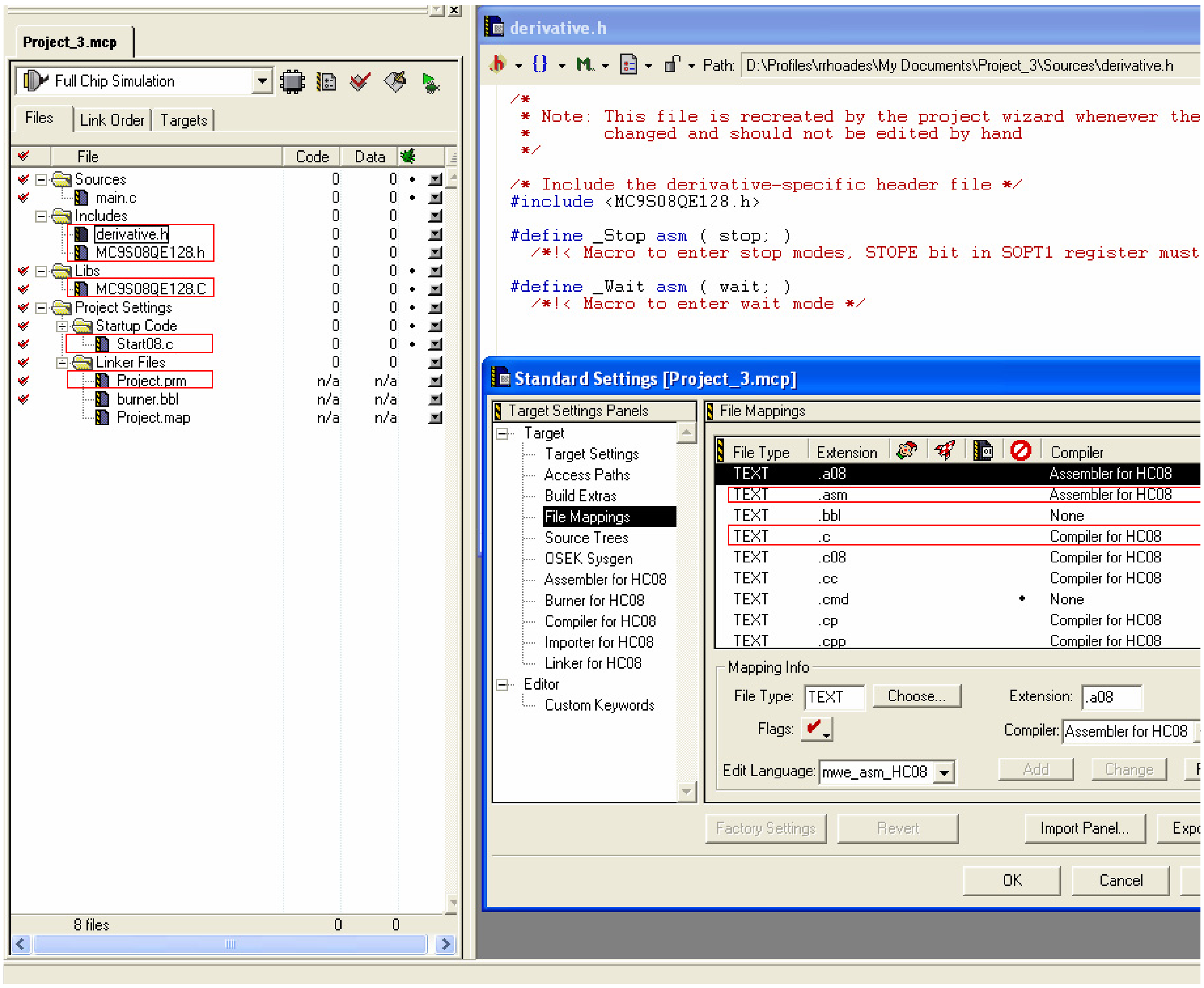
Connect to P&E BDM Multilink (USB and parallel) or P&E Cyclone Pro (USB, Serial and TCP/IP).

Errors: 0, warnings: 0, hints: 0

Figure 3. Changing the S08 Core to ColdFire V1 Core

The group of libraries is different in the 8 and 32-bit core. The integrated tool provides all the files used to build a project in both Flexis devices.

Figure 4 and Figure 5, show the basic project files for S08 and ColdFire V1 respectively.



File	Code	Data
Sources	0	0
main.c	0	0
Includes	0	0
derivative.h	0	0
MC9S08QE128.h	0	0
Libs	0	0
MC9S08QE128.C	0	0
Project Settings	0	0
Startup Code	0	0
Start08.c	0	0
Linker Files	0	0
Project.prm	n/a	n/a
burner.bbl	n/a	n/a
Project.map	n/a	n/a
8 files	0	0

