Freescale Semiconductor
Application Note

Document Number: AN3586
Rev. 0, 01/2008

Run uC/OS-II on MCF51QE128

Porting μC/OS-II Kernel to Freescale ColdFire V1 Core

by: Xu Wei
Systems and Applications Engineering

MCF51QE128 is a new Freescale MCU with a 32-bit ColdFire version 1 (V1) core, which is a reduced programming model of ColdFire core (V2–V5).The μC/OS-II is a real-time operating system (RTOS) for microcontrollers and microprocessors. It is widely used in the systems of many fields. The mC/OS-II runs on a large number of processor architectures such as ARM and MIPS. It can also run on Freescale ColdFire version 2 and version 3. AN1052 is available on Micrium website to help the users port uC/OS-II to ColdFire version 2 (V2).

This application note shows how to port uC/OS kernel on ColdFire version 1 and make it run on the silicon of MCF51QE128. The ColdFire V1 architecture and the uC/OS-II kernel are not detailed in this application note. However, it is beneficial to have some information on Freescale ColdFire V1 architecture and uC/OS-II kernel. All the code in this application note has been tested and debugged on MCF51QE128.

Contents

freescale™ semiconductor

1 Introduction

The MCF51QE128 is a member 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 package types. CPU clock rates on these devices can reach 50.33 MHz. Peripherals operate up to 25.165 MHz.

The MCF51QE128 is compatable with MC9S08QE128 whose core is 8-bit HC(S)08 in terms of peripheral setting and pin to pin. Using MCF51QE128, you could have the performance and flexibility of a 32-bit deivce with the simplicity of an 8-bit device.

For more information on MCF51QE128, visit www.freescale.com.

1.1 About Freescale ColdFire Core

The M68000 microprocessor family introduced in 1978 is one of the most popular architectures available. This architecture provides solutions to various markets including automotive, consumer, and communications. Because of its popularity, Freescale Semiconductor decided to continue the M68000 legacy by offering customers the same architecture but with greater performance and other functional enhancements.

The M68000 family is based on a complex instruction set computing (CISC) architecture that is easy to use, but there were needs for higher performance devices based on the same architecture. The solution is to offer customers a reduced instruction set computing (RISC) version based on the M68000. The ColdFire core was introduced to address issues needed by today's demanding complex applications.

The ColdFire family combines the benefits of CISCs and RISCs by keeping the same architecture that most customers are familiar and comfortable with, along with a much higher performance than CISC.

The ColdFire core is based on the concept of variable-length reduced instruction set (RISC) architecture. By utilizing variable-length instruction set, system designers can realize a significant system-level advantage over conventional RISC fixed-length instruction set. The ColdFire instruction set architecture supports variable-length RISC machines today. Although many ColdFire operations involve efficient 16-bit opcodes, the instruction length can be one, two, or three 16-bit words depending on the addressing modes used by the instruction.

For more information on ColdFire, visit www.freescale.com.

1.2 Freescale ColdFire V1 Architecture

The MCF51QE128 devices contain version 1 (V1) ColdFire core optimized for areas and low power. This CPU implements ColdFire instruction set architecture revision C (ISA_C) with a reduced programming model:

- No hardware support for MAC/EMAC and DIV instructions
- Upward compatible to all other ColdFire cores (V2–V5)

2 Introduction of μC/OS-II

The μC/OS-II is derived from μC/OS, the real-time kernel published in 1992. It is widely used in many applications, such as cameras, medical instruments, musical instruments, engine controls, network adapters, highway telephone call boxes, ATM machines, industrial robots, etc. Numerous colleges and universities use μC/OS to teach students about real-time systems.

The μC/OS-II is upward compatible with μC/OS (V1.11) and provides many improvements such as the addition of a fixed-sized memory manager, user definable callouts on task creation, task deletion, task switch and system tick, support of TCB extensions, stack checking, and much more.

2.1 Features

The μC/OS-II, the real time kernel is a portable, ROMable, preemptive real-time, multitasking kernel for microcontrollers and microprocessors. It can manage up to 64 tasks.

