Freescale Semiconductor
Application Note

Document Number: AN3732
Rev. 0, 06/2008

Migrating Between MC9S08AC and MCF51AC Flexis Devices

by: Bruno Castelucci
Bruno Nunes
RTAC, Americas

This application note describes how to migrate from the MC9S08AC device to the MCF51AC device. As the need to add new features accelerates, engineers face the challenging task of migrating applications from 8-bit to 32-bit microcontrollers (MCU). The AC family of devices is part of the Flexis series, which has a single development tool, common peripheral set, and pin-to-pin compatibility to reduce hardware/software investment and maximize reuse when moving between 8-bit and 32-bit.

Although most peripherals are identical in this family, there are differences between 8-bit and 32-bit architectures. This document helps designers develop applications for easy migration between 8-bit and 32-bit devices.

This application note covers the architectural differences between 8-bit and 32-bit AC 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, describing how the tool simplifies the porting experience between S08 and ColdFire V1 microcontrollers.

Contents

freescale™ semiconductor

1 Migrating Between 8-Bit and 32-Bit

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

Table 1. 8-Bit vs. 32-Bit

8-Bit	32-Bit
The core is not as complex.	Increased memory footprint and a wider portfolio of devices.
Devices 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.	Applications demand more performance.
	The gap between 32-bit prices and 8-bit prices is getting smaller.

A difficulty that can be found when migrating from an 8-bit to a 32-bit application 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. Hardware tools are commonly different; this implies extra cost and development time. The configuration of peripherals can get as complicated as the complexity gap between an 8-bit 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.

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

The Flexis series of microcontrollers have a single development tool to ease migration between 8-bit (S08) and 32-bit (CFV1). The same CodeWarrior version supports both cores. There is a common peripheral set to preserve software investment between 8-bit and 32-bit. The Flexis family has practical pin compatibility to maximize hardware reuse when moving between 8-bit and 32-bit.

2 AC Hardware Comparison

Figure 1 shows the hardware architecture the S08 (MC9S08AC) and ColdFire V1 (MCF51AC) share. This sharing makes the migrations easy for applications engineers and developers.

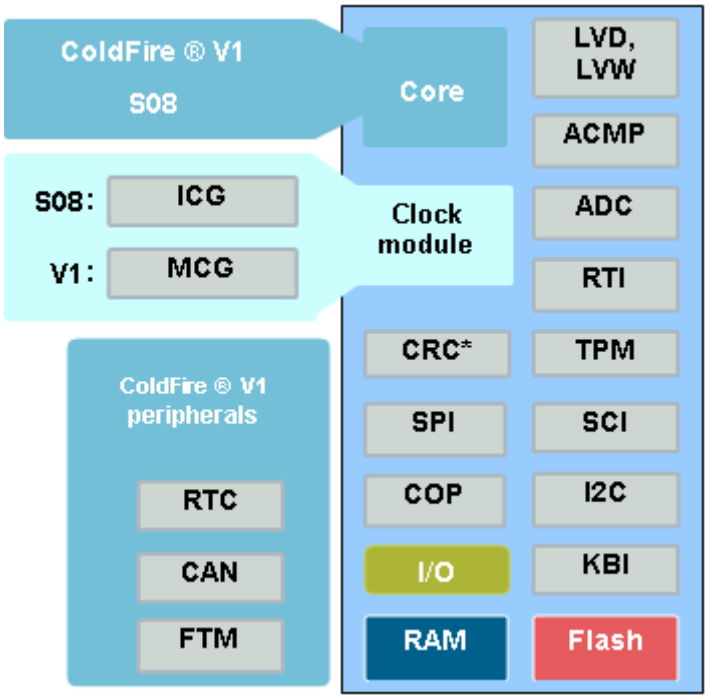

Figure 1. AC Devices Hardware Comparison

Although the CPU is different, a common set of peripherals is shared between the AC devices. This includes the ADC, serial modules, ACMP and others. The ColdFire device also has RTC, CAN and FTM modules that are not present in the S08. Besides the CPU, some other modules are different such as clock and timer modules that will be carefully reviewed in next sections.

The AC devices are pin-to-pin compatible, although some pin-count versions are not available for some devices. In Table 2 you can find the pin-count AC version availability.