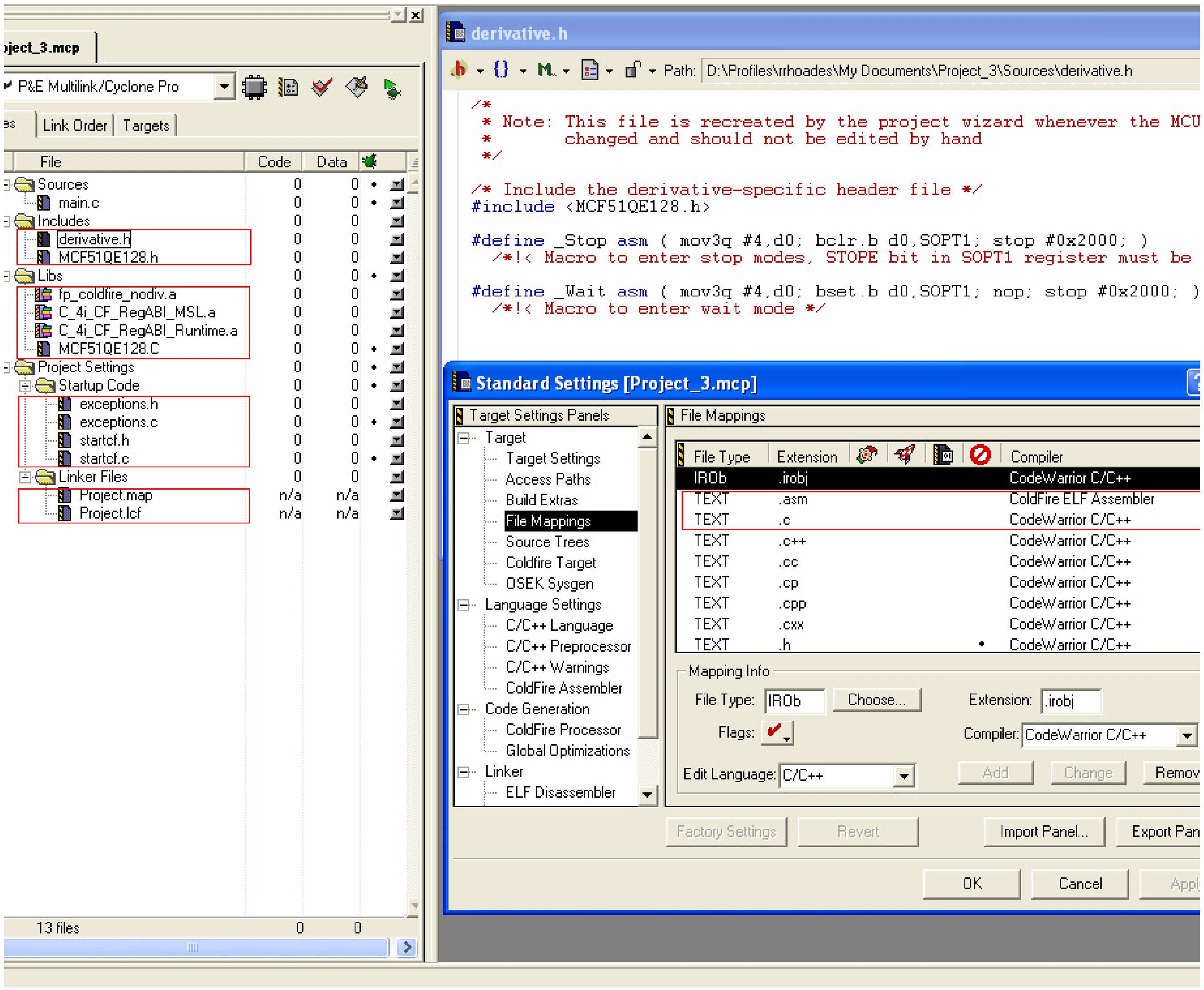
```
/*
 * Note: This file is recreated by the project wizard whenever the
 *       changed and should not be edited by hand
 */

/* Include the derivative-specific header file */
#include <MC9S08QE128.h>

#define _Stop asm ( stop; )
  /*!< Macro to enter stop modes, STOPE bit in SOPT1 register must

#define _Wait asm ( wait; )
  /*!< Macro to enter wait mode */
```

Target Settings Panels:
- Target
  - Target Settings
  - Access Paths
  - Build Extras
  - File Mappings
  - Source Trees
  - OSEK Sysgen
  - Assembler for HC08
  - Burner for HC08
  - Compiler for HC08
  - Importer for HC08
  - Linker for HC08
- Editor
  - Custom Keywords

File Type	Extension	Compiler
TEXT	.a08	Assembler for HC08
TEXT	.asm	Assembler for HC08
TEXT	.bbl	None
TEXT	.c	Compiler for HC08
TEXT	.c08	Compiler for HC08
TEXT	.cc	Compiler for HC08
TEXT	.cmd	None
TEXT	.cp	Compiler for HC08
TEXT	.cpp	Compiler for HC08

Mapping Info
File Type: TEXT Extension: .a08
Flags: ✓ Compiler: Assembler for HC08
Edit Language: mwe_asm_HC08

Figure 4. Basic CodeWarrior Project file for S08


ject_3.mcp

P&E Multilink/Cyclone Pro

Link Order | Targets

File	Code	Data
Sources	0	0
main.c	0	0
Includes	0	0
derivative.h	0	0
MCF51QE128.h	0	0
Libs	0	0
fp_coldfire_nodiv.a	0	0
C_4i_CF_RegABI_MSL.a	0	0
C_4i_CF_RegABI_Runtime.a	0	0
MCF51QE128.C	0	0
Project Settings	0	0
Startup Code	0	0
exceptions.h	0	0
exceptions.c	0	0
startcf.h	0	0
startcf.c	0	0
Linker Files	0	0
Project.map	n/a	n/a
Project.lcf	n/a	n/a
13 files	0	0

derivative.h

Path: D:\Profiles\rrhoades\My Documents\Project_3\Sources\derivative.h

```
/*
 * Note: This file is recreated by the project wizard whenever the MCU
 *       changed and should not be edited by hand
 */

/* Include the derivative-specific header file */
#include <MCF51QE128.h>

#define _Stop asm ( mov3q #4,d0; bclr.b d0,SOPT1; stop #0x2000; )
  /*!< Macro to enter stop modes, STOPE bit in SOPT1 register must be

#define _Wait asm ( mov3q #4,d0; bset.b d0,SOPT1; nop; stop #0x2000; )
  /*!< Macro to enter wait mode */
```

Standard Settings [Project_3.mcp]

Target Settings Panels
- Target
  - Target Settings
  - Access Paths
  - Build Extras
  - File Mappings
  - Source Trees
  - Coldfire Target
  - OSEK Sysgen
- Language Settings
  - C/C++ Language
  - C/C++ Preprocessor
  - C/C++ Warnings
  - ColdFire Assembler
- Code Generation
  - ColdFire Processor
  - Global Optimizations
- Linker
  - ELF Disassembler

File Mappings

File Type	Extension	Compiler
IROb	.irobj	CodeWarrior C/C++
TEXT	.asm	ColdFire ELF Assembler
TEXT	.c	CodeWarrior C/C++
TEXT	.c++	CodeWarrior C/C++
TEXT	.cc	CodeWarrior C/C++
TEXT	.cp	CodeWarrior C/C++
TEXT	.cpp	CodeWarrior C/C++
TEXT	.cxx	CodeWarrior C/C++
TEXT	.h	CodeWarrior C/C++

Mapping Info

File Type: IROb Extension: .irobj

Flags: ✓ Compiler: CodeWarrior C/C++

Edit Language: C/C++

Add | Change | Remov

Factory Settings | Revert | Import Panel... | Export Pan

OK | Cancel | Appl

Figure 5. Basic CodeWarrior Project file for ColdFire V1

This same CodeWarrior version supports 8-bit S08 devices and 32-bit ColdFire V1 plus support for RS08 and HC08.