- Portable

  Most of μC/OS-II is written in highly portable ANSI C, with target microprocessor specific code written in assembly language. Assembly language is kept to a minimum to make μC/OS-II easy to be ported to other processors. The μC/OS-II can be ported to a large number of microprocessors as long as the microprocessor provides a stack pointer and the CPU registers can be pushed onto and popped from the stack. Also, the C compiler must provide in-line assembly or language extensions to enable and disable interrupts from C. The μC/OS-II can run on most 8-bit, 16-bit, 32-bit, or even 64-bit microprocessors or micro-controllers and DSPs. Please check for the availability of ports at www.Micrium.com.
- Scalable

  The μC/OS-II is designed scalably so that customers can only use the features needed in the applications. This means that a product can have a few of μC/OS-II's features while another product can have the full set of features. It reduces the amount of memory (RAM and ROM) needed by μC/OS-II on a product-per-product basis. Scalability is accomplished with the use of conditional compilation. Users only specify (through #define constants) which features are needed for the application.
- Preemptive

  The μC/OS-II is a fully-preemptive real-time kernel. It always runs the task with highest priority ready. Most commercial kernels are preemptive and μC/OS-II is comparable in performance with many of them.
- Multi-tasking

  The μC/OS-II can manage up to 64 tasks. However, the current version of the software reserves eight of these tasks for system use. This leaves the application with up to 56 tasks. The μC/OS-II cannot do round-robin scheduling because each task has a unique priority assigned. There are thus 64 priority levels.
- Deterministic

  Execution time of all μC/OS-II functions and services are deterministic. You can calculate how much time it takes μC/OS-II to execute a function or a service. Execution time of all μC/OS-II services do not depend on the number of tasks running in the application.

- Task stacks

  Each task requires its own stack. However, μC/OS-II allows each task to have a different stack size. This allows it to reduce the amount of RAM needed in the application. With μC/OS-II's stack-checking feature, the stack space each task actually requires can be calculated.
- Services

  The μC/OS-II provides a number of system services as follows:
  - Semaphores
  - Event flags
  - Mutual exclusion semaphores
  - Message mailboxes
  - Message queues
  - Task management
  - Fixed-sized memory block management
  - Time management
- Interrupt Management

  Interrupts can suspend the execution of a task. If a higher priority task is awakened as a result of the interrupt, the task with the highest priority runs as soon as all the nested interrupts complete. Interrupts can be nested up to 255 levels.
- Robust and reliable

  The μC/OS-II is based on μC/OS, which has been used in hundreds of commercial applications since 1992. μC/OS-II uses the same core and most of the same functions as widely used μC/OS yet offers more features.

2.2 Licensing

No licensing is required for μC/OS-II in educational use.

Contact Micriìm for proper license to use μC/OS-II in commercial products.

3 Tool Chain

This application uses Freescale CodeWarrior Development Studio for Microcontroller V6.0, which supports ColdFire V1 architecture to build, test, and debug the porting. Assembler, compiler, and linker are integrated into one toolchain. Please refer to the www.Freescale.com for more information on the Freescale MCU development tools.

The compiler and its configuration is also critical to the port. The following configuration of the compiler must be noticed.

- Parameter passing

  Parameter passing can be configured as standard, compact, and register. The port in this application can work with any of the three configurations. Register is recommended.
- Global Optimizations

There are a couple of optimization levels for TIMING or SIZE. The port in this application can work with any level of optimization for TIMING or SIZE.

4 Kernel Files Porting on ColdFire V1

4.1 OS_CPU.H

4.1.1 Compiler Independent Data Type

Different microprocessors have different word length. The port of μC/OS-II includes a series of type definitions in the file of OS_CPU.H that ensures portability.

