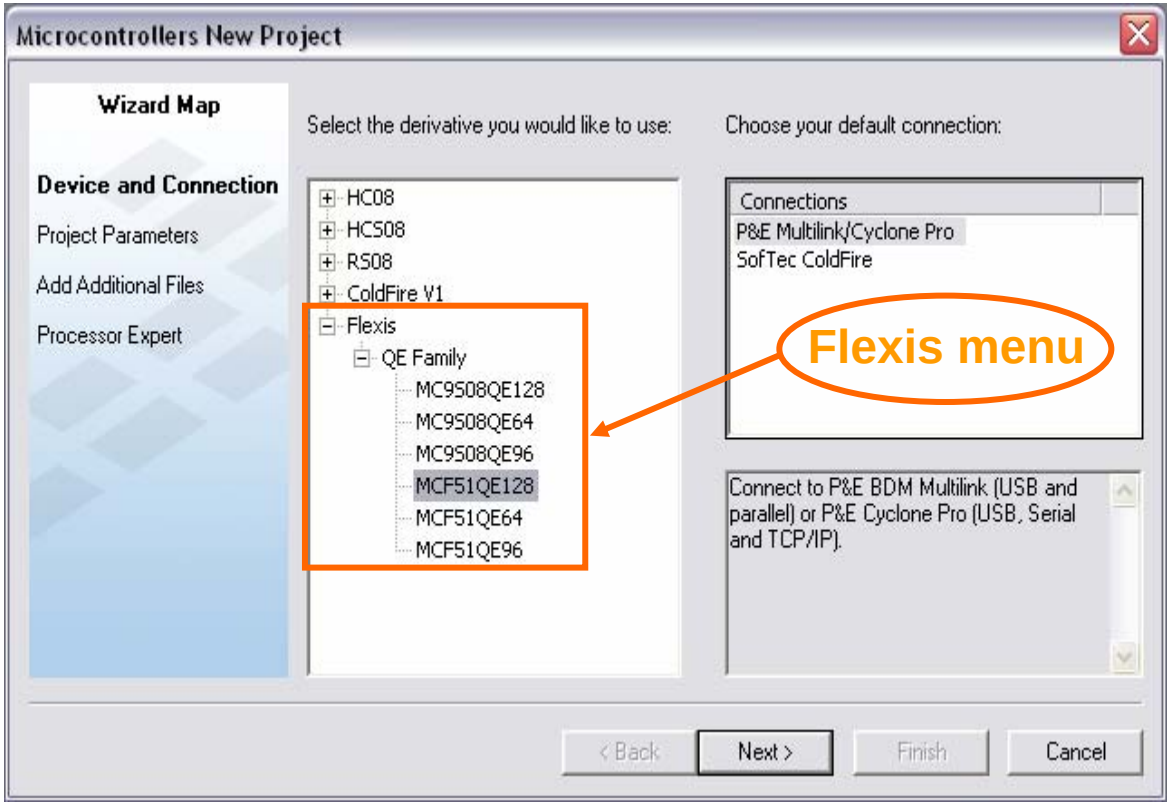
Porting Tips

Migrating from 8-bit S08 to 32-bit ColdFire V1 using CodeWarrior for Microcontrollers V6.x

Project Wizard

- **Project Wizard now includes 32-bit ColdFire V1 devices along with 8-bit HC08, HCS08, and RS08.**



Take advantage of the 1st truly compatible 8-bit HCS08 and 32-bit ColdFire V1 microcontrollers in the **Flexis** series menu