7 Common Porting Mistakes when moving from 8-bit S08 to 32-bit ColdFire V1

To obtain a better understanding of what the porting tips are and how to use them, it is necessary to take a specific example migrating from 8-bit S08 to 32-bit ColdFire V1.

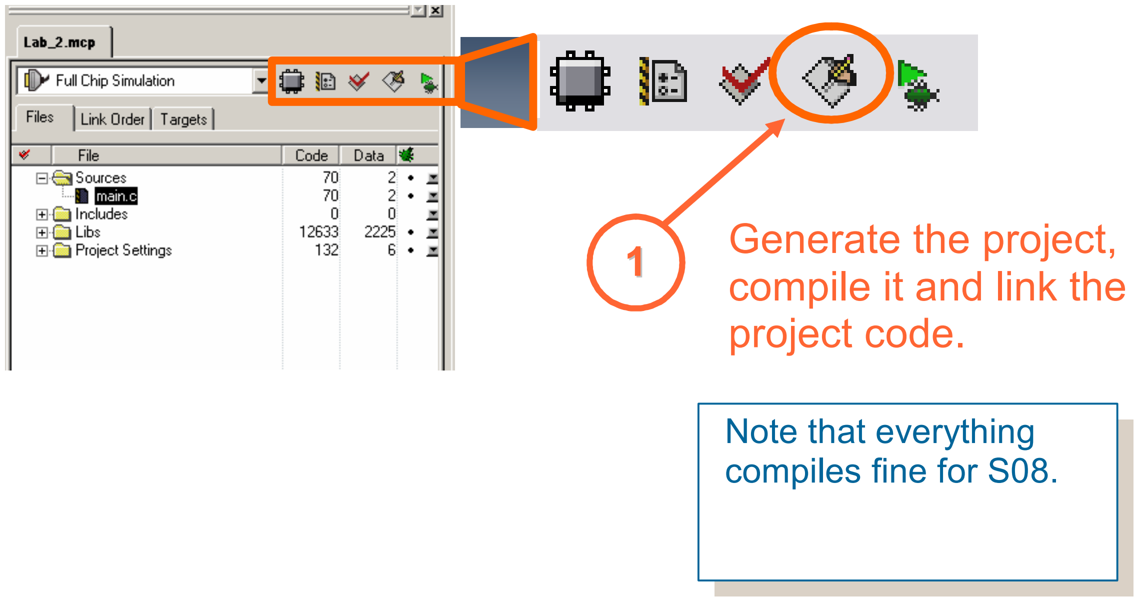
An example based on the MC9S08QE128 project is developed. The objective is to migrate it to the MCF51QE128. These particular devices are the first Flexis devices, and therefore were selected for the project. This same procedure could be used to migrate future members of the Flexis family.

Along with this application note is a zip file. Save it and unzip it. For this example use the file Lab_2.mcp in the Controller Continuum\Lab_2 folder. Generate the, compile it and link the project code Figure 6; no errors must be generated after this.

After clicking the debug button Figure 7 this project is ready to run in MC9S08QE128.

Lab_2.mcp project exercises. The KBI and RTI peripherals:

- The RTC is enabled to blink LED1 every second and increment a global_variable.
- The KBI is configured so that every time a push button is pressed, LED2 toggles
- The MCU is in stop mode until an interruption occurs.



File	Code	Data
Sources	70	2
main.c	70	2
Includes	0	0
Libs	12633	2225
Project Settings	132	6

Figure 6. How to Make the Project?

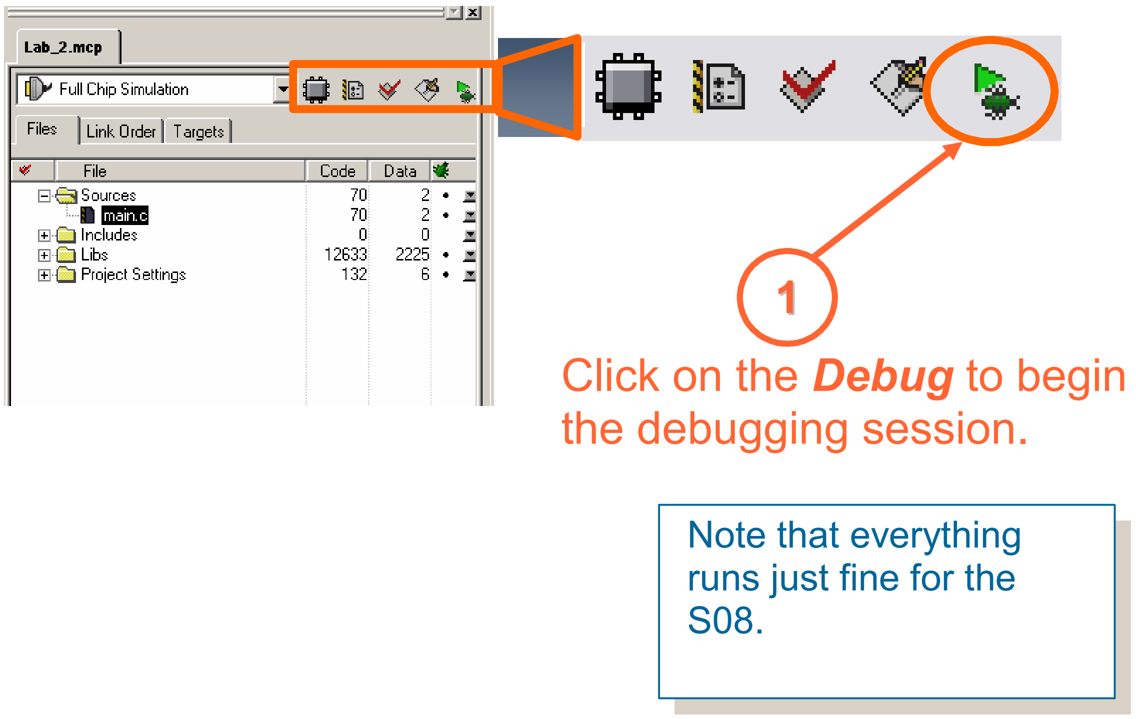


Figure 7. How to Debug the Project?

