# Migrating Code Between ColdFire V1 and V2

by: Alfredo Soto
   TIC Mexico, RTAC Americas
   Daniel Torres
   Go to Market, RTAC Americas

# 1 Introduction

In today's designs, easily changing a device is important to provide better performance or to reduce complexity or power consumption.

It is very common to find an embedded software development fully programmed in C. Having software tools such as compilers and integrated development environments (IDE) that allow you to use a single language to program different devices is a good start; however, when migrating between microcontrollers you also need to learn about the new peripherals, functionality, and development support.

The new ColdFire® V1 device is a subset of the V2 architecture and gives a new connection point to migrate your applications when looking for different features.

## 1.1 Abstract

As part of the Freescale Controller Continuum, this cross reference guides you through migrating from the

**Contents**

*freescale*™
semiconductor

ColdFire V1 family to the ColdFire V2. Although the V1 architecture is a subset of the V2 architecture, there are some considerations to keep in mind when migrating from one to another. This application note covers the initialization of the different common peripherals to allow the reuse of the rest of an application.

## 1.2    Objective

This document details how to migrate C code of the common peripherals used in the MCF51QE128 to MCF52210, such as the analog-to-digital converter, timers, serial  communications interface, serial peripheral interface, inter-integrated circuit, and general-purpose input/output. This document also describes the interrupt processing in both architectures.

Example code is provided for the common peripherals between the MCF51QE128 and the MCF52210; with this, a C programmer can easily adapt the peripheral initialization from one device to the other and migrate between a ColdFire V1 and a ColdFire V2 with less effort.

# 2    ColdFire V1 and V2 Overview

As of June 2007, existing devices in the MCF51QE family have these common peripherals related to the V2: analog-to-digital converter, timers, serial  communication interface, serial peripheral interface, inter-integrated circuit, and general-purpose input/output. This does not imply V1 and V2 peripheral modules are the same. This application note describes how to easily migrate between the devices and applies to future MCF51x and MCF52x peripherals.

The MCF51QE128 and MCF51QE64 are members of the low-cost, low-power, high-performance Version 1 (V1) ColdFire family of 32-bit microcontroller units (MCUs). All MCUs in the family use the enhanced V1 ColdFire core and are available with a variety of modules, memory sizes, memory types, and packages. CPU clock rates on these devices can reach 50.33 MHz. Peripherals operate up to 25.165 MHz.

The ColdFire V1 MCF51QE device-family features up to 128 KB of flash memory, up to 8 KB of RAM, two analog comparators (ACMP), up to 24 analog-to-digital channels (ADC), two inter-integrated circuit (IIC) modules, one keyboard interrupt (KBI) module, up to 70 general-purpose input/output (GPIO) terminals, real-time counter (RTC), two serial communications interface (SCI) modules, two serial peripheral interface (SPI) ports, three timer/pulse-width modulator (TPM) modules.

The MCF5221x represents a family of highly-integrated 32-bit microcontrollers based on the V2 ColdFire microarchitecture. Featuring 16 KB of internal SRAM and 128 KB of flash memory, four 32-bit timers with DMA request capability, a 4-channel DMA controller, two IIC modules, three UARTs, and a queued SPI, the MCF5221x family is designed for general-purpose industrial control applications. This 32-bit device is based on the Version 2 (V2) ColdFire reduced instruction set computing (RISC) core with a multiply-accumulate unit (MAC) and divider providing 76 Dhrystone 2.1 MIPS at a frequency up to 80 MHz from internal flash.

This application note provides explanations and code examples that help migrate between a ColdFire V1 and a V2, allowing a user to initialize the peripherals in the new device and keep the main routines of the application without changes.

## 2.1 General-Purpose Input/Outputs

Many of the pins associated with the external interface in both the ColdFire's V1 MC51FQE128 processors and the ColdFire's V2 MCF52211 processors may be used for several functions. When not used for their primary function, many of the pins may be used as general-purpose digital I/O pins. In some cases, the pin function is set by the operating mode and the alternate pin functions are not supported.

The ColdFire V1 MC51QE family rapid GPIO (RGPIO) module provides a 16-bit general-purpose I/O module directly connected to the processor's high-speed 32-bit local platform bus. This connection to the processor's high-speed platform bus plus support for single-cycle, zero wait-state data transfers allows the RGPIO module to provide improved pin performance when compared to more traditional GPIO modules located on the internal slave peripheral bus.

Many of the pins associated with a device may be used for several functions. Their primary functions are to provide external interfaces to access off-chip resources. When not used for the primary function, many of the pins may be used as general-purpose digital I/O (GPIO) pins. The definition of the exact pin functions and the affected signals is specific to each device. Every GPIO port, including the RGPIO module, has registers that configure, monitor, and control the port pins.

The key features of this module include:

- 16 bits of high-speed GPIO functionality connected to the processor's local 32-bit platform bus
- Memory-mapped device connected to the ColdFire core's local bus
    — Support for all access sizes: byte, word, and longword
    — All reads and writes complete in a single data phase cycle for zero wait-state response
- Data bits can be accessed directly or via alternate addresses to provide set, clear, and toggle functions
    — Alternate addresses allow set, clear, toggle functions using simple store operations without the need for read-modify-write references
- Unique data direction and pin enable control registers
- Package pin toggle rates typically 1.5–3.5 times faster than comparable pin mapped onto peripheral bus

The ColdFire V2 MCF5221x family digital I/O pins are grouped into 8-bit ports. Some ports do not use all eight bits. Each port has registers that configure, monitor, and control the port pins. The MCF52211 ports module controls the configuration for the following external pins:

- External bus accesses
- Chip selects
- Debug data
- Processor status
- USB
- IIC serial control
- QSPI
- UART transmit/receive
- 32-bit DMA timers

**Migrating Code Between ColdFire V1 and V2, Rev. 0, Draft C**

The MCF52211 ports includes these distinctive features:

- Control of primary function use on all ports
- Digital I/O support for all ports; registers for:
  - Storing output pin data
  - Controlling pin data direction
  - Reading current pin state
  - Setting and clearing output pin data registers

Table 1 shows the MC51QEx rapid GPIO and the MCF5221x GPIO common operating modes.

**Table 1. MC51QEx Rapid GPIO and the MCF5221x GPIO Common Operating Modes**

| Attribute | ColdFire V1 MCF51QE128 GPIO | ColdFire V2 MCF5221x GPIO |
| --- | --- | --- |
| Number of pins | Up to 70 | Up to 56 |
| Input | Yes | Yes |
| Output | Yes | Yes |
| Direct address access to bits | Yes | Yes |
| Access size | Byte, word, longword | Byte, word, longword |
| Rapid | Single-cycle, zero wait-state data transfers | No |
| Slew rate control | Yes | Yes |
| Strength control | Yes | Yes (2 mA–10 mA) |
| Interrupt request | No | No |

In Figure 1, the general-purpose I/O modules are configured as outputs. A GPIO pin toggles after a 500 ms elapsed. The set and clear registers are not used in this code example; in this case, the data register is directly modified.