The Flexis™ Series of Microcontrollers

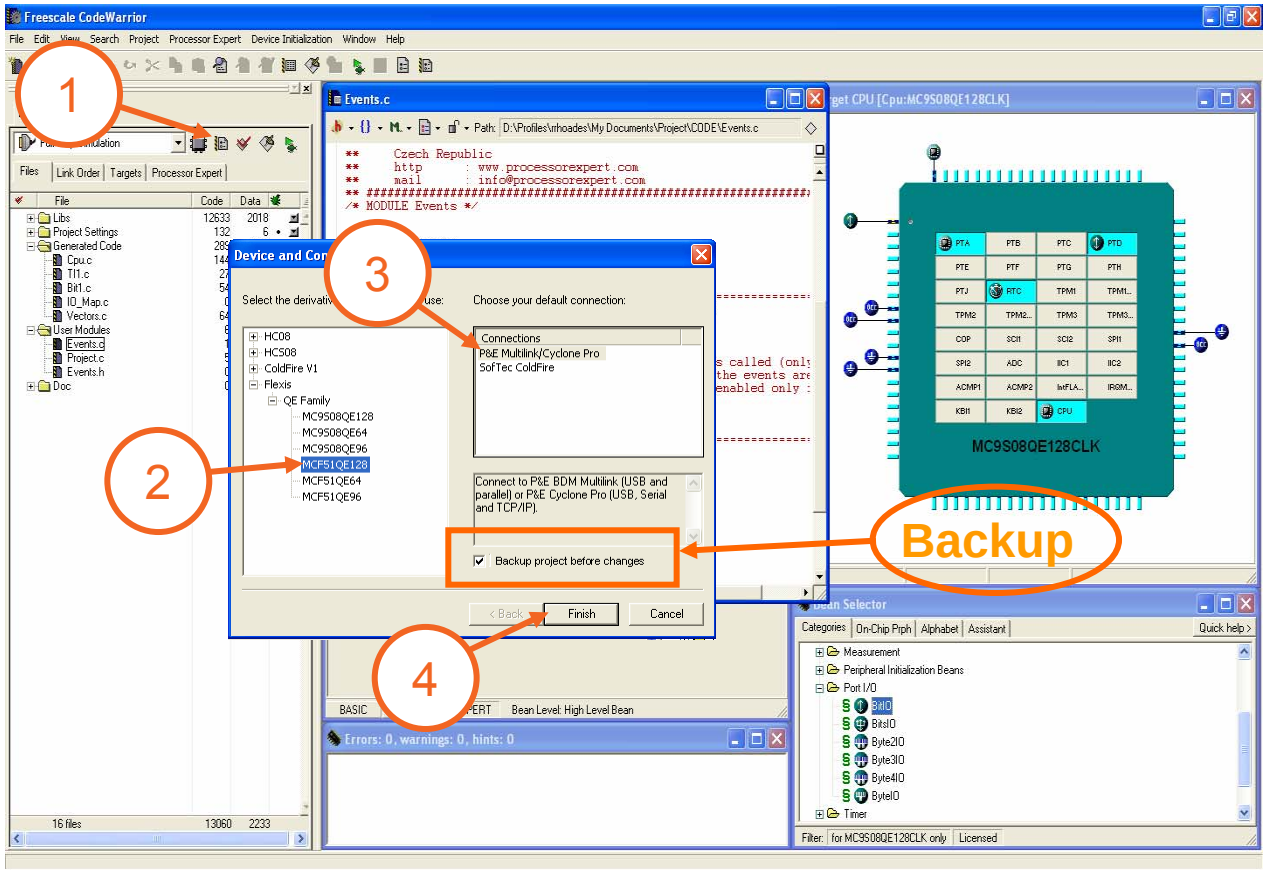
Flexis™ devices have:

- Single development tool to ease migration between 8-bit (S08) and 32-bit (ColdFire V1)
- Common peripheral set to preserve software investment between 8-bit and 32-bit
- Pin compatibility wherever practical to maximize hardware reuse when moving between 8-bit and 32-bit

freescale™ semiconductor

MCU Change Wizard

- **A project can be re-targeted from 8-bit S08 to 32-bit ColdFire V1 in as few as 4 mouse clicks.**