Notice that, when working with the S08QE128, the project compiles and runs perfectly. We have a code that is using GPIO, KBI and RTI. The stop instruction is also used to minimize current consumption while executing the code.

In order to change the core from S08 to ColdFire V1, follow the steps in Chapter 6 MCU Change Wizard. There is one more step to follow in his chapter besides those, a step is added.

1. Click on the MCU Change Wizard icon Figure 8.
2. Select the MC51QE128
3. Select the connection mode P&E Multilink/Cyclone Pro.
4. Accept the option from the check box, create a backup zip file to backup
5. Click finish Figure 9.

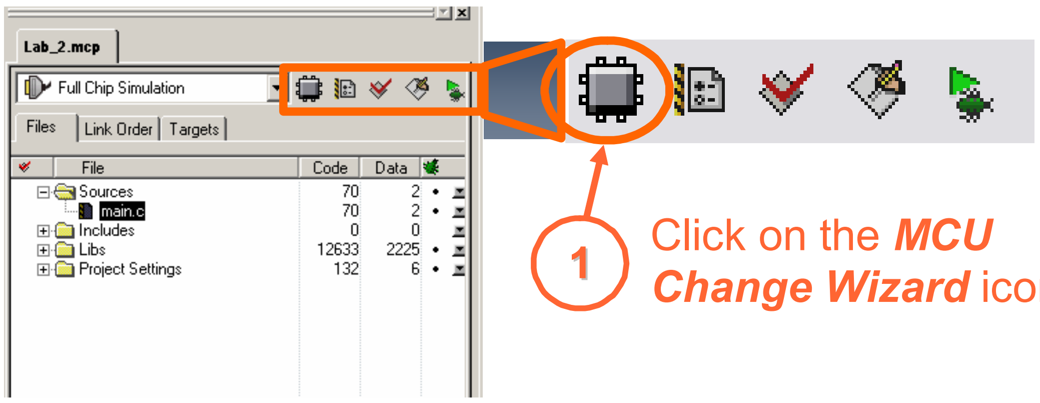


Figure 8. How to Change the S08 Core to ColdFire V1 Core? Firs step

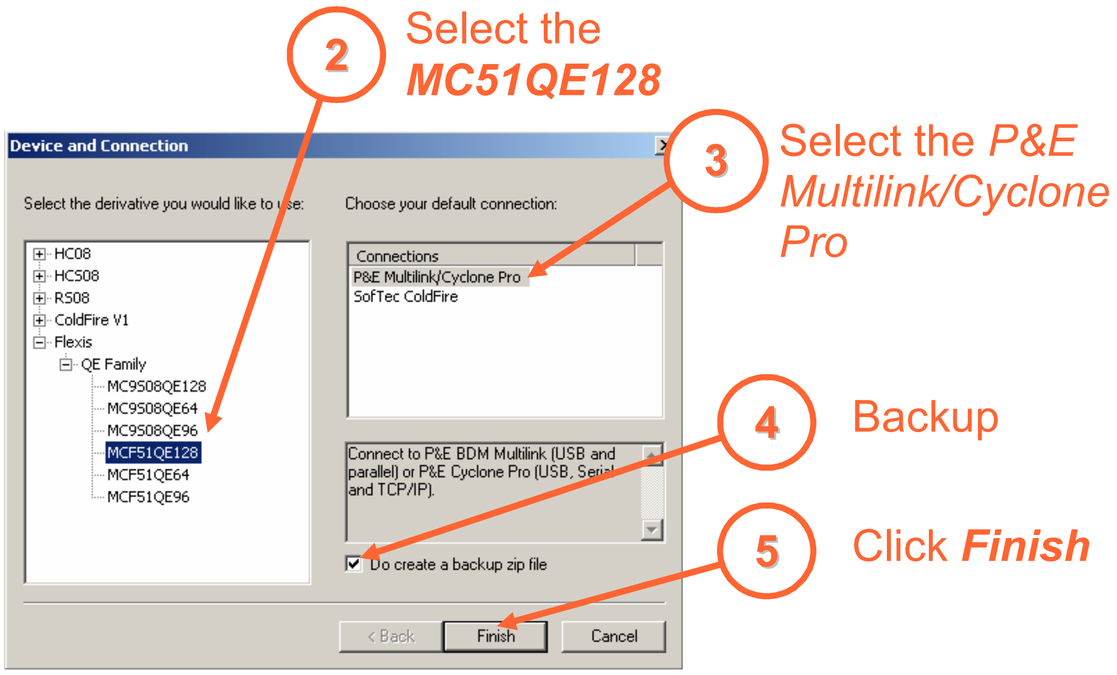
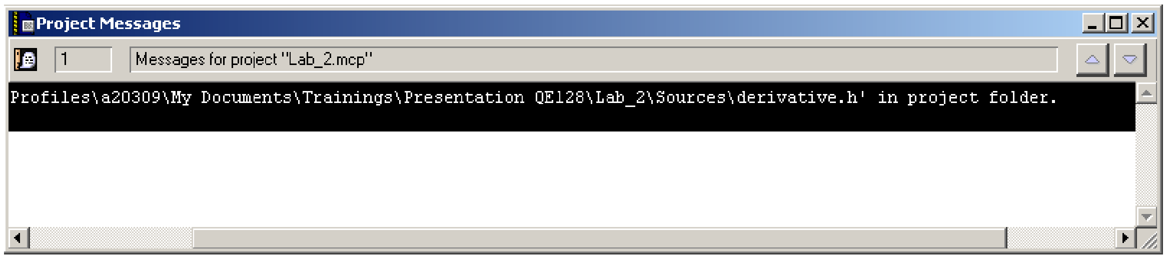