- Floating-point is included but not used
- A stack entry is 32-bit wide for ColdFire V1 processor
- The status register is 16-bit wide for CFV1

```
typedef unsigned char  BOOLEAN;
typedef unsigned char  INT8U;                    /* Unsigned  8 bit quantity */
typedef signed   char  INT8S;                    /* Signed 8 bit quantity */
typedef unsigned short INT16U;                   /* Unsigned 16 bit quantity */
typedef signed   short INT16S;                   /* Signed 16 bit quantity */
typedef unsigned long  INT32U;                   /* Unsigned 32 bit quantity */
typedef signed   long  INT32S;                   /* Signed 32 bit quantity */
typedef float          FP32;                     /* Single precision floating point */
typedef double         FP64;                     /* Double precision floating point */

typedef unsigned long  OS_STK;                   /* Each stack entry is 32-bit wide */
typedef unsigned short OS_CPU_SR;                /* Define size of CPU status register */
```

4.1.2 Critical Section

The μC/OS-II, as all other real-time kernels, needs to disable interrupts to access critical sections of code and re-enable interrupts when done. Two macros, OS_ENTER_CRITICAL() and OS_EXIT_CRITICAL(), are used to disable and enable the interrupt respectively in the μC/OS-II. There are three ways to enter and exit the critical sections, but only one of them needs to be followed and it depends on the processor and the compiler. Here, the third way is followed.

```
#define OS_CRITICAL_METHOD  3

#if OS_CRITICAL_METHOD == 3
#define  OS_ENTER_CRITICAL()  __asm{\
      MOVE.L D0, -(A7)\
      MOVE.W SR, D0\
      MOVE.W D0, cpu_sr\
      ORI.L  #0x0700,D0\
      MOVE.W D0, SR\
      MOVE.L (A7)+, D0\
}                                                    /* Disable interrupts */
#define  OS_EXIT_CRITICAL()   __asm{\
      MOVE.L D0, -(A7)\
      MOVE.W cpu_sr, D0\
      MOVE.W D0, SR\
```

```
      MOVE.L (A7)+, D0\
}                                                          /* Enable  interrupts */
#endif
```

4.1.3 Task Level Context Switch

The Macro OS_TASK_SW() is invoked when task level context switch needs to be performed. Context switch is processor specific so assembly instructions need to be executed. In this case, an execption is generated by the software. There are 16 execptions that can be generated by software (#0 – #15). The corresponding vector numbers are from No. 32 – No. 47. For more detailed information of ColdFire V1 software exceptions, please refer to *ColdFire Core Reference Manual,* which can be downloaded at www.freescale.com. Here, the software exception (#14) is used for task level context switch.

```
#define uCOS                  #14            /* Interrupt vector # used for context switch */
#define OS_TASK_SW() __asm{\
      TRAP uCOS\
}
```

4.1.4 Stack Growth

On ColdFire V1 processor, the stack grows from the high memory to low memory, so OS_STK_GROWTH is set to 1 to indicate this stack growing direction.

```
#define  OS_STK_GROWTH 1                    /* Stack grows from HIGH to LOW memory on CF  */
```

4.1.5 ColdFire V1 Specific

The macro OS_INITIAL_SR specifies the initial value of status register when a task is created. It assumes that a task runs in the superviser mode and all interrupts are enabled.

```
#define  OS_INITIAL_SR 0x2000      /* Supervisor mode, interrupts enabled */
```

4.2 OS_CPU_C.C and OS_CPU_A.ASM

Two processor specific files, OS_CPU_C.C and OS_CPU_A.ASM, are required for porting. For the sake of easy maintenance, a file is put together by embeding assembly and has a new OS_CPU_C.C.