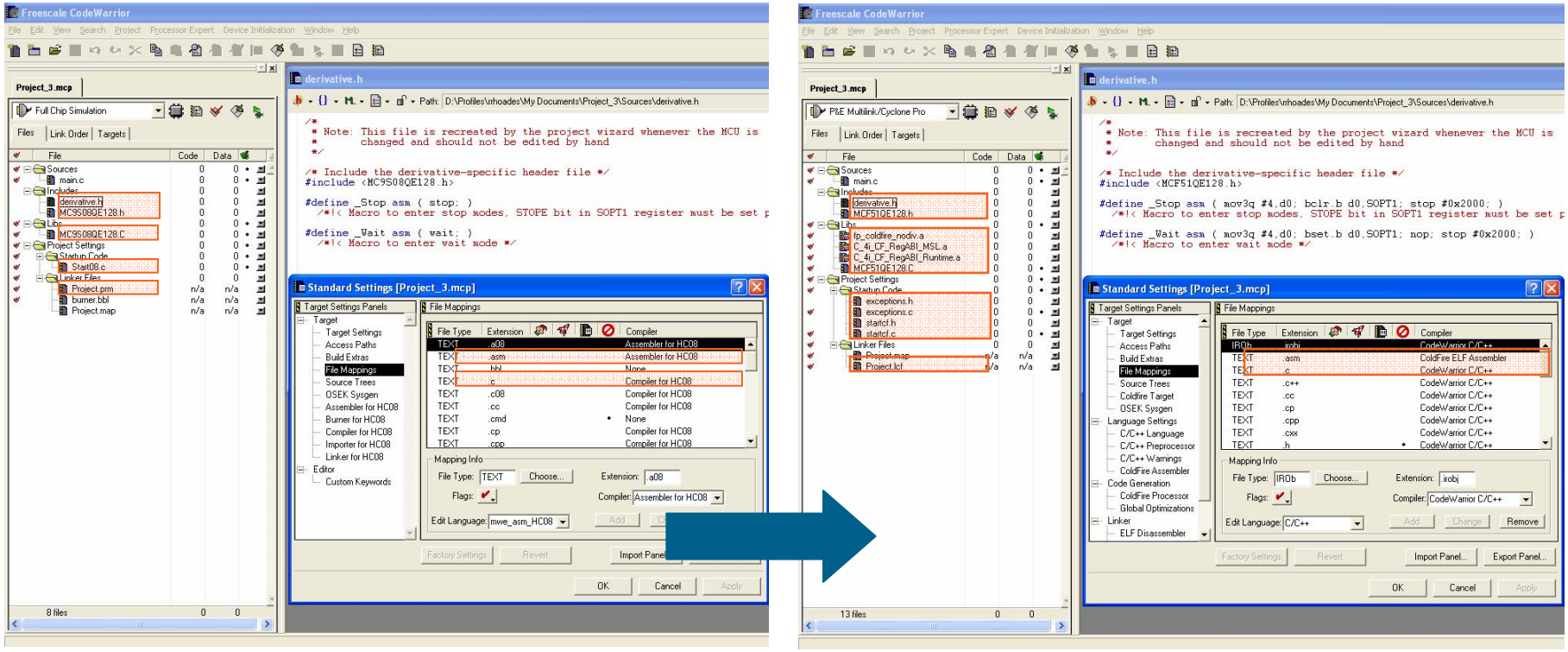


Take advantage of the **project backup** feature available in the MCU Change Wizard which saves a working copy of your project before you retarget