Table 2. AC Devices Pin-Count Availability

Device	80	64	48	44	32
MC9S08AC8/16			X	X	X
MC9S08AC32/48/60		X	X	X	X
MC9S08AC96/128	X	X		X	
MCF51AC128/256	X	X			

3 Common Porting Issues

When porting the software from 8-bit MC9S08AC to 32-bit ColdFire MCF51ACV1AC be aware of subtle architecture differences that can affect the software operation. In this section, these architectural differences will be shown along with how to avoid some mistakes. Other more complex porting issues (clock and timer) that must also be considered will be presented in other sections of this application note.

3.1 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. For example:

```
asm BCLR 0,PTBDD
```

Must be replaced by:

```
PTBDD_PTBDD0=0;
```

3.2 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 will not match up in memory space because vector numbers are different. For example:

For the MC9S08AC, the RTI interrupt is vector number 29.

```
#define VectorNumber_Vrti                29
```

For the MCF51AC, the RTI interrupt is vector number 91.

```
#define VectorNumber_Vrti                91
```

A common mistake is to use the following improper interrupt assignment:

```
interrupt 29 RTI_ISR()
```

This assignment works only for S08. Then instead of going back and forth when using the S08 and ColdFire V1, use the vector numbers defined by CodeWarrior. A solution is to replace with:

```
interrupt VectorNumber_Vrti void RTI_ISR()
```

3.3 Porting Tip 3: Reference Memory Using Register_Bitname Peripheral Declarations in CodeWarrior Header Files

The on-chip memory in the MC9S08AC and MCF51AC series of MCUs consist of RAM, flash program memory for nonvolatile data storage, plus I/O and control/status registers.

These areas are located in different address in MC9S08AC and MCF51AC, as we can see in the Figure 2.