2
Select the *MC51QE128*
3
Select the *P&E Multilink/Cyclone Pro*
4
Backup
5
Click *Finish*
Device and Connection
Select the derivative you would like to use:
HC08
HCS08
RS08
ColdFire V1
Flexis
QE Family
MC9S08QE128
MC9S08QE64
MC9S08QE96
MCF51QE128
MCF51QE64
MCF51QE96
Choose your default connection:
Connections
P&E Multilink/Cyclone Pro
SofTec ColdFire
Connect to P&E BDM Multilink (USB and parallel) or P&E Cyclone Pro (USB, Serial and TCP/IP).
Do create a backup zip file
< Back
Finish
Cancel

Figure 9. How to Change the S08 Core to ColdFire V1 Core? Second Step

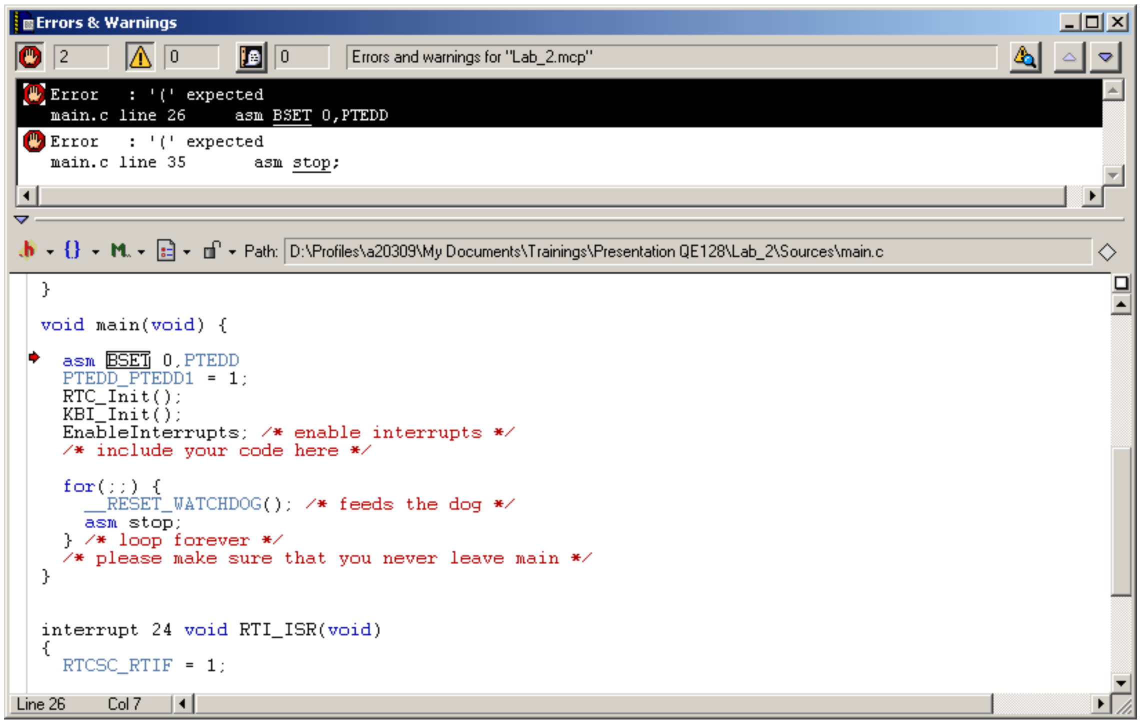
6. CodeWarrior automatically changes the derivative.h file and shows the message shown in Figure 10:



```
Profiles\a20309\My Documents\Trainings\Presentation QE128\Lab_2\Sources\derivative.h' in project folder.
```

Figure 10. Project Messages

7. Compile the project.
8. The device has changed. Figure 11 shows the porting mistakes errors.


Errors & Warnings
2
0
0
Errors and warnings for "Lab_2.mcp"
Path: D:\Profiles\a20309\My Documents\Trainings\Presentation QE128\Lab_2\Sources\main.c
Line 26
Col 7

```
Error   : '(' expected
main.c line 26      asm BSET 0,PTEDD
Error   : '(' expected
main.c line 35       asm stop;
```

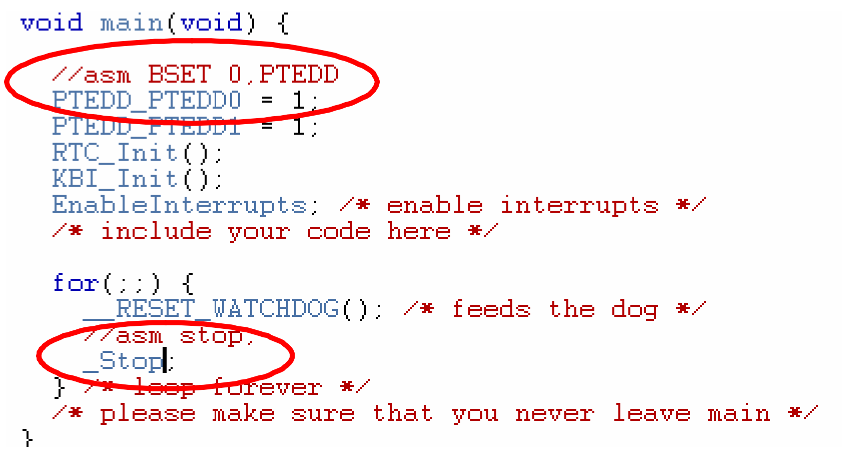
```
}
void main(void) {

  asm BSET 0,PTEDD
  PTEDD_PTEDD1 = 1;
  RTC_Init();
  KBI_Init();
  EnableInterrupts; /* enable interrupts */
  /* include your code here */

  for(;;) {
    __RESET_WATCHDOG(); /* feeds the dog */
    asm stop;
  } /* loop forever */
  /* please make sure that you never leave main */
}

interrupt 24 void RTI_ISR(void)
{
  RTCSC_RTIF = 1;
```

Figure 11. Error Messages after Compiling the Project using the ColdFire V1 Core

When porting the software from 8-bit S08 to 32-bit ColdFire V1, architecture differences could affect the software operation. In this example, the architectural differences will lead the compiler to find errors. These errors are intentionally introduced to work through them and a code version that works for both devices without any changes can be developed.