Figure 1 shows the GPIO_init functions. Figure 2 shows the code required in both processors to toggle a pin.

```
void GPIO_Init(void) {
/* Configure PTE as outputs */
PTEDD |= PTEDD_PTEDD7_MASK;
/* Configure PTE as outputs */
PTED = 0x00;     /* Put 0's in PTE port */
}
```

**Figure 1A. Code Snippet for the MC51QEx Processor**

```
void GPIO_init()
{
    /*Configure Port TC as output*/
    MCF_GPIO_DDRTC  = 0
     | MCF_GPIO_DDRTC_DDRTC0;   /* pin TC0 to LED1 */
     MCF_GPIO_PORTTC = 0x00;   /* clear port TC */
}
```

**Figure 1B. Code Snippet for the MCF5221x Processor**

**Figure 1. GPIO_init Function**

```
  while(1) {
     Delay (16000);   /* 500mS     Delay*/PTED_PTED7 ^= 1;  /*Toggle Led*/
  }
}
```

**Figure 2A. Code Snippet for the MC51QExx**

```
  while (1) {
    GPT_delay(153);   /* 500mS Delay */
    MCF_GPIO_PORTTC ^= 0x01;   /* Toggle Led */
  }
 }
```

**Figure 2B. Code Snippet for the MCF5221x**

**Figure 2. Toggling a GPIO pin**

## 2.2    Timers

The ColdFire V1 MCF51QE device family includes up to three independent timer/PWM (TPM) modules that support traditional input capture, output compare, or buffered edge-aligned pulse-width modulation (PWM) on each channel. A control bit in each TPM configures all channels in that timer to operate as center-aligned PWM functions. In each of these two TPMs, timing functions are based on a separate 16-bit counter with prescaler and modulo features to control frequency and range (period between overflows) of the time reference. This timing system is ideally suited for a wide range of control applications, and the

center-aligned PWM capability on the 3-channel TPM extends the field of applications to motor control in small appliances.

The bus clock to TPM1, TPM2, and TPM3 can be gated on and off using the TPMx bits in SCGC1. These bits are set after any reset, which enables the bus clock to this module. To conserve power, these bits can be cleared to disable the clock to any of these modules when not in use.

The TPM included in the ColdFire V1 MC51QE devices has these features:

- Each TPM may be configured for buffered, center-aligned pulse-width modulation (CPWM) on all channels
- Clock sources independently selectable per TPM (multiple TPMs device)
- Selectable clock sources (device dependent): bus clock, fixed system clock, external pin
- Clock prescaler taps for divide by 1, 2, 4, 8, 16, 32, 64, or 128
- 16-bit free-running or up/down (CPWM) count operation
- 16-bit modulus register to control counter range
- Timer system enable
- One interrupt per channel plus a terminal count interrupt for each TPM module (multiple TPMs device)
- Channel features:
  — Each channel may be input capture, output compare, or buffered edge-aligned PWM
  — Rising-edge, falling-edge, or any-edge input capture trigger
  — Set, clear, or toggle output compare action
  — Selectable polarity on PWM outputs

The MCF5221x family has one 4-channel general-purpose timer module (GPT). It consists of a 16-bit counter driven by a 7-stage programmable prescaler.

A timer overflow function allows software to extend the timing capability of the system beyond the 16-bit range of the counter. Each of the four timer channels can be configured for input capture, which can capture the time of a selected transition edge, or for output compare, which can generate output waveforms and timer software delays. These functions allow simultaneous input waveform measurements and output waveform generation.

Additionally, channel 3 can be configured as a 16-bit pulse accumulator that can operate as a simple event counter or as a gated time accumulator. The pulse accumulator uses the GPT channel 3 input/output pin in event mode or gated time accumulation mode.

Features of the general-purpose timer include:

- Four 16-bit input capture/output compare channels
- 16-bit architecture
- Programmable prescaler
- Pulse-widths variable from microseconds to seconds
- Single 16-bit pulse accumulator
- Toggle-on-overflow feature for pulse-width modulator (PWM) generation

- External timer clock input (SYNCA/SYNCB)

Table 2 shows the TPM and GPT common operating modes.

**Table 2. TPM and GPT Common Operating Modes**

| Attribute | ColdFire V1 MC51QE TPM | ColdFire V2 MCF5221x GPT |
|---|---|---|
| 16-bit architecture | Yes | Yes |
| 16-bit free running counter | Yes | Yes |
| Programmable prescaler | Divides bus clock by 1, 2, 4, 8, 16, 32, 64, or 128 (25 MHz max clock) | Divides the module clock by 1 or 16 (40 MHz max clock) |
| Input capture | Rising or falling edges | Rising or falling edges |
| Output compare | Set, clear, or toggle the channel pin | Set, clear, or toggle the channel pin |
| Number of channels | Up to 10 channels | Up to 4 channels |
| Pulse-width modulation capabilities | Yes, center- or edge-aligned | Toggle-on-overflow feature for pulse-width modulator (PWM) generation |
| External clock support | Yes (<0.5 $f_{bus}$) | Yes |
| Interrupts | Channel interrupts on input capture and output compare, timer overflow | Channel interrupts on input capture and output compare, timer overflow |

Figure 3 is a code example showing the migration of a C code developed for both timers configured to generate a 500 ms delay. This example code configures the timers in a simple manner without the use of interrupt flags. Both timers are configured in output compare mode. Each timer toggles the channel pin after the timer overflow flag has been set, meaning that the timer has reached the counter modulo value. Figure 3 shows the main function.

```
void main(void) {
  MCU_Init();                    /* Function that initializes the MCU */
  GPIO_Init();                   /* Initializes the Ports of the MCU  */
  TPWM_configuration();          /* Initializes the TPM module */
   while(1) {
      Delay (16000);             /* 500mS Delay*/
      PTED_PTED7 ^= 1;           /*Toggling a GPIO */
   }
}
```

**Figure 3A. Code Snippet for the MC51QEx Processor**

```
void main(void)
{
    PLL_init();                 /* Configure PLL */

    GPIO_init();                /* Configure LEDs  on MCF52211EVB*/

    GPT_init();                 /* Configure timer */

    while (1) {
        GPT_delay(153);         /* 500mS Delay */
        MCF_GPIO_PORTTC ^= 0x01;  /* togglin GPIO */
    }
}
```

**Figure 3B. Code Snippet for the MCF5221x Processor**

**Figure 3. Timer Code Example**

First, you must configure the MCU to disable watchdog timer, to enable the background and reset pin, and to provide clock to the TPM or GPT modules. The ColdFire V2 MCF5221x family requires the initialization of the PLL to clock down the bus clock so the GPT timer can produce a 500 ms delay. Figure 4 and Figure 5 show this code snippet. Figure 4 refers to the V1 QE family and Figure 5 refers to V2 MCF5221x family

```
void MCU_Init(void) {
    SOPT1 =(SOPT1_STOPE_MASK | SOPT1_RSTPE_MASK
      /* Watchdog disable. Stop Mode Enable.  */
    | SOPT1_BKGDPE_MASK);
      /* Background Pin enable. RESET pin enable */
    SCGC1 = SCGC1_TPM1_MASK;
      /* Bus Clock to the TPM1 module is enabled */
}
```

**Figure 4. Code Snippet for the MC51QEx Processor, MCU_init Function**

**Migrating Code Between ColdFire V1 and V2, Rev. 0, Draft C**

```
void PLL_init(void ) {
      /*Configure PLL with selected frequency*/
      MCF_CLOCK_SYNCR =  MCF_CLOCK_SYNCR_PLLMODE
              /* PLL in programming mode */
              | MCF_CLOCK_SYNCR_CLKSRC
              /* PLL output drives system clock */
              | MCF_CLOCK_SYNCR_RFD(1)
              /* divider PLL / 2 */
              | MCF_CLOCK_SYNCR_MFD(3)
              /* multiplier PLL x18 */
              | MCF_CLOCK_SYNCR_PLLEN;
              /* enabling PLL */
}
```

**Figure 5. Code Snippet for the MCF5221x Processor, PLL_init Function.**

The GPIOs are used in this example as a user interface, demonstrating that the time delay generated with the timers corresponds to approximately 500 ms.