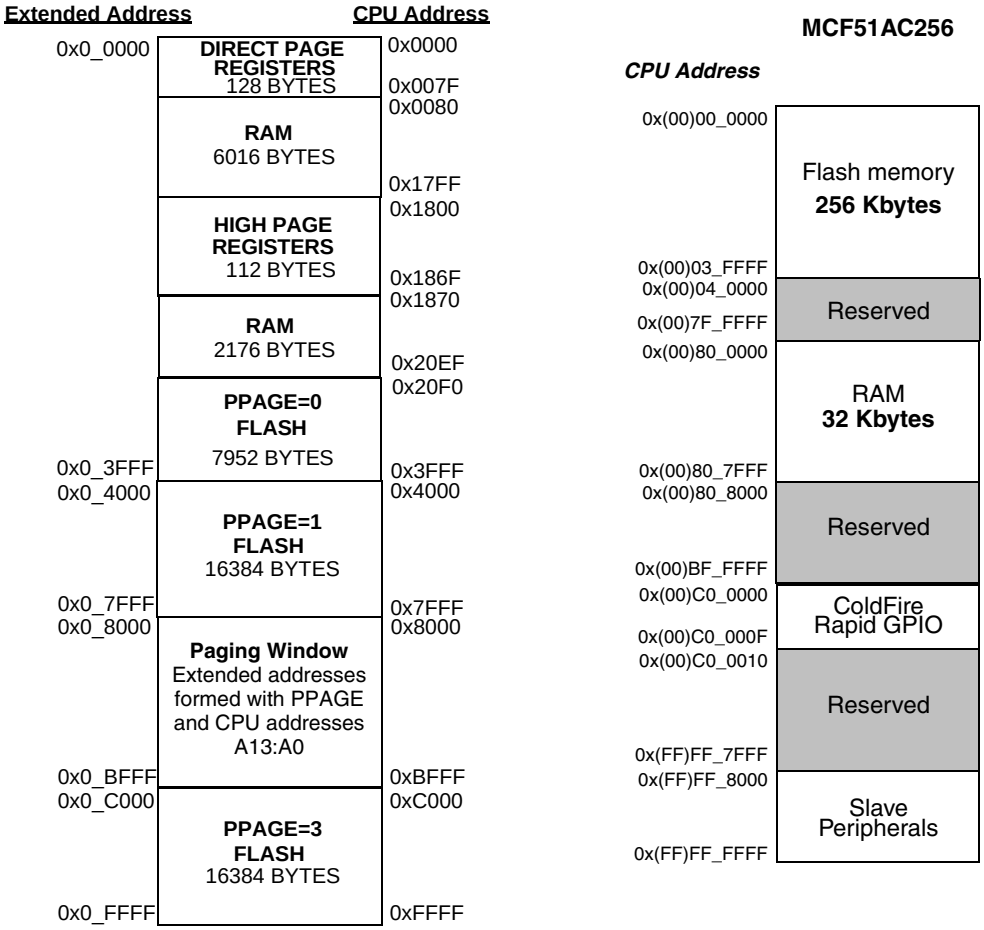

Figure 2. Different Memory Maps

The S08 architecture uses a 16-bit data bus. That means the maximum memory addressable is more than 64 KB, so for the MC9S08AC96 and MC9S08AC128 there is a need to use the paging window and a MMU (memory management unit). That isn't necessary in the CF V1 device after the data bus is 32-bit.

For further reference on memory paging for S08 devices, refer to AN3730 available at the Freescale website www.freescale.com.

In both devices, RAM and flash memory allocation is determined by respective linker files.

A good programming practice is to make reference to memory using register_bitname peripheral declarations in CodeWarrior header files (for example: PTAD_PTAD0) and allow the linker to place variables in available memory and remove absolute address.

An example of improper absolute memory declaration is:

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

In this case, the code would work for the S08 device, after the 0x80 address is RAM area, but it wouldn't work for the CFV1 because the 0x80 is a flash memory area.

To avoid that mistake, the best practice is to use the linker to allocate the variable, so the code should be replaced by:

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

3.4 Porting Tip 4: Avoid Software Delays and Maintain Timing through Peripherals

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. These devices execute at a different frequency: ColdFire V1 instructions execute at CPU frequency, and S08 instructions execute at bus clock frequency.

The cycle time required by some instructions differs depending on the platform (S08 or ColdFire) used. For this reason, it is necessary to avoid software delays and use a common time base through peripherals, like TPM, FTM, and RTC.

3.5 Porting Support

A useful 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 */
```

The pragmas can be disabled with these commands.

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

4 Clock Module Migration

4.1 Clock Module Differences

The clock module of MC9S08AC is called ICG (internal clock generator). For the Coldfire V1 this module is called MCG (multipurpose clock generator). The integrated clock module provides multiple options for clock source and in-application clock switching.

The MC9S08AC ICG provides multiple options for clock sources. This offers a user great flexibility when making choices between cost, precision, current draw, and performance. The ICG consists of four functional blocks: oscillator, internal reference generator, frequency-locked loop (FLL), and clock select blocks.

The multipurpose clock generator (MCG) module provides several clock source choices for the MCF51AC. The module contains a frequency-locked loop (FLL) and a phase-locked loop (PLL) that are controllable by either an internal or an external reference clock. The module can select either of the FLL or PLL clocks, or either of the internal or external reference clocks as a source for the MCU system clock. The selected clock source is passed through a reduced bus divider which allows a lower output clock frequency to be derived. The MCG also controls a crystal oscillator (XOSC), which allows an external crystal, ceramic resonator, or another external clock source to produce the external reference clock.

There are differences between modes of operation supported by ICG and MCG modules, Table 1 and Table 2 show these modes and provide a brief description of each one.

ICG Modes

- Off — The output clock, ICGOUT, is static. This mode may be entered when the STOP instruction is executed.
- Self-clocked (SCM) — Default mode of operation that is entered immediately after reset. The ICG's FLL is open loop and the digitally controlled oscillator (DCO) is free running at a frequency set by the filter bits.
- FLL engaged internal (FEI) — In this mode, the ICG's FLL is used to create frequencies that are programmable multiples of the internal reference clock.
- FLL bypassed external (FBE) — In this mode, the ICG is configured to bypass the FLL and use an external clock as the clock source.
- FLL engaged external (FEE) — The ICG's FLL is used to generate frequencies that are programmable multiples of the external clock reference.

MCG Modes

- Stop — Entered whenever the MCU enters a stop state.
- FLL engaged internal (FEI) (default) — MCGOUT is derived from the FLL clock, which is controlled by the internal reference clock.
- FLL engaged external (FEE) — MCGOUT is derived from the FLL clock, which is controlled by the external reference clock.
- FLL bypassed internal (FBI) — MCGOUT is derived from the internal reference clock; the FLL is operational, but its output clock is not used. This mode is useful to allow the FLL to acquire its target frequency while the MCGOUT clock is driven from the internal reference clock.
- FLL bypassed external (FBE) — MCGOUT is derived from the external reference clock; the FLL is operational, but its output clock is not used. This mode is useful to allow the FLL to acquire its target frequency while MCGOUT is driven from the external reference clock.
- PLL engaged external (PEE) — MCGOUT is derived from the PLL clock, which is controlled by the external reference clock.

- PLL bypassed external (PBE) — MCGOUT is derived from the external reference clock; the PLL is operational, but its output clock is not used. This mode is useful to allow the PLL to acquire its target frequency while MCGOUT is driven from the external reference clock.
- Bypassed low power internal (BLPI) — MCGOUT is derived from the internal reference clock. The PLL and FLL are disabled, and MCGLCLK is not available for BDC communications.
- Bypassed low power external (BLPE) — MCGOUT is derived from the external reference clock. The PLL and FLL are disabled, and MCGLCLK is not available for BDC communications.

For reference, the register map for both clock modules can be found in the next lines.

The ICG module registers are:

- ICG Control Register 1 (ICGC1)
- ICG Control Register 2 (ICGC2)
- ICG Status Register 1 (ICGS1)
- ICG Status Register 2 (ICGS2)
- ICG Filter Registers (ICGFLTU, ICGFLTL)
- ICG Trim Register (ICGTRM).

The MCG module has the following registers:

- MCG Control Register 1 (MCGC1)
- MCG Control Register 2 (MCGC2)
- MCG Control Register 3 (MCGC3)
- MCG Control Register 4 (MCGC4)
- MCG Status and Control Register (MCGSC)
- MCG Test Register (MCGT)

For further reference on clock modules details, please refer to corresponding data sheets and reference manuals.

4.2 Clock Code Examples

To ease the ICG to MCG migration path, in this section you can find examples using ICG and MCG modules. It will be provided code for both modules, using internal and external clock references.

NOTE

The Device Initialization tool in CodeWarrior is also a powerful tool to help in the migration. With this tool you can visually configure both modules, and the code necessary will be automatically generated.

4.2.1 Internal Clock Configuration

In the following initialization function Int_S08_ICG_init(), the frequency FLL will be used (in FEI mode) to multiply the internal 243 kHz (approximate) reference clock up to 40 MHz to achieve 20 MHz bus frequency for the MC9S08AC device.