freescale semiconductor

MCU Change Wizard

- **In the background CodeWarrior reconfigures your project with…**



Correct Build Tools
- Assembler
- Compiler
- Linker

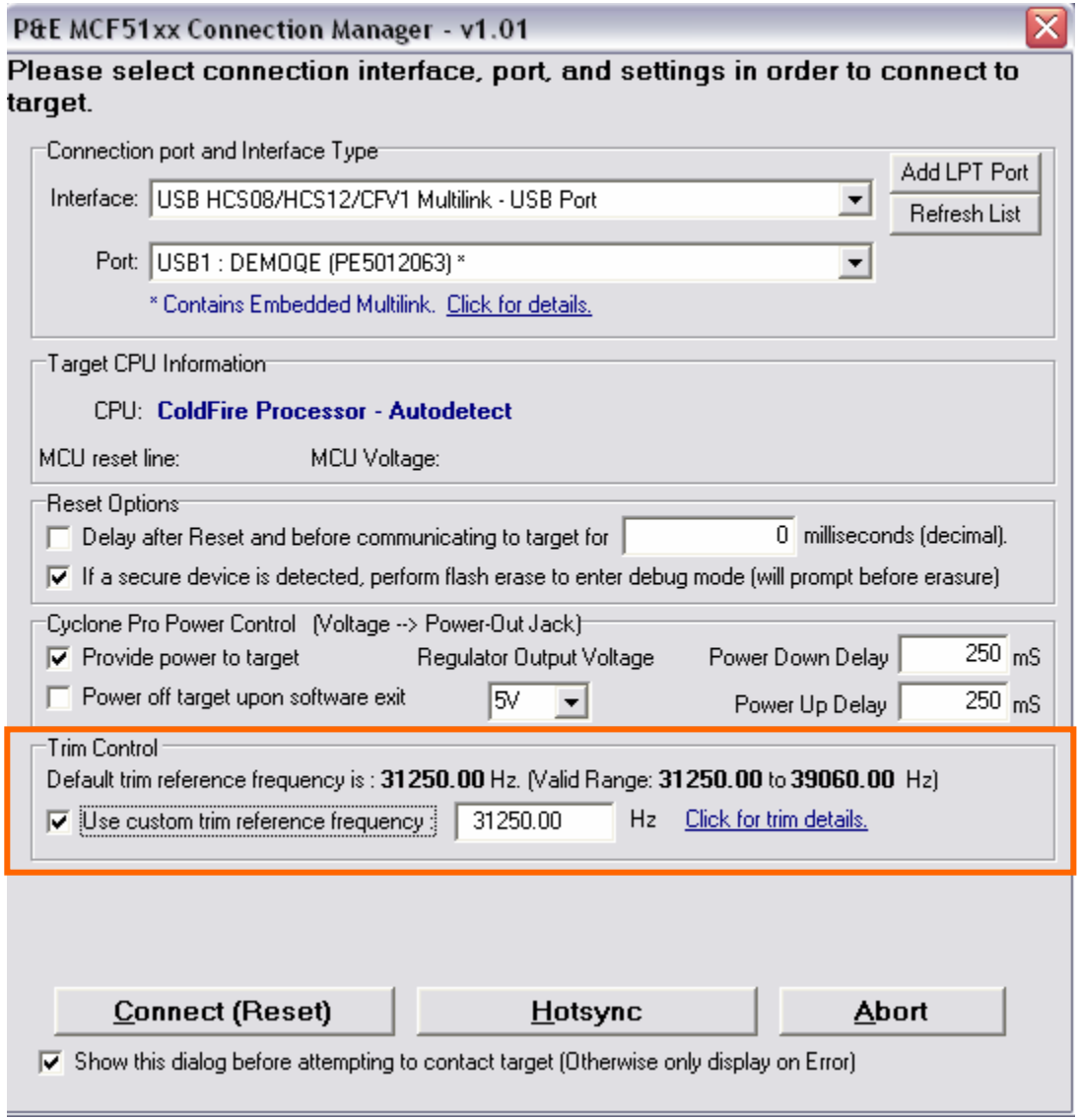
Correct Supporting Files
- Startup Code
- Libraries
- C header files
- Porting Support file

Updated derivative.h file
- New _Stop and _Wait macros

Updated map file

Updated hidef.h file
- New EnableInterrupts and DisableInterrupts macros

Auto Trim Wizard



Reference frequency trim value can vary between devices. For accuracy use the Auto Trim Wizard available in the Connection Manager then copy the trimmed value into the local MCU trim register (as shown below) and modify the clock module to output your desired bus frequency

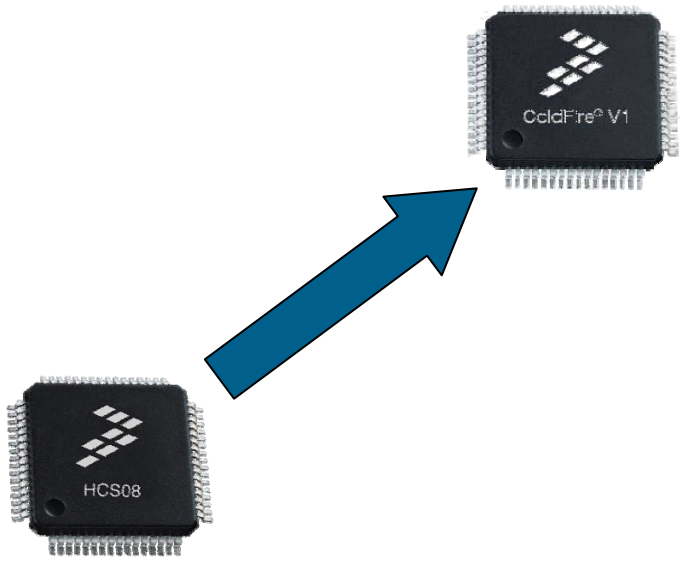
```
ICSTRM = NVICSTRM;
```


freescale semiconductor

Common Porting Mistakes

- When porting your software from 8-bit S08 to 32-bit ColdFire V1 be aware of subtle architecture differences that can affect your software operation.