The next step is to configure the timer modules, achieved in the GPT_init functions. To generate a 500 ms delay, you must use the prescalers. In the case of the V1 QE ColdFire family, the bus clock is divided by 128. For the V2 MCF5221x family, the clock divider is higher than that since the V2 bus clock is running at a higher frequency. This is shown in Table 6.

```
void TPWM_configuration (void)
{
    /* TPM clock source is: Bus rate clock divided by 128 */
    TPM1SC = (TPM1SC_PS_MASK | TPM1SC_CLKSA_MASK);
}
```

**Figure 6A. Code Snippet for the MC51QEx Processor**

```
void GPT_init( void )
{
      /* timer selected as output compare */
      MCF_GPT_GPTIOS |= MCF_GPT_GPTIOS_IOS0;
      /* GPT disconnected from output pin logic */
      MCF_GPT_GPTCTL1 = 0;
}
```

**Figure 6B. Code Snippet for the MCF5221x Processor**

**Figure 6. Timer Init Function**

Because the timers are not configured to generate an interrupt, the main program polls the timer-overflow flag to verify that the timer has reached to its maximum value. The polling is performed in the delay function as shown in Figure 7. After the maximum timer module value has been reached, the GPIO pin is toggled to demonstrate that a 500 ms elapsed, the TOF is cleared, and the timer is disabled. Figure 7 shows the code snippet.; Figure 7A refers to the V1 QE family and Figure 7B refers to V2 MCF5221x family.

```
Delay(UINT16 compare) {
  /* The counter counts up to compare value */
  TPM1MOD = compare;
  while (!TPM1SC_TOF);
  /* clear overflow flag */
  TPM1SC_TOF = 0;
}
```

**Figure 7A. Code Snippet for the MC51QEx Processor**

```
void GPT_delay(unsigned int time)  {
  unsigned int u32longCounter;
  /* turn on timer counter */
  MCF_GPT_GPTSCR1 = MCF_GPT_GPTSCR1_GPTEN;
  /* counter's time */
  for(u32longCounter = 0; u32longCounter < time; u32longCounter++) {
  /* Stay until timer overflow flag asserted */
      while( !(MCF_GPT_GPTFLG2 & MCF_GPT_GPTFLG2_TOF) ) ;
            /* clear overflow flag */
      MCF_GPT_GPTFLG2 = MCF_GPT_GPTFLG2_TOF;
  }
  /* turn off timer counter */
  MCF_GPT_GPTSCR1 &= (~MCF_GPT_GPTSCR1_GPTEN);
}
```

**Figure 7B. Code Snippet for the MC5221x Processor**

**Figure 7. Approx. 500 ms Delay Function**

## 2.3    Inter-Integrated Circuit (IIC)

The Coldfire V1, MCF51QE128 has up to two IIC modules. The IIC interface is designed to operate up to 100 kbps, with maximum bus loading and timing. The device is capable of operating at higher baud rates, up to a maximum of clock/20, with reduced bus loading. The IIC1 module pins, SDA and SCL can be repositioned under software control using the SOPT2 register, thus adding flexibility to this module.

The ColdFire V2 contains up to two IIC modules. The modules operate up to 100 kbps, with maximum bus loading and timing. The device can operate at higher baud rates, up to a maximum of the internal bus clock divided by 20, with reduced bus loading.

Table 3 shows the most important features for both MCU families.

**Table 3. Features of V1 MC51QE and V2 MCF5221x**

| IIC Features | V1 MC51QE | V2 MCF5221x |
|---|---|---|
| Compatible with IIC bus standard | Y | Y |
| Multiple-master operation | Y | Y |
| Software-programmable clock frequency | Y | Y |
| Software-selectable acknowledge bit | Y | Y |
| Interrupt-driven, byte-by-byte data transfer | Y | Y |

**Table 3. Features of V1 MC51QE and V2 MCF5221x (continued)**

| IIC Features | V1 MC51QE | V2 MCF5221x |
|---|:---:|:---:|
| Arbitration-lost interrupt with automatic mode switching from master to slave | Y | Y |
| Calling address identification interrupt | Y | Y |
| START and STOP signal generation/detection | Y | Y |
| Repeated START signal generation | Y | Y |
| Acknowledge bit generation/detection | Y | Y |
| Bus-busy detection | Y | Y |
| General call recognition | Y | Y |
| 10-bit address extension | Y | N |

Figure 8 shows example routines for using the IIC modules on the ColdFire V1 and V2, respectively. The first code of every code snippet is for the ColdFire V1 MC51QEx family and the second is for the ColdFire V2 MCF5221x.

The next routines show the IIC initialization. The IIC frequency is set to approximately 400 kHz. If the IIC bus is busy, a stop condition is sent.

**Migrating Code Between ColdFire V1 and V2, Rev. 0, Draft C**

```
void I2CInit (void) {
  unsigned char temp;
  IIC1F = 0x22;   /* set the frequency near 400 kHz*/
  IIC1C1  = 0 | IIC1C1_IICEN;   /* start the module */
    /* if bit busy set, send a stop condition to slave module */
if(IIC1S & IIC1S_BUSY) {
  IIC1C1 = 0;    /* clear control register */
  IIC1C1 = IIC1C1_IICEN |   /* enable module */
  IIC1C1_MST;   /* send a START condition */
  temp = IIC1D;   /* dummy read */
  IIC1S = 0;   /* clear status register */
  IIC1C1 = 0;   /* clear control register */
  IIC1C1= 0 | IIC1C1_IICEN;   /* enable the module again */
  }
}
```

**Figure 8A. Code Snippet for Initializing the IIC Module in the V1 MC51QExx Processor**

```
void I2Cinit(void) {
   uint8 temp;
     /* I2C pins configuration */
MCF_GPIO_PQSPAR = 0 | MCF_GPIO_PQSPAR_PQSPAR3(2)
                 | MCF_GPIO_PQSPAR_PQSPAR2(2);

/* set the frequency near 400 kHz, see MCF5213RM table for details */
   MCF_I2C0_I2FDR = MCF_I2C_I2FDR_IC(0x32);
   MCF_I2C0_I2CR = 0 | MCF_I2C_I2CR_IEN;  /* start the module */
     /* if bit busy set, send a stop condition to slave module */
   if( MCF_I2C0_I2SR & MCF_I2C_I2SR_IBB) {
       MCF_I2C0_I2CR = 0;   /* clear control register */
       MCF_I2C0_I2CR = MCF_I2C_I2CR_IEN |   /* enable module */
              MCF_I2C_I2CR_MSTA /* send a START condition*/
              temp = MCF_I2C0_I2DR; /* dummy read */
   MCF_I2C0_I2SR = 0;   /* clear status register */
   MCF_I2C0_I2CR = 0;   /* clear control register */
/* enable the module  again */
  MCF_I2C0_I2CR = 0 | MCF_I2C_I2CR_IEN;
     }
}
```