7.1 Porting Tip 1: Remove the assembly code

When switching between different architectures the use of in-line assembly instructions is ineffective. 8-bit S08 core and 32-bit ColdFire V1 core have different instruction set architectures. If the code has some in-line assembly instructions, this can lead to a compiler error stating that the instruction operand is invalid. The C code for replacing in-line assembly instructions with correct compiler instructions is an example of that.



```
void main(void) {
  //asm BSET 0,PTEDD
  PTEDD_PTEDD0 = 1;
  PTEDD_PTEDD1 = 1;
  RTC_Init();
  KBI_Init();
  EnableInterrupts; /* enable interrupts */
  /* include your code here */

  for(;;) {
    __RESET_WATCHDOG(); /* feeds the dog */
    //asm stop;
    _Stop;
  } /* loop forever */
  /* please make sure that you never leave main */
}
```

Figure 12. Assembly Instructions

Notice in the figure the assembly instruction for setting bit 0 of the data direction register for Port E has been replaced for an equivalent C instruction.

_Stop and _Wait definitions replace commonly used in line assembly to send the MCU to a low-power mode. These definitions are in derivative.h and get updated when porting the project between S08 and ColdFire V1. By using these definitions for Stop and Wait instructions this ensures that the code works in both cores.

Once there is no assembly, the project compiles. However, this is no guarantee that the code works for both cores. There are a few more things to check.

7.2 Porting Tip 2: Assign Interrupt Vectors using Interrupt Declarations in CodeWarrior Header files

Interrupt vector tables between the 8-bit S08 and 32-bit ColdFire V1 are not identical and reside in different memory locations; therefore vector assignments do not match up in memory space. Interrupt vector assignments maintain a relative vector number between S08 and ColdFire V1.

Table 3 shows an example of the difference in the interrupt vector tables for the RTC interrupt (Vrtc).

Table 3. Address vs. Vector

	Address	Vector
MC9S08QE128	0xFFCE	24
MCF51QE128	0x(00)00_0158	86

Common mistakes:

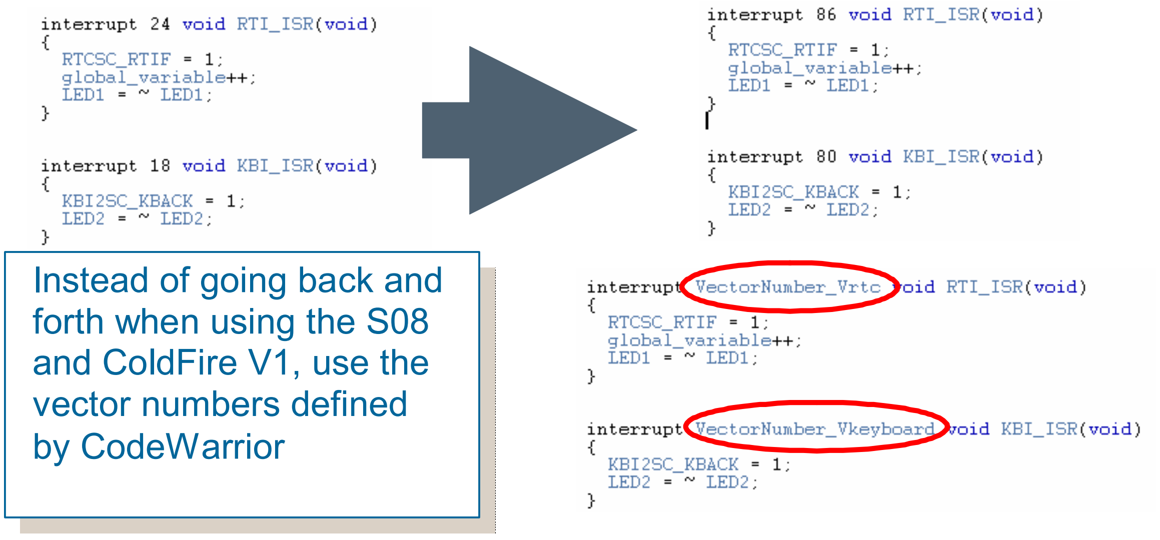
1. Solving the problem with this equation:

```
ColdFire V1 Vector Number = S08 Vector Number + 62
```

2. Improper interrupt vector assignment that only works for S08:

```
interrupt 24 RTC_ISR { }  // Only works for S08
```

Porting Tip 2 -- Using the VectorNumber_Vname interrupt declarations defined in the CodeWarrior header files Figure 13.



```
interrupt 24 void RTI_ISR(void)
{
  RTCSC_RTIF = 1;
  global_variable++;
  LED1 = ~ LED1;
}

interrupt 18 void KBI_ISR(void)
{
  KBI2SC_KBACK = 1;
  LED2 = ~ LED2;
}
```

```
interrupt 86 void RTI_ISR(void)
{
  RTCSC_RTIF = 1;
  global_variable++;
  LED1 = ~ LED1;
}

interrupt 80 void KBI_ISR(void)
{
  KBI2SC_KBACK = 1;
  LED2 = ~ LED2;
}
```

Instead of going back and forth when using the S08 and ColdFire V1, use the vector numbers defined by CodeWarrior

```
interrupt VectorNumber_Vrtc void RTI_ISR(void)
{
  RTCSC_RTIF = 1;
  global_variable++;
  LED1 = ~ LED1;
}

interrupt VectorNumber_Vkeyboard void KBI_ISR(void)
{
  KBI2SC_KBACK = 1;
  LED2 = ~ LED2;
}
```

Figure 13. Correct Assignment Interrupts Vectors

7.3 Porting Tip 3: Reference Memory using Register_Bitname Peripheral Declarations in CodeWarrior Header Files

Memory maps between the 8-bit S08 and 32-bit ColdFire V1 are different, therefore absolute memory declarations do not match up in memory space. Figure 14 shows the memory map differences between both cores.