```
void Int_S08_ICG_init(){

   ICGC1 = 0x4C;
/*********************************************************************
Bit 7 HGO 0 Configures oscillator for low power
Bit 6 RANGE 1 Configures oscillator for high-frequency range; FLL prescale factor is 1
Bit 5 REFS 1 Oscillator using crystal or resonator requested (bit is really a don't care)
Bits 4:3 CLKS 01 FLL engaged, internal reference clock mode
Bit 2 OSCSTEN 1 Enables the oscillator
Bit 1 LOCD 0 Loss-of-clock enabled
Bit 0 0 Unimplemented or reserved, always reads zero
*********************************************************************/


ICGC2 = 0x70;
/*********************************************************************
Bit 7 LOLRE 0 Generates an interrupt request on loss of lock
Bit 6:4 MFD 111 Sets the MFD multiplication factor to 18
Bit 3 LOCRE 0 Generates an interrupt request on loss of clock
Bit 2:0 RFD 000 Sets the RFD division factor to ÷1
*********************************************************************/

ICGTRM = *(unsigned char*far)0xFFBE;
/* Initialize ICGTRM register from a non volatile memory */
  while(!ICGS1_LOCK) {                /* Wait until FLL is locked*/
  }
}
```

In the Int_V1_MCG_init() function below, the FLL will be used (FEI) to multiply the internal 32 kHz to achieve a 20 MHz bus frequency. This system will also use the trim function to fine tune the frequency based on nonvolatile memory position.