**Figure 8B. Code Snippet for Initializing the IIC Module in the V2 MCF5221x**

**Figure 8. Initializing the IIC Module**

Figure 9 shows the routine for sending a byte to an IIC device. It writes the specified data byte in the address specified. The code example was developed to read and write a serial EEPROM memory (part number 24C04) using the IIC module.

```
voidI2CSendByte(unsigned char data, unsigned char   address, unsigned char id)
{
   unsigned char Temp;
   IIC1C1 |= IIC1C1_TX;/* setting in Tx mode */
   /* generates start condition */
   IIC1C1 |= IIC1C1_MST;
   IIC1D  = id;   /* set device ID to write */
   /* wait until one byte transfer completion */
   while(!(IIC1S & IIC1S_IICIF));
   /* clear the completion transfer flag */
   IIC1S &= ~IIC1S_IICIF;
   IIC1D = address;   /* memory address */
   /* wait until one byte transfer completion */
   while(!(IIC1S & IIC1S_IICIF));
   /* clear the completion transfer flag */
   IIC1S &= ~IIC1S_IICIF;
   IIC1D = data;     /* memory data */
   /* wait until one byte transfer completion */
   while(!(IIC1S & IIC1S_IICIF));
   /* clear the completion transfer flag */
   IIC1S &= ~IIC1S_IICIF;
   /* generates stop condition */
   IIC1C1 &= ~IIC1C1_MST;
}
```

**Figure 9A. Code Snippet Showing How to Send a Byte Over the IIC Bus Using the MC51QEx Processor**

```
 void I2CSendByte(uint8 data, uint8 address, uint8 id)
{
   MCF_I2C0_I2CR |= MCF_I2C_I2CR_MTX;  /* setting in Tx mode */

     /* generates start condition */

  MCF_I2C0_I2CR |= MCF_I2C_I2CR_MSTA;

   MCF_I2C0_I2DR = id; /* set decide ID to write */

    /* wait until one byte transfer completion */

   while( !(MCF_I2C0_I2SR & MCF_I2C_I2SR_IIF ));

    /* clear the completion transfer flag */

   MCF_I2C0_I2SR &= ~MCF_I2C_I2SR_IIF;

   MCF_I2C0_I2DR = address;/* memory address */

    /* wait until one byte transfer completion */

   while( !(MCF_I2C0_I2SR & MCF_I2C_I2SR_IIF ));

    /* clear the completion transfer flag */

   MCF_I2C0_I2SR &= ~MCF_I2C_I2SR_IIF;

   MCF_I2C0_I2DR = data;/* memory data */

   /* wait until one byte transfer completion */

   while( !(MCF_I2C0_I2SR & MCF_I2C_I2SR_IIF ));

    /* clear the completion transfer flag */

   MCF_I2C0_I2SR &= ~MCF_I2C_I2SR_IIF;
    /* generates stop condition */
   MCF_I2C0_I2CR &= ~MCF_I2C_I2CR_MSTA;
}
```

**Figure 9B. Code Snippet Showing How to Send a Byte Over the IIC Bus Using the MCF5221x Processor**

**Figure 9. Sending a Byte Over the IIC Bus**

**Migrating Code Between ColdFire V1 and V2, Rev. 0, Draft C**

Figure 10 shows the function used to receive a byte. First, it writes the address of the data to read and then it receives the data from the slave device.

```
unsigned char I2CReceiveByte(unsigned char address, unsigned char id)  {
  unsigned char data;
  IIC1C1 |= IIC1C1_TX;                     /* setting in Tx mode */
  IIC1C1 |= IIC1C1_MST;                    /* send start condition */
  IIC1D = id;                              /* device ID to write */
      /*Wait until one byte transfer completion */
while(!(IIC1S & IIC1S_IICIF ));
IIC1C1 &= ~IIC1S_IICIF;                    /* clear the completion transfer flag */
IIC1D = address;                           /* memory address */
   /* wait until one byte transfer completion */
while(!(IIC1S & IIC1S_IICIF ));            /* clear the completion transfer flag */
IIC1S &= ~IIC1S_IICIF;
IIC1C1 |= IIC1C1_RSTA;                     /* resend start */
IIC1D = id | 0X01;                         /* device id to read */
   /* wait until one byte transfer completion */
while(!(IIC1S & IIC1S_IICIF ));
IIC1S &= ~IIC1S_IICIF;
IIC1C1 &= ~IIC1C1_TX;                      /* setting in Rx mode */
IIC1C1 |= IIC1C1_TXAK;                     /* send NO ACK */
data = IIC1D;                              /* dummy read */
/* wait until one byte transfer completion */
while(!(IIC1S & IIC1S_IICIF ));
/* clear the completion transfer flag */
IIC1C1 &= ~IIC1S_IICIF;
data = IIC1D;                              /* read data received */
/* wait until one byte transfer completion */
while( !(IIC1S & IIC1S_IICIF ));
IIC1S &= ~IIC1S_IICIF;                     /* clear the completion transfer flag */
IIC1C1 &= ~IIC1C1_MST;                     /* generates stop condition */
return data;                              /* send the received data */
}
```

**Figure 10A. Code Snippet Showing How to Receive a Byte Over the IIC Bus Using the MC51QEx Processor**

```
 uint8 I2CReceiveByte(uint8 address, uint8 id)
{
   uint8 data;
   MCF_I2C0_I2CR |= MCF_I2C_I2CR_MTX;                  /* setting in Tx mode */

   MCF_I2C0_I2CR |= MCF_I2C_I2CR_MSTA;                 /* send start condition */

   MCF_I2C0_I2DR = id;                                 /* device ID to write */

  /* wait until one byte transfer completion */
   while( !(MCF_I2C0_I2SR & MCF_I2C_I2SR_IIF ));

   MCF_I2C0_I2SR &= ~MCF_I2C_I2SR_IIF;                 /* clear the transfer flag */

   MCF_I2C0_I2DR = address;                            /* memory address */

  /* wait until one byte transfer completion */

  while( !(MCF_I2C0_I2SR & MCF_I2C_I2SR_IIF ));

   MCF_I2C0_I2SR &= ~MCF_I2C_I2SR_IIF;                 /* clear the transfer flag */
   MCF_I2C0_I2CR |= MCF_I2C_I2CR_RSTA;                 /* resend start */

   MCF_I2C0_I2DR = id | 0x01;                          /* device id to read */

   /* wait until one byte transfer completion */
   while( !(MCF_I2C0_I2SR & MCF_I2C_I2SR_IIF ));

 MCF_I2C0_I2SR &= ~MCF_I2C_I2SR_IIF;                   /* clear the transfer flag */

 MCF_I2C0_I2CR &= ~MCF_I2C_I2CR_MTX;                   /* setting in Rx mode */

 MCF_I2C0_I2CR |= MCF_I2C_I2CR_TXAK;                   /* send NO ACK */

 data = MCF_I2C0_I2DR;                                 /* dummy read */
/* wait until one byte transfer completion */
while( !(MCF_I2C0_I2SR & MCF_I2C_I2SR_IIF ));

MCF_I2C0_I2SR &= ~MCF_I2C_I2SR_IIF;                    /* clear the transfer flag */

data = MCF_I2C0_I2DR;/* read data received */
/* wait until one byte transfer completion */
while( !(MCF_I2C0_I2SR & MCF_I2C_I2SR_IIF ));

MCF_I2C0_I2SR &= ~MCF_I2C_I2SR_IIF;                    /* clear the transfer flag */
MCF_I2C0_I2CR &= ~MCF_I2C_I2CR_MSTA;                   /* generates stop condition */

return data;                                          /* send the received data */

}
```

**Figure 10B. Code Snippet Showing How to Receive a Byte Over the IIC Bus Using the MCF5221x Processor**

**Figure 10. Receiving a Byte Over the IIC Bus**

## 2.4 Analog-to-Digital Converter (ADC)

Table 4 shows the main features of the ADC modules for the ColdFire V1 and V2.