  - In-line assembly instructions
  - Interrupt vector assignments
  - Entering stop and wait modes
  - Absolute memory declarations
  - Software timing



freescale semiconductor

In-line Assembly Instructions

- 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
  - Using in-line assembly can lead to a compiler error stating that the instruction operand is invalid for the chosen architecture
  - Nothing in either compiler limits you from using in-line assembly
- *Porting Tip:*
  - *Avoid in-line assembly at all costs*

Interrupt Vector Assignments

- 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 will not match up in memory space.
  - Interrupt vector assignments maintain a relative vector number between S08 and ColdFire V1
    - For MC9S08QE128, RTC interrupt (Vrtc) is located at address 0xFFCE and is vector number 24.
    - For MCF51QE128, RTC interrupt (Vrtc) is located at address 0x(00)00_0158 and is vector number 86.
    - Simple Equation: ColdFire V1 Vector Number = S08 Vector Number + 62
  - Example of improper fixed interrupt vector assignment
    - interrupt 24 RTC_ISR { } // Only works for S08

- *Porting Tip:*
  - *Assign interrupt vectors using VectorNumber_Vname interrupt declarations in CodeWarrior header files instead of using fixed vector numbers*

Interrupt Vector Assignments

MC9S08QE128

Address	Vector Number	Assignment
0xFFFE 0xFFFF	0	Reset
0xFFFC 0xFFFD	1	SWI
0xFFFA 0xFFFB	2	IRQ
0xFFF8 0xFFF9	3	Low Voltage Detect
0xFFF6 0xFFF7	4	TPM1 Channel 0
0xFFF4 0xFFF5	5	TPM1 Channel 1
0xFFF2 0xFFF3	6	TPM1 Channel 2
0xFFF0 0xFFF1	7	TPM1 Overflow
0xFFEE 0xFFEF	8	TPM2 Channel 0
0xFFEC 0xFFED	9	TPM2 Channel 1
0xFFEA 0xFFEB	10	TPM2 Channel 2
0xFFE8 0xFFE9	11	TPM2 Overflow
0xFFE6 0xFFE7	12	SPI2
0xFFE4 0xFFE5	13	SPI1
0xFFE2 0xFFE3	14	SCI1 Error
0xFFE0 0xFFE1	15	SCI1 Receive
0xFFDE 0xFFDF	16	SCI1 Transmit
0xFFDC 0xFFDD	17	IIC1 and IIC2
0xFFDA 0xFFDB	18	KBI1 and KBI2
0xFFD8 0xFFD9	19	ADC Conversoin
0xFFD6 0xFFD7	20	ACMP1 and ACMP2
0xFFD4 0xFFD5	21	SCI2 Error
0xFFD2 0xFFD3	22	SCI2 Receive
0xFFD0 0xFFD1	23	SCI2 Transmit
0xFFCE 0xFFCF	24	RTC
0xFFCC 0xFFCD	25	TPM3 Channel 0
0xFFCA 0xFFCB	26	TPM3 Channel 1
0xFFC8 0xFFC9	27	TPM3 Channel 2
0xFFC6 0xFFC7	28	TPM3 Channel 3
0xFFC4 0xFFC5	29	TPM3 Channel 4
0xFFC2 0xFFC3	30	TPM3 Channel 5
0xFFC0 0xFFC1	31	TPM3 Overflow