```
void Int_V1_MCG_init(){

/*  System clock initialization */
  MCGTRM = *(unsigned char*far)0x03FF; /* Initialize MCGTRM register from a nonvolatile memory */
  MCGSC = *(unsigned char*far)0x03FE;  /* Initialize MCGSC register from a nonvolatile memory */

  MCGC2 = 0x00;                       /* Set MCGC2 register */
/*********************************************************************
Bit 7:6 BDIV 00 – Divides selected clock by 1
Bit 5 RANGE 0 low frequency range selected for the crystal oscillator of 32 kHz to 100 kHz
Bit 4 HGO High Gain Oscillator Select – 0 Configure crystal oscillator for low power operation
Bit 3 Power Select – 0 FLL (or PLL) is not disabled in bypass modes.
Bit 2 EREFS 0  not used here
Bit 1 ERCLKEN 0 not used
Bit 0 EREFSTEN 0  not used
*********************************************************************/

  MCGC1 = 0x06;                       /* Set MCGC1 register */
/*********************************************************************
Bit 7:6 CLKS 00  Output of FLL or PLL is selected.
Bit 5:3 RDIV not used
Bit 2 IREFS  1 Internal reference clock selected
Bit 1 IRCLKEN 1 Enables the internal reference clock for use as MCGIRCLK.
Bit 0 IREFSTEN 0 0 Internal reference clock is disabled in stop
*********************************************************************/
```

```
  MCGC3 = 0x81;                         /* Set MCGC3 register */
/*********************************************************************
Bit 7 LOLIE 1 Generate an interrupt request on loss of lock.
Bit 6 PLLS 0 FLL is selected
Bit 5 CME 0 Clock monitor is disabled.
Bit 4 DIV32 0 Divide-by-32 is disabled.
Bit 3:0 VDIV VCO Divider – not used
*********************************************************************/

MCGC4 = 0x21;                         /* Set MCGC4 register */
/*********************************************************************
Bit 7:6 0 Reserved.
Bit 5 DMX32 1 DCO is fined tuned for maximum frequency with 32.768 kHz reference.
Bit 4:2 0 reserved
Bit 1 DRST: indicate the current frequency range for the FLL output, DCOOUT.
Bit 0 DRS select the frequency range for the FLL output:
01Mid range
*********************************************************************/

  while(!MCGSC_LOCK) {                  /* Wait until FLL is locked */
  }
}
```

4.2.2 External Clock Configuration

In the following initialization function Ext_S08_ICG_init(), a 4 MHz external crystal oscillator was used to generate a 16 MHz bus clock frequency.

```
void Ext_V1_ICG_init(){

  ICGC1 = 0xFC;
/*********************************************************************
Bit 7 HGO 1 Configures oscillator for high gain
Bit 6 RANGE 1 Configures oscillator for high-frequency range; FLL prescale factor is 1
Bit 5 REFS 1 Oscillator using crystal or resonator requested
Bits 4:3 CLKS 11 FLL engaged, external reference
Bit 2 OSCSTEN 1 Enables the oscillator
Bit 1 LOCD 0 Loss-of-clock enabled
Bit 0 0 Unimplemented or reserved, always reads zero
*********************************************************************/

  ICGC2 = 0x20;
/*********************************************************************
Bit 7 LOLRE 0 Generates an interrupt request on loss of lock
Bit 6:4 MFD 010 Sets the MFD multiplication factor to 8
Bit 3 LOCRE 0 Generates an interrupt request on loss of clock
Bit 2:0 RFD 000 Sets the RFD division factor to ÷1
*********************************************************************/
  while(!ICGS1_LOCK) {                  /* Wait until FLL is locked*/
  }

}
```

The same was made using the MCG module for the MCF51AC device in the Ext_V1_MCG_init() function below.

```
void Ext_V1_MCG_init(){
```

```
MCGC2 = 0x36;                         /* Set MCGC2 register */
/*********************************************************************
Bit 7:6 BDIV 00 – Divides selected clock by 1
Bit 5 RANGE 1 High frequency range selected for the crystal oscillator of 1 MHz to 40 MHz for external
clock source.
Bit 4 HGO High Gain Oscillator Select – 1 Configure crystal oscillator for high gain operation
Bit 3 Power Select – 0 FLL (or PLL) is not disabled in bypass modes.
Bit 2 EREFS 1 Oscillator requested
Bit 1 ERCLKEN 1 External Reference Enable – Enables the external reference
Bit 0 EREFSTEN 0 External Reference Stop Enable- External reference clock is disabled in stop mode.
*********************************************************************/
MCGC3 |= (unsigned char)0x10;