The ADC in the ColdFIre V1 MC51QExx family can perform an analog-to-digital conversion on any of the software selectable channels. In 12-bit and 10-bit mode, the selected channel voltage is converted by a successive approximation algorithm into a 12-bit digital result. In 8-bit mode, the selected channel voltage is converted by a successive approximation algorithm into a 9-bit digital result.

When the conversion is completed, the result is placed in the data registers (ADCRH and ADCRL). In 10-bit mode, the result is rounded to 10 bits and placed in the data registers (ADCRH and ADCRL). In 8-bit mode, the result is rounded to eight bits and placed in ADCRL. The conversion complete flag (COCO) is then set and an interrupt is generated if the conversion complete interrupt has been enabled (AIEN = 1). The ADC module can automatically compare the result of a conversion with the contents of its compare registers. The compare function is enabled by setting the ACFE bit and operates in conjunction with any of the conversion modes and configurations.

The MCF5221x ADC's conversion process is initiated by a sync signal from one of two input pins (SYNCx) or by writing 1 to a STARTn bit in the control register. Starting a single conversion actually begins a sequence of conversions or a scan of up to eight single-ended or differential samples one at a time in sequential scan mode.

Scan sequence is determined by defining eight sample slots in ADC listing (ADLST1/2) registers, processed in order SAMPLE0–7 during sequential scan, or in order SAMPLE0–3 by converter A and in order SAMPLE4–7 by converter B in parallel scan. SAMPLE slots may be disabled using the SDIS register. The following pairs of analog inputs can be configured as a differential pair: AN0–1, AN2–3, AN4–5, and AN6–7. When configured as a differential pair, a reference to either member of the differential pair by a sample slot results in a differential measurement using that differential pair.

The ADC can be configured to perform a single scan and halt, perform a scan when triggered, or perform the scan sequence repeatedly until manually stopped. The single scan (once mode) differs from the triggered mode only in that SYNC input signals must be re-armed after each using a once mode scan, and subsequent SYNC inputs are ignored until the SYNC input is re-armed. This arming can occur anytime after the SYNC pulse occurs, even while the scan it initiated remains in process.

Optional interrupts can be generated at the end of a scan sequence. Interrupts are available to indicate the scan ended, that a sample was out of range, or at several different zero crossing conditions. Out-of-range is determined by the high and low limit registers.

**Table 4. Features of the ADC Modules for the ColdFire V1 and V2**

| ADC Feature | V1 | V2 |
|---|---|---|
| Resolution | 12-bit | 12-bit |
| Channels | 24 | 8 |
| Conversion time | 2.5 µs | 1.125 µs |
| Automatic compare function | Y | N |
| Internal temperature sensor | Y | N |
| Internal bandgap reference channel | Y | N |

**Migrating Code Between ColdFire V1 and V2, Rev. 0, Draft C**

**Table 4. Features of the ADC Modules for the ColdFire V1 and V2 (continued)**

| ADC Feature | V1 | V2 |
|---|---|---|
| Operation in stop mode | Y | N |
| Fully functional from 3.6 V to 1.8 V | Y | N |
| Simultaneous sampling of two channels | N | Y |
| Single or continuous conversion | Y | Y |
| Optional interrupts on conversion complete, zero crossing (sign change), or under/over low/high limit | N | Y |
| Automatic compare with interrupt for less-than, or greater-than or equal-to, programmable value. | Y | N |
| Ability to sequentially scan and store up to 8 measurements | N | Y |
| Signed or unsigned result | N | Y |
| Input clock selectable from up to four sources | Y | N |
| Single ended or differential inputs for all input pins with support for an arbitrary mix of input types | N | Y |

Figure 11 shows an example of the ADC usage for the ColdFire V1. The MCU_init disables the watchdog timer and the STOP mode, and enables the background debug pin and the $\overline{\text{RESET}}$ pin.

The GPIO_init function initializes the GPIO Port E module to be able to display the quantized analog data in the LED as a binary number.

The KBI_init configures the keyboard interrupts as an external interrupt to be triggered when one of the push buttons connected to the pins are pressed. At that moment, the ADC initialized the sampling process. The MCU waits until the ADC complete conversion flag has been set, acknowledging that a sample value is ready to be read out from the ADC data register. Then the ADC conversion complete flag is cleared and ADC is disabled, this occurs at the KBI_ISR function. At the ADC_ISR function, the data read is then passed to the GPIO Port E data register to be displayed in the eight LED array located on the evaluation board.

```
void main(void) {
  MCU_Init();                   /* Function that initializes the MCU */
  KBI_configuration();          /*Function that initializes the KBI module*/
  ADC_Init();                   /*Function that initializes the ADC module*/
  asm( move.w #2200, SR);       /*Interrupts enable*/
  while(1);
}
void MCU_Init(void) {
 SOPT1 = 0x23;   /* Watchdog disable. Stop mode enable.Background Pin enable.RESET pin enable */
  SCGC1 = 0x10; /* Bus Clock to the ADC module is enable */
  SCGC2 = 0x18; /* Bus Clock to the KBI and ACMP module is enable */
}
void GPIO_Init(void) {
PTEDD = 0xFF;          /* Configure PTE port as outputs */
PTED = 0x00;           /* Put 0's in PTE port */
}
void KBI_configuration(void) {
 KBI2SC = 0x06;        /* KBI interrupt request enabled. Detects edges only */
 KBI2PE = 0xF0;        /* PTD4, PTD5, PTD6 and PTD7 enabled as Keyboard interrupts */
 KBI2ES = 0x00;        /* Pins detects falling edge and low level (Pull-up) */
}
void ADC_Init (void) {
  ADCSC1 = 0x00;       /* Enable ADC interrupt */
/* Interrupt disable. One conversion and channel 0 active */
  ADCSC2 = 0x00;       /* Software trigger selected */
  ADCCFG = 0x30;       /* Input clock/2. Long Sample time configuration. 8-bit conversion */
  APCTL1 = 0x00;       /* ADC0 pin disable */
}
void interrupt VectorNumber_Vkeyboard KBI_ISR(void) {
  KBI2SC_KBACK = 1;    /* Clear KBI interrupt flag */
  APCTL1_ADPC0 = 1;    /* Select the channel for ADC input */
   ADCSC1_AIEN = 1;
}
void interrupt VectorNumber_Vadc ADC_ISR(void) {
 PTED = ADCRL;         /* Move the acquired ADC value to PTE port */
}
}
```