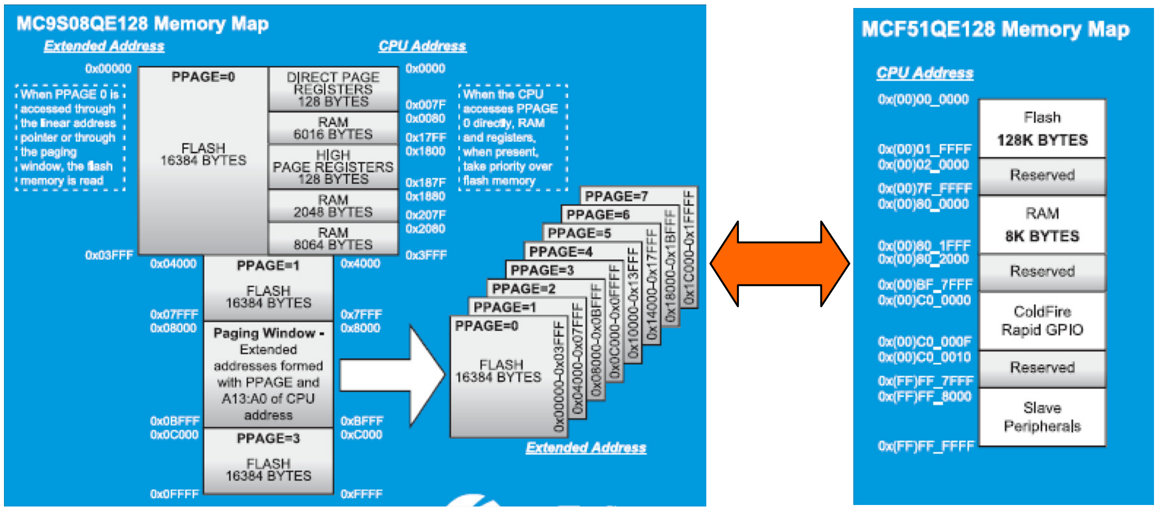


Figure 14. Memory Maps Differences between S08 and ColdFire V1

RAM and FLASH memory allocation is determined by respective linker files. Peripheral register maps maintain relative addresses between S08 and ColdFire V1. The MC9S08QE128, SCI1C1 register is located at address 0x0022 and the SCI1C1 register of the MCF51QE128 is located at address 0x(FF)FF_8000 + 0x0022 = 0x(FF)FF_8022.

Common mistakes:

1. An example of improper absolute memory declarations is:

   ```
   int var @ 0x400 = 1;
   ```

2. And last is the global_variable:

   ```
   unsigned int near global_variable @ 0x80 = 0;
   ```

Porting Tip 3 -- Reference memory using Register_Bitname peripheral declarations in CodeWarrior header files and allow linker to place variables in available memory. Remove the absolute address and near declaration:

```
unsigned int global_variable = 0;
```

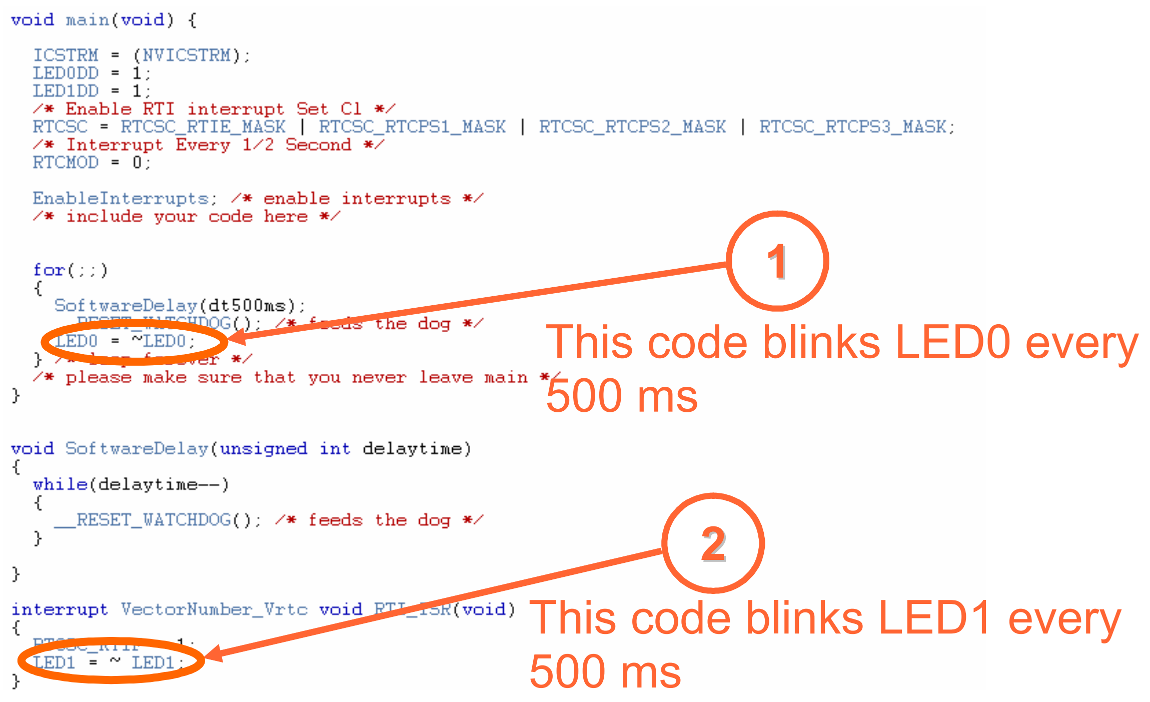
Make the project and debug. The project should now work on the ColdFire V1.

An important thing to notice here is the fact that these porting errors mention in Section 7.2 and Section 7.3 are not marked as errors by CodeWarrior.

7.4 Porting Tip 4: Avoid Software Delays and Maintain Timing through Peripherals, like TPM and RTC, with a Time Base

When migrating between 8-bit S08 and 32-bit ColdFire V1, software timing results in a problem because of the different instruction sets and instruction timings. ColdFire V1 instructions execute at CPU frequency. S08 instructions execute at Bus clock frequency. An example of instruction differences is the popular nop instruction that differs in cycle time between S08 and ColdFire V1.