  MCGC1 = 0x10;                        /* Set MCGC1 register */
/*********************************************************************
Bit 7:6 CLKS 00  Output of FLL is selected.
Bit 5:3 RDIV 010 External Reference Divider by 1
Bit 2 IREFS  1  External reference clock selected
Bit 1 IRCLKEN 0 not used
Bit 0 IREFSTEN 0  not used
*********************************************************************/
  MCGC3 = 0x91;                         /* Set MCGC3 register */
/*********************************************************************
Bit 7 LOLIE 1 Generate an interrupt request on loss of lock.
Bit 6 PLLS 0 FLL is selected
Bit 5 CME 0 Clock monitor is disabled.
Bit 4 DIV32 1 Divide-by-32 is enabled.
Bit 3:0 VDIV 0001 VCO Divider – not used
*********************************************************************/
   MCGC4 = 0x01;                         /* Set MCGC4 register */
/*********************************************************************
Bit 7:6 0 Reserved.
Bit 5 DMX32 0 DCO isn't fined tuned
Bit 4:2 0 reserved
Bit 1 DRST 0 : indicate the current frequency range for the FLL output, DCOOUT.
Bit 0 DRS select the frequency range for the FLL output:
01Mid range
*********************************************************************/
while(MCGSC_IREFST);                 /* Wait until external reference is selected */

  while(!MCGSC_LOCK) ;                 /* Wait until FLL is locked */

  while((MCGSC & 0x0C) != 0x00) {      /* Wait until FLL clock is selected as a bus clock
reference */
  }