**Figure 11A. Code Snippet Showing How to Take a Sample from the V1 MC51QE ADC**

```
Void main( void ) {

  ADC_Init();

  EnableInterrupts;

  for(;;)     ;/* Infinite loop */

}

void ADC_Start(void )  {

  unsigned int i;

  MCF_ADC_CTRL1 |= MCF_ADC_CTRL1_START0;  /* request conversion*/

/* wait until result is ready for both converters */
  while((MCF_ADC_ADSTAT & MCF_ADC_CTRL1_EOSIE0)== 0)

  i = MCF_ADC_ADRSLT(0);  /* read converter A and B */

  MCF_ADC_ADSTAT = MCF_ADC_CTRL1_EOSIE0;  /* clear ADC flag */
}

/* Setup Module
 * It enables all channels. SO the order would be chan0-to chan7;  */

void ADC_Init()
{

  MCF_GPIO_PANPAR = ALL_ADC;  /* all pins in ADC mode */

  MCF_ADC_CTRL1 = MCF_ADC_CTRL1_SMODE(5);  /*triggered parallel*/


/* ADC clock 5 MHz and both ADC convertions are at the same time */
  MCF_ADC_CTRL2   = 0
        | MCF_ADC_CTRL2_SIMULT
        | MCF_ADC_CTRL2_DIV(2);


  MCF_ADC_ADLST1 = 0 | MCF_ADC_ADLST1_SAMPLE0(0)
                     | MCF_ADC_ADLST1_SAMPLE1(1)
                     | MCF_ADC_ADLST1_SAMPLE2(2)
                     | MCF_ADC_ADLST1_SAMPLE3(3);
  MCF_ADC_ADLST2 = 0 | MCF_ADC_ADLST2_SAMPLE4(4)
                     | MCF_ADC_ADLST2_SAMPLE5(5)
                     | MCF_ADC_ADLST2_SAMPLE6(6)
                     | MCF_ADC_ADLST2_SAMPLE7(7);
  MCF_ADC_ADSDIS  = ALL_ENABLED;
```

```
  MCF_ADC_POWER    = DEFAULT_DELAY;

  /* wait until module is powered-up */
  while(MCF_ADC_POWER & MCF_ADC_POWER_PSTS0);

  /* wait until module is powered-up */
  while(MCF_ADC_POWER & MCF_ADC_POWER_PSTS1);
}
__declspec(interrupt:0) void SW1Isr()  {
  /*clear the EPORT interrupt flag*/
  MCF_EPORT_EPFR |= MCF_EPORT_EPFR_EPF5;

  ADC_Start();   /* Acquired the ADC value */
}
```

**Figure 11B. Code Snippet Showing How to Take a Sample from the V1 MCF5221x ADC**

**Figure 11. Taking a Sample from the ADC**

## 2.5    Interrupt Controller

The ColdFire processor architecture defines a 3-bit interrupt priority mask field in the processor's status register (SR[I]). This field and the associated hardware support seven levels of interrupt requests with the processor providing automatic nesting capabilities. The levels are defined in descending numeric order with 7 > 6 ... > 1. Level 7 interrupts are treated as non-maskable, edge-sensitive requests. Levels 6–1 are maskable, level-sensitive requests. The SR[I] field defines the processor's current interrupt level. The processor continuously compares the encoded IRQ level from CF1_INTC against SR[I]. Interrupt requests are inhibited for all levels less than or equal to the current level, except the edge-sensitive Level 7 request, which cannot be masked.

Exception processing for ColdFire processors is streamlined for performance and includes all actions from the detection of the fault condition to the initiation of fetch for the first handler instruction. Exception processing is comprised of four major steps.

1. The processor makes an internal copy of the status register (SR) and enters supervisor mode.
2. The processor determines the exception vector number.
3. The processor saves the current context by creating an exception stack frame on the system stack.
4. The processor calculates the address of the first instruction of the exception handler.

The ColdFire V1 MCF51QE interrupt controller includes:

- Memory-mapped off-platform slave module
  - 64-byte space located at top end of memory: 0xFF_FFC0–0xFF_FFFF
  - Programming model accessed via the peripheral bus
  - Encoded interrupt level and vector sent directly to processor core
- Support of 30 peripheral I/O interrupt requests plus seven software (one per level) interrupt requests
- Fixed association between interrupt request source and level plus priority
  - 30 I/O requests assigned across seven available levels and nine priorities per level
  - Exactly matches HCS08 interrupt request priorities

**Migrating Code Between ColdFire V1 and V2, Rev. 0, Draft C**

— Up to two requests can be remapped to the highest maskable level + priority
- Unique vector number for each interrupt source
  — ColdFire vector number = 62 + HCS08 vector number
  — ColdFire vector number = 64 + Interrupt source number (0, 1, 2,..., 29)
  — Details on IRQ and vector assignments are device-specific
- Support for service routine interrupt acknowledge (software IACK) read cycles for improved system performance
- Combinatorial path provides wake-up signal from wait and sleep modes

The general features of the ColdFire V2 MCF5221x interrupt controller include:

- 57 interrupt sources
  — 50 fully-programmable interrupt sources
  — 7 fixed-level interrupt sources
- Each of the 57 sources has a unique interrupt control register (ICR*nx*) to define the software-assigned levels and priorities within the level
- Unique vector number for each interrupt source
- Ability to mask any individual interrupt source, plus global mask-all capability
- Supports hardware and software interrupt acknowledge cycles
- Wake-up signal from low-power stop modes

The 50 fully-programmable and seven fixed-level interrupt sources for the interrupt controller on the MCF5221x manage the complete set of interrupt sources from all of the modules on the device.

Table 5 shows the MC51QEx and the MCF5221x GPIO interrupt controller module common characteristics and features.

**Table 5. MC51QEx/MCF5221x GPIO Interrupt Controller Module Common Features**

| Attribute | ColdFire V1 MC51QE TPM | ColdFire V2 MCF5221x GPT |
|---|---|---|
| Exception vector table | 103, 4-byte entries, located at lower end of memory at reset, relocatable with the VBR | 256, 4-byte entries, located at lower end of memory at reset, relocatable with the VBR |
| More on vectors | 64 for CPU + 39 for device specific, reset at lowest address | 64 for CPU + 192 for device specific, reset at lowest address |
| Exception stack frame | 8-byte frame: F/V, SR, PC; General-purpose registers (An, Dn) must be saved/restored by the ISR | The first longword of the exception stack frame, pointed to by SP, contains the 16-bit format/vector word (F/V) and the 16-bit status register, and the second long word contains the 32-bit program counter address. |
| Interrupt levels | 7= f(SR[I]) with automatic hardware support for nesting | Same |
| Non-maskable IRQ support | Yes, with Level 7 interrupts | Same |

**Table 5. MC51QEx/MCF5221x GPIO Interrupt Controller Module Common Features (continued)**

| Attribute | ColdFire V1 MC51QE TPM | ColdFire V2 MCF5221x GPT |
|---|---|---|
| Core-enforced IRQ sensitivity | Level 7 is edge sensitive, else level sensitive | Same |
| INTC vectoring | Fixed priorities and vector assignments, plus any two IRQs can be remapped as the highest priority level 6 requests | Same |
| Software IACK | Yes | Yes |
| Exit instruction from ISR | RTE | RTE |

Figure 12 and Figure 13 show how the interrupt request level works. In the example we activate two programmable interrupt timers (PIT) to generate a periodical interrupt. The SW1 and SW2 buttons of the M52211EVB are also configured.