Figure 15 shows an example of code with delays that blink LEDs.



```
void main(void) {

  ICSTRM = (NVICSTRM);
  LED0DD = 1;
  LED1DD = 1;
  /* Enable RTI interrupt Set C1 */
  RTCSC = RTCSC_RTIE_MASK | RTCSC_RTCPS1_MASK | RTCSC_RTCPS2_MASK | RTCSC_RTCPS3_MASK;
  /* Interrupt Every 1/2 Second */
  RTCMOD = 0;

  EnableInterrupts; /* enable interrupts */
  /* include your code here */


  for(;;)
  {
    SoftwareDelay(dt500ms);
    __RESET_WATCHDOG(); /* feeds the dog */
    LED0 = ~LED0;
  } /* loop forever */
  /* please make sure that you never leave main */
}

void SoftwareDelay(unsigned int delaytime)
{
  while(delaytime--)
  {
    __RESET_WATCHDOG(); /* feeds the dog */
  }

}

interrupt VectorNumber_Vrtc void RTI_ISR(void)
{
  RTCSC_RTIF = 1;
 LED1 = ~ LED1;
}
```

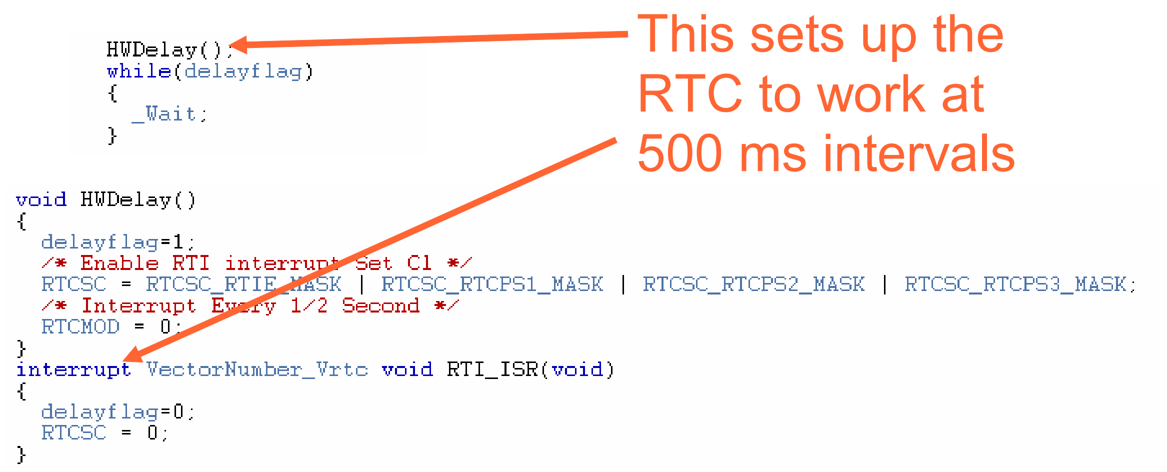
1
This code blinks LED0 every 500 ms
2
This code blinks LED1 every 500 ms

Figure 15. Bad use of Delays. Using Software Delays when Timings are Different.

When running the same code in a ColdFire V1, the LED that blinks based on a software delay changes its blinking frequency. This is as a result of the different execution times between the two instruction sets.

It is simpler and more reliable to keep a steady time base within a program by hardware. However, if it is absolutely necessary to have a software delay, keep in mind that the timing varies from the S08 to the ColdFire V1 and this is something that needs to be taken care off.

Using hardware modules lessens the load of the CPU, and enables the use of low-power modes in a program. Figure 16 shows the correct way to use delays.



```
HWDelay();
while(delayflag)
{
  _Wait;
}
```

```
void HWDelay()
{
  delayflag=1;
  /* Enable RTI interrupt Set C1 */
  RTCSC = RTCSC_RTIE_MASK | RTCSC_RTCPS1_MASK | RTCSC_RTCPS2_MASK | RTCSC_RTCPS3_MASK;
  /* Interrupt Every 1/2 Second */
  RTCMOD = 0;
}
interrupt VectorNumber_Vrtc void RTI_ISR(void)
{
  delayflag=0;
  RTCSC = 0;
}
```

Figure 16. Correct use of Delays by Hardware, using the Internal Modules of the MCU

8 How CodeWarrior Helps with these Unnoticed Porting Headaches and Common Porting Mistakes.

Follow these simple guidelines and take advantage of CodeWarrior porting aids to avoid common porting mistakes.

1. Remove the assembly code.
2. Assign interrupt vectors using VectorNumber_Vname interrupt declarations in CodeWarrior header files instead of using fixed vector numbers.
3. Reference memory using Register_Bitname peripheral declarations in CodeWarrior header files and allow linker to place variables in available memory.
4. Avoid software delays and maintain timing through peripherals, like TPM and RTC, with a time base.

8.1 Pragmas

A new feature in CodeWarrior that can help port applications faster are Porting Tips. These Porting Tips are built-in the compiler and can be controlled through new pragma instructions. They are automatically added to a ColdFire V1 project in the file porting_support.h:

```
#pragma warn_absolute on  /* Report All Absolute addressing in code */
```

This pragma reports all absolute addressing found in code and includes reporting fixed interrupt assignments.

```
#pragma check_asm report  /* Report printed on any found asm code */
```

This pragma reports anytime it finds invalid assembly codes.

```
#pragma check_asm skip  /* All asm code skipped */
#pragma warn_absolute off
```

The pragmas can be disabled with these commands.

9 Conclusion

In most embedded control designs, there are many major factors that help to make design decisions. The decision typically depends on an accumulation of many minor, but still important, details. The 8-bit to 32-bit Freescale Controller Continuum delivers unique capabilities to system designers, in terms of optimizing power, size, performance, peripheral usage, and development tools. In addition, with the ability to seamlessly transition upward, the Freescale Controller Continuum provides companies a long term planning capability. This promises to change the product roadmaps in many industries.

In this application note we have covered some of the new CodeWarrior features and we have brought guidelines on embedded software development to help create a 100% portable code.

By following the porting tips to avoid porting mistakes and with the proper use of the new CodeWarrior features, a single application can be developed that keeps the code compiling, the peripherals working and can improve some of the project features.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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

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

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

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3465
Rev. 0
06/2007