MCF51QE128

Address	Vector Number	Assignment
0x00000000	0	Initial Supervisor Stack Pointer
0x00000004	1	Initial Program Counter
0x00000008 - 0x000000FC	2 - 63	Reserved for internal CPU Exceptions
0x00000100	64	IRQ
0x00000104	65	Low Voltage Detect
0x00000108	66	TPM1 Channel 0
0x0000010C	67	TPM1 Channel 1
0x00000110	68	TPM1 Channel 2
0x00000114	69	TPM1 Overflow
0x00000118	70	TPM2 Channel 0
0x0000011C	71	TPM2 Channel 1
0x00000120	72	TPM2 Channel 2
0x00000124	73	TPM2 Overflow
0x00000128	74	SPI2
0x0000012C	75	SPI1
0x00000130	76	SCI1 Error
0x00000134	77	SCI1 Receive
0x00000138	78	SCI1 Transmit
0x0000013C	79	IIC1 and IIC2
0x00000140	80	KBI1 and KBI2
0x00000144	81	ADC Conversoin
0x00000148	82	ACMP1 and ACMP2
0x0000014C	83	SCI2 Error
0x00000150	84	SCI2 Receive
0x00000154	85	SCI2 Transmit
0x00000158	86	RTC
0x0000015C	87	TPM3 Channel 0
0x00000160	88	TPM3 Channel 1
0x00000164	89	TPM3 Channel 2
0x00000168	90	TPM3 Channel 3
0x0000016C	91	TPM3 Channel 4
0x00000170	92	TPM3 Channel 5
0x00000174	93	TPM3 Overflow
0x00000178 - 0x0000017C	94 - 95	Reserved; unused for V1
0x00000180	96	Level 7 SW Interrupt
0x00000184	97	Level 6 SW Interrupt
0x00000188	98	Level 5 SW Interrupt
0x0000018C	99	Level 4 SW Interrupt
0x00000190	100	Level 3 SW Interrupt
0x00000194	101	Level 2 SW Interrupt
0x00000198	102	Level 1 SW Interrupt
0x0000019C	103 - 255	Reserved; unused for V1

freescale semiconductor

Interrupt Vector Assignments

- We now need to change the interrupt vector number…

S08

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


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

ColdFire V1

```
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 VectorNumber_Vname 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;
}
```


freescale semiconductor

Entering Stop and Wait Modes

- The assembly instruction for entering stop mode between the 8-bit S08 and 32-bit ColdFire V1 are slightly different.
- 32-bit ColdFire V1 instruction set does not have a “wait” instruction to enter wait mode. Therefore, the core looks for WAITE bit in SOPT1 register to be set prior to the “stop #0x2000” instruction.

- *Porting Tip:*
  - *Use predefined project level _Stop and _Wait declarations in CodeWarrior derivative.h file to enter stop and wait modes*

```
#define _Stop asm ( stop; )
#define _Wait asm ( wait; )
```