Each of these interrupt sources have a different interrupt level. At the beginning of the program, PIT0 has an interrupt level of 2 and a priority of 3 and the PIT1 has an interrupt level of 3 and a priority of 2. SW1 is connected to IRQ1 and SW2 to IRQ5, so they have interrupt levels of 1 and 5, respectively. The interrupt service routines (ISR) of the PITs have a delay cycle inside of them, and they indicate when the ISR begins and when it ends. The ISR of SW1 inverts the interrupt level and priority of the PITs, and the SW2 restores the initial values of those values.

The program shows how a higher priority interrupt request is asserted even when a ISR of a lower priority is running, and how the processor returns from the higher priority ISR to finish the lower priority ISR. When the program begins, the PIT0's ISR begins and if the PIT1 interrupt request is asserted, the PIT1's ISR will interrupt the PIT0's ISR. This happens until you press the SW1 button, which swaps the interrupt level of the PITs, then the PIT1's ISR is interrupted by PIT0's ISR.

The two external interrupts serve as configuration modifiers only. SW1 inverts the initial parameters of interrupt level and priority between PIT0 and PIT1. SW2 restores the initial parameters of the program. Both ISRs stop and restart the PITs to change the interrupt level. This is performed to ensure that the PITs do not generate an interrupt request twice, after the interrupt level is changed.

```
void main(void) {
  MCU_Init();            /* Function that initializes the MCU */
  KBI_configuration();   /* Function that initializes the KBI module */
  EnableInterrupts;      /* enable interrupts */
  Timer3();              /* Init Timer 3 */
  Timer2();              /* Init Timer 2 */
  INTC_PL6P7 = 16;       /* Remapped High priority Isr for Push button*/
  for(;;) {
  } /* loop forever */
}

void MCU_Init(void) {
 SOPT1 = (SOPT1_STOPE_MASK | SOPT1_RSTPE_MASK |
          SOPT1_BKGDPE_MASK); /* Watchdog disable. Stop  Mode Enable. Background Pin enable.
RESET pin enable */
  SCGC1 = (SCGC1_TPM3_MASK|SCGC1_TPM2_MASK);
/* Bus Clock to the TPM1 and TPM2 modules are enabled */
  SCGC2 = SCGC2_KBI_MASK; /* Bus Clock to the KBI module is enabled */
}
void KBI_configuration(void) {
  KBI2SC = (KBI1SC_KBIE_MASK|KBI1SC_KBACK_MASK;
  /* KBI interrupt request enabled. Detects edges only */
  KBI2PE = (KBI2PE_KBIPE4_MASK|KBI2PE_KBIPE5_MASK;
/* PTA2 and PTA3 pins enabled as Keyboard interrupts */
  KBI2ES = 0x00; /* Pins detects rising edge and high level (Pull-down) */
}
}
void Timer3 (void) {
  TPM3SC = (TPM3SC_CLKSA_MASK|TPM3SC_PS_MASK|
/* TPM source clock to preescaler input = Bus rate clock */
      TPM3SC_TOIE_MASK);/*TPM clock sourc divided by 128 */
  TPM3MOD = 2442;
  TPM3C1SC = 0x50;
  TPM3C1V = 0x0000;
}
void Timer2 (void) {
  TPM2SC = (TPM2SC_CLKSA_MASK|TPM2SC_PS_MASK|
/*TPM source clock to prescaler input = Bus rate clock */
  TPM2SC_TOIE_MASK);    /* TPM clock source divided by 128 */
  TPM2MOD = 1221;
  TPM2C1SC = 0x50;
  TPM2C1V = 0x0000;
```

```
void interrupt VectorNumber_Vtpm3ch TPM3_ISR(void) {
  UINT16 delay;
  UINT8 dummy;
  for (delay = 0; delay < 1221; delay++) {  }
  PTED_PTED2 ^= 1;
  dummy = TPM3C1V;
  TPM3C1SC_CH1F;     /* Clears timer flag */
 TPM3C1SC_CH1F = 0;
}

void interrupt VectorNumber_Vtpm2ch1 TPM2_ISR(void) {
  UINT16 delay3;
  UINT8 dummy;
  for (delay3 = 0; delay3 < 0xFF; delay3++) {  }
  PTED_PTED3 ^= 1;
  dummy = TPM2C1V;
   TPM2C1SC_CH1F;     /* Clears timer flag */
  TPM2C1SC_CH1F = 0;
}
```

**Figure 12. Code Snippet Showing How to Configure the Interrupt Controller, Configure the Timers, and Adjust Interrupt Levels and Priorities**

```
void main(void)  {

   MCU_Init(); /* Initializes the MCU*/
   EnableInterrupts();  /* Enable Interrupts*/
   IntConfig(PIT0_VECTOR,PIT0_INT_LVL,PIT0_INT_PRI   ,PIT0Isr);
   IntConfig(PIT1_VECTOR,PIT1_INT_LVL,PIT1_INT_PRI   ,PIT1Isr);
   PIT_Init(PIT0, PRESCALER, PIT0_MODULE);
   PIT_Init(PIT1, PRESCALER, PIT1_MODULE);
   while(1);    // Idle
}

void MCU_Init(void)  {
       mcf5xxx_startup();   /*Stand Alone initialization*/
}

void EnableInterrupts(void)  {
 asm { move.w SR,D0; andi.l #0xF8FF,D0; move.w D0,SR;  }
 mcf5xxx_set_handler ((64 + 7), SW2Isr); /*set handler to IRQ7*/
 mcf5xxx_set_handler ((64 + 5), SW1Isr); /*set handler to IRQ5*/
 }

void IntConfig(uint8 vector, uint8 level, uint8 priority, void (*handler) (void)) {

  /*configures the interrupt level and priority for the vector*/
  MCF_INTC0_ICR(vector)=(MCF_INTC_ICR_IP(priority)
                    |MCF_INTC_ICR_IL(level));

  /*masks the interrupt vector to allow the interrupt request*/
  if (vector < 32){

   /*if the interrupt vector is smaller than 32 writes to IMRL*/
   MCF_INTC0_IMRL &= ~(0x00000001<<vector);

}
else if (vector < 64){

  /*if the interrupt vector is between 32 and 64 writes to IMRH*/
  MCF_INTC0_IMRH &= ~(0x00000001<<vector-32);.

}

/*set the handler function of the interrupt*/
mcf5xxx_set_handler ((64 + vector), handler);
}
void PIT_Init(uint8 pit, uint8 prescaler, uint16 module )  {
  /*define preescaler*/
  MCF_PIT_PCSR(pit) =(MCF_PIT_PCSR_PRE(prescaler)
                    |MCF_PIT_PCSR_PIE
                    /*reset counter when write to PMR*/
                    |MCF_PIT_PCSR_OVW
                    /*reload module value when counter reaches 0*/
                    |MCF_PIT_PCSR_RLD
                    /*enable PIT*/
                    |MCF_PIT_PCSR_EN);
```

```
  /*write module to PMR*/
  MCF_PIT_PMR(pit) = MCF_PIT_PMR_PM(module);
}


*************************************************************/
/*PITInterrupts*/
/* PITx = PIT0 or PIT1
/*Programmable Interrupt Timer x Interrupt Service
Routine                                                   *
*    this function needs to be pointed on the vector.s file) */
*************************************************************/
 __declspec(interrupt:0) void PITxIsr()
{
   uint16 delay;      /*iterator for first cycle*/
   uint16 delay2;     /*iterator for second cycle*/

   /*clear PTCx*/
   MCF_GPIO_CLRTC &= ~(MCF_GPIO_CLRTC_CLRTCx);

   /*delay cycle*/
   for (delay = 0; delay < 0xFFFF; delay++) {
     for (delay2 = 0; delay2 < 0x0FF; delay2++) {
   }
/ }

   /*restarts the counter by writing to PMR*/
   MCF_PITx_PMR = MCF_PIT_PMR_PM(PITx_MODULE);

   /*clear PITx interrupt flag*/
   MCF_PITx_PCSR |= MCF_PIT_PCSR_PIF;

  /*set PTCx*/
  MCF_GPIO_SETTC |= MCF_GPIO_SETTC_SETTCx;
}
```