}
```

5 Timer Module Migration

The 8-bit MC9S08AC devices have three TPM (timer and PWM module). The 32-bit MCF51AC devices have two FTM (FlexTimer module) and also one TPM (timer and PWM module).

5.1 Timer Modules Features

The FTM module has the same features as the TPM module and others such as dead time insertion and fault detection. The Table 3 shows a comparison of the two different timer modules' features.

Table 3. Timer Modules Features

FTM Features Include	TPM Features Include
• Backwards compatible with TPM	
• FTM source clock is selectable — Source clock can be the system clock, the fixed system clock, or a clock from an external pin — External clock pin may be shared with any FTM channel pin or a separated input pin • Prescaler divide-by 1, 2, 4, 8, 16, 32, 64, or 128 — External clock source is synchronized to the system clock by FTM	• TPM source clock is selectable — Source clock can be the system clock, the fixed system clock or a clock from an external pin — External clock pin may be shared with any TPM channel pin or a separated input pin • Prescaler divide-by 1, 2, 4, 8, 16, 32, 64, or 128
• FTM has a 16-bit counter — It can be a free-running counter or a counter with initial and final value — The counting can be up or up-down	• TPM has a 16-bit counter — It can be a free-running counter or a counter modulo final value — The counting can be up or up-down
• The generation of one interrupt per channel • The generation of one interrupt in the end of the counting	• The generation of one interrupt per channel • The generation of one interrupt in the end of the counting
• Each channel can be configured for input capture, output compare, or edge-aligned PWM mode • In input capture mode — the capture can occur on rising edges, falling edges, or both edges — an input filter can be selected for some channels • In output compare mode the output signal can be set, cleared or toggled on match • All channels can be configured for center-aligned PWM mode • Each pair of channels can be combined to generate a PWM signal (with independent control of both edges of PWM signal) • The FTM channels can operate as pairs with equal outputs, pairs with complementary outputs, or independent channels (with independent outputs) • The deadtime insertion is available for each complementary pair • Generation of triggers to ADC (hardware trigger) • Software control of PWM outputs • A fault input for global fault control • The polarity of each channel is configurable • The load of the FTM registers that have write buffers can be synchronized • Write protection for critical registers	• Each channel can be configured for input capture, output compare, or edge-aligned PWM mode • In input capture mode — the capture can occur on rising edges, falling edges, or both edges • In output compare mode the output signal can be set, cleared, or toggled on match • All channels can be configured for center-aligned PWM mode • Each TPM may be configured for buffered, center-aligned pulse-width modulation (CPWM) on all channels

5.2 Timer Modules in Each AC Version

Some of the 9S08AC MCU devices have timer modules with more or fewer channels than others to reduce price and best fit in each market application. Table 4 shows how many and what timer modules are present in each device, separated by pin count.

Table 4. Timer Modules on AC Devices

Device	80	64	48	44	32
MC9S08AC8/16	—	—	TPM1 – 4 ch TPM2 – 2 ch TPM3 – 2 ch	TPM1 – 4 ch TPM2 – 2 ch TPM3 – 2 ch	TPM1 – 2 ch TPM2 – 2 ch TPM3 – 2 ch
MC9S08AC32/48/60	—	TPM1 – 6 ch TPM2 – 2 ch TPM3 – 2 ch	TPM1 – 4 ch TPM2 – 2 ch TPM3 – 2 ch	TPM1 – 4 ch TPM2 – 2 ch TPM3 – 2 ch	TPM1 – 2 ch TPM2 – 2 ch TPM3 – 2 ch
MC9S08AC96/128	TPM1 – 6 ch TPM2 – 6 ch TPM3 – 2 ch	TPM1 – 6 ch TPM2 – 2 ch TPM3 – 2 ch	—	TPM1 – 4 ch TPM2 – 2 ch TPM3 – 2 ch	—
MCF51AC128/256	FTM1 – 6 ch FTM2 – 6 ch TPM3 – 2 ch	FTM1 – 6 ch FTM2 – 2 ch TPM3 – 2 ch	—	—	—

5.3 Timer Modules — Migrating within 9S08AC Devices

The FTM module is backwards compatible with TPM. This means that FTM has all the TPM features.

For full compatibility between both timer modules we should consider a project using only TPM features, made for any of the 9S08AC or MCF51AC devices (even if using the FTM).

All the TPM registers are compatible with the FTM registers, but as the timer modules are different, the nomenclature changes. For example if we consider the TPM Status and Control Register (TPMxSC), there is a compatible FTM register named FTM Status and Control Register (FTMxSC) with the same control bits and options from the TPM register.

When migrating between an 8-bit 9S08AC device to a 32-bit one, the nomenclature changes should be considered and reviewed so that the correspondent timer module works properly.

To ease this migration, all the registers in Table 5 should be considered. This table shows the correspondent register in 8-bit and 32-bit Flexis AC devices.

Table 5. Timer Modules' Registers

TPM	FTM	Definition
TPMxSC	FTMxSC	Status and Control
TPMxCNTH	FTMxCNTH	Counter Register High
TPMxCNTL	FTMxCNTL	Counter Register Low
TPMxMODH	FTMxMODH	Modulo Register High
TPMxMODL	FTMxMODL	Modulo Register Low
TPMxC0SC	FTMxC0SC	Channel 0 Status and Control
TPMxC0VH	FTMxC0VH	Channel 0 Value High
TPMxC0VL	FTMxC0VL	Channel 0 Value Low

To ease the migration path, the header files described in the next chapter were created.

NOTE

The Device Initialization tool in CodeWarrior is also a powerful tool to help in the migration. With this tool you can visually configure both modules, and the necessary code will be automatically generated.

5.4 Header Files

The header files created with this document are useful when migrating your application between 8-bit and 32-bit Flexis AC devices, especially when the timer module is used.

There are two header files. The TimerH_S08toV1.h is to be used when migrating from an MC9S08AC device to an MCF51AC. The TimerH_V1toS08.h file is to be used when migrating from an MCF51AC to an MC9S08AC.

The header files are composed of #defines where the FTM registers are mapped using the nomenclature from the TPM registers (and vice-versa for the other header file), as shown in the example for the TimerH_S08toV1.h header file:

```
#define TPM1SC FTM1SC
```

The header file that contains this example is used when migrating from the MC9S08AC to the MCF51AC. In this case, the old TPM registers that were defined to work with the TPM module on the MC9S08AC device will be automatically understood by the compiler as the new FTM registers for the MCF51AC, and the project should migrate with no further problems.

Both header files contain all the compatible timer registers, and also all the bit definitions inside these registers. You can find the full header files in the file AN3732.zip available at Freescale website www.freescale.com .

To use these header files in your project, you must include them in the .c file you use timer registers, not only in initialization routines but also in all use of timer registers. To successfully include the files, you should place them in your project folder and then add a include as follows in the beginning of your .c files:

```
#include "TimerH_S08toV1.h"
```

6 Conclusion

The 8-bit to 32-bit Freescale Controller Continuum delivers unique capabilities to system designers, in terms of power optimization, size, performance, peripheral usage, and development tools. By following the porting tips such as being aware of clock and timer module differences and proper use of the new CodeWarrior features, a single application that compiles in 8- and 32-bit architectures can be developed, with working peripherals and improvements to some of the project features. The Flexis series is the 8- to 32-bit connection point in the Freescale Controller Continuum where S08 and ColdFire V1 microcontrollers share common sets of peripherals and development tools to deliver the ultimate in migration flexibility. In addition, with the ability to seamlessly transition upward, the Freescale Controller Continuum provides companies a long term planning capability. It is possible to move quickly from an 8-bit design to a 32-bit design — perfect for developing a portfolio of products that span the performance spectrum.

Appendix A MCU Change Wizard

The MCU change wizard in CodeWarrior for microcontrollers allows you to change 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 derivative to migrate the application.
3. Select the connection mode.
4. Click on the finish button.

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

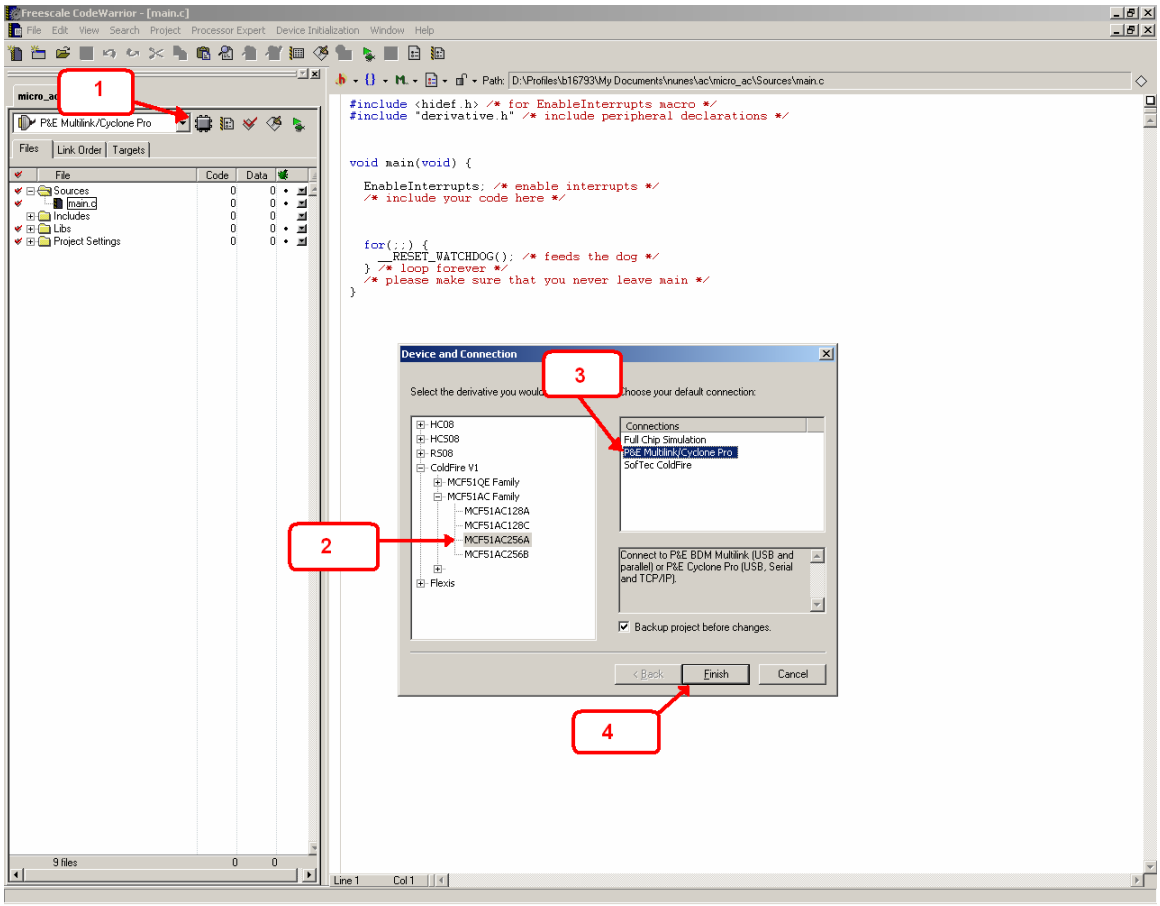

Figure 3. MCU Change Wizard

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

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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
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 +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3732
Rev. 0
06/2008

freescale™
semiconductor