S08

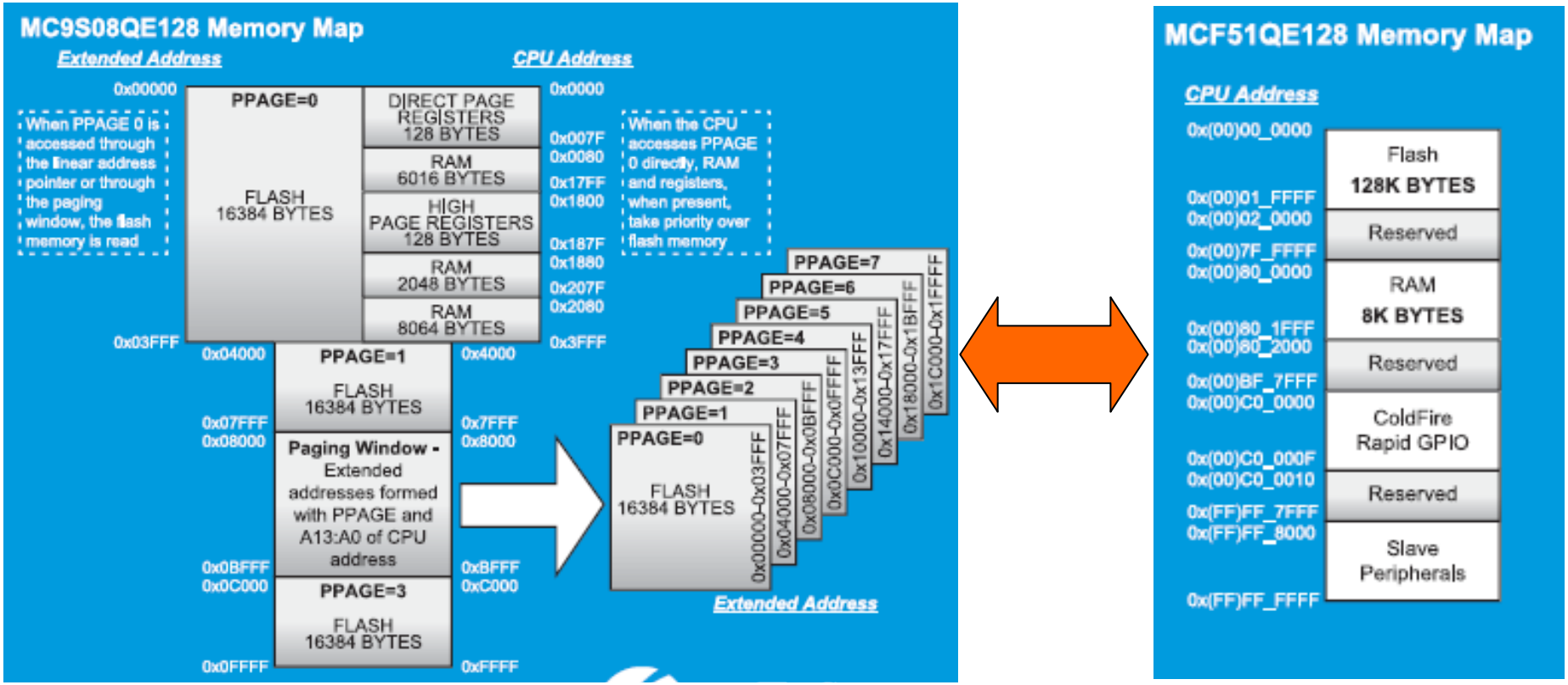
```
#define _Stop asm ( mov3q #4,d0; bclr.b d0,SOPT1; stop #0x2000; )
#define _Wait asm ( mov3q #4,d0; bset.b d0,SOPT1; nop; stop #0x2000; )
```

ColdFire V1

freescale semiconductor

Absolute Memory Declarations

- Memory maps between the 8-bit S08 and 32-bit ColdFire V1 are different, therefore absolute memory declarations will not match up in memory space.



freescale semiconductor

Absolute Memory Declarations

- RAM and FLASH memory allocation is determined by respective linker files.
- Peripheral register maps maintain relative addresses between S08 and ColdFire V1
  - For MC9S08QE128, SCI1C1 register is located at address 0x0022
  - For MCF51QE128, SCI1C1 register is located at address 0x(FF)FF_8000 + 0x0022 = 0x(FF)FF_8022
- Example of improper absolute memory declarations
  - int var @ 0x400 = 1;

Porting Tip:

- *Reference memory using Register_Bitname peripheral declarations in CodeWarrior header files and allow linker to place variables in available memory*

Software Timing

- In many software applications it is common to have software delays or timing in code.
- When migrating between 8-bit S08 and 32-bit ColdFire V1, software timing will result 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
- Example of instruction differences:
  - Popular “nop” instruction differs in cycle time between S08 and ColdFire V1

- *Porting Tip:*
  - *Avoid software delays and maintain timing through peripherals, like TPM and RTC, with a time base*

Porting Support

- A new feature in CodeWarrior that can help you port applications faster are "Porting Tips" built-in the compiler that can be controlled by user through new pragma instructions in porting_support.h:

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

  - This pragma will report all absolute addressing found in code.
  - This includes reporting fixed interrupt assignments.

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

  - This pragma will report anytime it finds invalid assembly code.

- These pragmas can also be disabled with these commands:

```
#pragma check_asm skip  /* All asm code skipped, no error for invalid instruction */
#pragma warn_absolute off
```

Programming for the Controller Continuum

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

  - Avoid in-line assembly at all costs
  - Assign interrupt vectors using VectorNumber_Vname interrupt declarations in CodeWarrior header files instead of using fixed vector numbers
  - Use pre-defined project level _Stop and _Wait declarations in CodeWarrior derivative file to enter stop and wait modes
  - Reference memory using Register_Bitname peripheral declarations in CodeWarrior header files and allow linker to place variables in available memory
  - Avoid software delays and maintain timing through peripherals, like TPM and RTC, with a time base


freescale™
semiconductor