**Figure 13. Code Snippet Showing How to Configure the Interrupt Controller and SR to Receive Interrupt Request from Peripheral Modules, PITInit: Configures the Programmable Interrupt Timers**

```
void interrupt VectorNumber_Vkeyboard KBI_ISR(void){
/* Check PTA2 or PTA3 was pressed and store the value on button variable*/
button = (UINT8) (((PTDD)&0x30)>>4);
TPM3SC_TOIE = 0;    //Clear timers flags
TPM2SC_TOIE = 0;
KBI2SC_KBACK = 1;   /* Clear KBI flag */
switch (button) {
 case (1):
   PTED_PTED0 ^= 1     /* Turn PTC0 on */
   INTC_PL6P6 = 24;    /* set Level and priority for Timer 3*/
   break;
 case (2):
   PTED_PTED1 ^= 1;    /* Turn PTC1 on */
   INTC_PL6P6 = 7;     /* Return normal Level and priority*/
   break;
   }
}
```

**Figure 14. Keyboard Isr (PTD4 or PTD5): Push Button Interrupt Service Routine (Push Button Located on the EVBQE128 Starter Kit Configured by Software as External Interrupt)**

```
__declspec(interrupt:0) void SWxIsr()
{

  /*stops the PIT0, clears the PIF and resets the counter*/
  MCF_PIT0_PCSR &= ~(MCF_PIT_PCSR_EN);

  MCF_PIT0_PMR = MCF_PIT_PMR_PM(PIT0_MODULE);

  /*stops the PIT1, clears the PIF and resets the counter*/
  MCF_PIT1_PCSR &= ~(MCF_PIT_PCSR_EN);

  MCF_PIT1_PMR = MCF_PIT_PMR_PM(PIT1_MODULE);

  /*swaps the interrupt level and priority*/
      IntConfig(PIT0_VECTOR,PIT1_INT_LVL,PIT1_INT_PRI,PIT0Isr);

      IntConfig(PIT1_VECTOR,PIT0_INT_LVL,PIT0_INT_PRI,PIT1Isr);

  /*clear the EPORT interrupt flag*/
  MCF_EPORT_EPFR |= MCF_EPORT_EPFR_EPF1;

  /*restarts the PIT0-1*/
  MCF_PIT0_PCSR |= MCF_PIT_PCSR_EN;

  MCF_PIT1_PCSR |= MCF_PIT_PCSR_EN;
}
```

**Figure 15. SWxIsr (SW1 or SW2): SWx Push Button Interrupt Service Routine (Push Button Located on the 52211EVB Connected to IRQ5 and IRQ1 Pin, in the Edge Port Module)**

**Migrating Code Between ColdFire V1 and V2, Rev. 0, Draft C**

The interrupt controller is easy to configure. The only detail that needs to be considered is the configuration of the status register. Only two or three registers configure the interrupt requests of a specific vector. It must have a record of the interrupt level and priority from all the interrupts used because they can generate problems if they are entering ISRs without an order.

## 2.6    Benefits of Our Solution

With the addition of the low-end ColdFire V1 family to the 32-bit portfolio, the opportunity to migrate from high-performance 8-bit microcontrollers to low/mid-performance 32-bit processors is possible. The V1 product family enables the applications where performance, flexibility, and scalability are constraints, and allows the possibility to access Freescale's existing 32-bit portfolio.

It is important to consider a set of devices that provides a logical and natural migration path when designing systems that trend to a higher integration and complexity. Freescale offers a set of devices that targets this migration, enabling hardware designers to offer multiple configurations and performance levels of their products with one hardware and board design. Embedded software engineers found the same issues when developing software platforms for this type of scalable systems. Software re-investment and design cycle time is drastically reduced because devices with peripheral and tool compatibility makes the transition between low-end 32-bit and mid-end 32-bit devices fast and simple.

When switching between devices and architectures normally the software developer enters into a heavy learning curve about software development, peripherals availability, software and hardware tools, and architecture differences. By providing low-level drivers initialization and code examples; this document reduces the effort of moving from a ColdFire V1 to a ColdFire V2 device.

The code snippets provided  in this application note are intended help you initialize the common peripherals to provide quicker migration between the ColdFire V1 and the ColdFire V2 processors. By doing this, you should be able to keep the main application working independently from the device selected. This document provides tips on migrating between this two architectures. From this information and these examples, you can modify the code to suit your needs. From S08 to V1, from V1 to V2; step by step, user applications can evolve among Freescale's portfolio.

# 3    Conclusion

This document describes the similarities and differences between the low-end ColdFire V1 MC51QExx and  the V2 MCF5221 families. The V1 MC51QExx family is an ideal bridge for programmers who are looking for an entry-point to the 32-bit architecture because it preserves the easy-to-use experience of 8-bit architecture. The MCF5221x family of microcontrollers have all the benefits, performance, and set of peripherals used in mid-end 32-bit processors. This document shows how to migrate existing C code for the ColdFire V1 MC51QExx peripherals to the V2 MCF5221xx family processors.

If you have questions about your application, contact our support team.

# 4 Hardware and Software Used to Test the Code

Find the newest software updates and configuration files for the MCF52211 on the Freescale Semiconductor web page: www.freescale.com.

- This application note considers the MC51QExx family and MCF5221x family devices.
- For more information on peripheral modules, refer to MCF5221x and MC51QExxColdFire Integrated Microcontroller Reference Manual, rev 1 at www.freescale.com
- All code examples for the V2 ColdFire board was developed in the CodeWarrior™ tool for ColdFire V6.4. The V1 code examples were developed on CodeWarrior for Microcontrollers V6.0.
- All firmware was tested with the EVBQE128 Starter Kit and the M52211EVB, rev A.

Download the source files from www.freescale.com.

THIS PAGE IS INTENTIONALLY BLANK

**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: AN3464
Rev. 0, Draft C
07/2007