4.2.1 The Stack Frame Initialization Function

```
OS_STK  *OSTaskStkInit (void (*task)(void *pd), void *p_arg, OS_STK *ptos, INT16U opt)
{
      OS_STK  *pstk32;
      INT32U  init_A5 = 0x00A500A5L;

      opt = opt;              /* 'opt' is not used, for preventing compiler warning */

      __asm{
          MOVE.L A5, init_A5
      }

      pstk32 = (OS_STK *)((INT32U)ptos & 0xFFFFFFFC);

      *pstk32   = 0;                      /* SIMULATE CALL TO FUNCTION WITH ARGUMENT */
```

```
    *--pstk32 = (INT32U)p_arg;                                          /*    p_arg */
    *--pstk32 = (INT32U)task;                                 /* Task returns address */

                                                          /* SIMULATE INTERRUPT STACK FRAME */
    *--pstk32 = (INT32U)task;                                 /* Task returns address */
    *--pstk32 = (INT32U)(0x40000000 | OS_INITIAL_SR); /* format and status register */

                                                        /* SAVE ALL PROCESSOR REGISTERS */
    *--pstk32 = (INT32U)0x00A600A6L;                                    /* Register A6 */
    *--pstk32 = (INT32U)init_A5;                                        /* Register A5 */
    *--pstk32 = (INT32U)0x00A400A4L;                                    /* Register A4 */
    *--pstk32 = (INT32U)0x00A300A3L;                                    /* Register A3 */
    *--pstk32 = (INT32U)0x00A200A2L;                                    /* Register A2 */
    *--pstk32 = (INT32U)0x00A100A1L;                                    /* Register A1 */
    *--pstk32 = (INT32U)p_arg;                                          /* Register A0 */
    *--pstk32 = (INT32U)0x00D700D7L;                                    /* Register D7 */
    *--pstk32 = (INT32U)0x00D600D6L;                                    /* Register D6 */
    *--pstk32 = (INT32U)0x00D500D5L;                                    /* Register D5 */
    *--pstk32 = (INT32U)0x00D400D4L;                                    /* Register D4 */
    *--pstk32 = (INT32U)0x00D300D3L;                                    /* Register D3 */
    *--pstk32 = (INT32U)0x00D200D2L;                                    /* Register D2 */
    *--pstk32 = (INT32U)0x00D100D1L;                                    /* Register D1 */
    *--pstk32 = (INT32U)p_arg;                                          /* Register D0 */
    return ((OS_STK *)pstk32);                 /* Return pointer to new top-of-stack */
}
```

4.2.2 OSStartHighRdy()

OSStartHighRdy() is invoked by OSStart() to start running the highest priority task created before the OSStart().

```
asm void OSStartHighRdy(void)
{
    JSR       OSTaskSwHook                /* Invoke user defined context switch hook */

    MOVEQ.L   #1, D4                                          /* OSRunning = TRUE; */
    MOVE.B    D4,OSRunning                  /*   Indicates that we are multitasking */

    MOVE.L    OSTCBHighRdy,A1     /* Point to TCB of highest prio task ready to run*/
    MOVE.L    (A1),A7               /* Get the stack pointer of the task to resume */

    MOVEM.L   (A7),D0-D7/A0-A6                              /* Store all the regs */
    LEA       60(A7),A7                          /* Advance the stack pointer */

    RTE                                                       /* Return to task */
}
```

4.2.3 OSCtxSw()

A task level context switch occurs when a task is no longer able to run. Because a task cannot continue to run (the next most important task) in its ready status, it needs to resume execution. OSCtxSw() consists of saving CPU registers on the stack of the task and restoring the register from the stack of a new task. The vector of #14 software execption must be the entry of OSCtxSw().

```
asm void OSCtxSw(void)
{
```

```
      LEA       -60(A7),A7
      MOVEM.L   D0-D7/A0-A6,(A7)           /* Save the registers of the current task */

      JSR       OSTaskSwHook              /* Invoke user defined context switch hook */

      MOVE.L    OSTCBCur,A1          /* Save stack pointer in the suspended task TCB */
      MOVE.L    A7,(A1)

      MOVE.L    OSTCBHighRdy,A1                          /* OSTCBCur = OSTCBHighRdy */
      MOVE.L    A1,OSTCBCur
      MOVE.L    (A1),A7               /* Get the stack pointer of the task to resume */

      MOVE.B    OSPrioHighRdy,D0                       /* OSPrioCur = OSPrioHighRdy */
      MOVE.B    D0,OSPrioCur

      MOVEM.L   (A7),D0-D7/A0-A6                       /* Restore the CPU registers */
      LEA       60(A7),A7

      RTE                                                                 /* Run task */
}
```

4.2.4 OSIntCtxSw():

When an interrupt service rutine (ISR) completes, it needs to check whether there is a more important task than the interrupted task. If that is the case, OSIntCtxSw() is invoked. At that moment, the ISR had already saved the CPU register of interrupted task. Therefore, the CPU register of a new task only needs to be restored.

```
asm void OSIntCtxSw(void)
{
      JSR        OSTaskSwHook             /* Invoke user defined context switch hook */

      MOVE.B     OSPrioHighRdy,D0                       /* OSPrioCur = OSPrioHighRdy */
      MOVE.B     D0,OSPrioCur

      MOVE.L     OSTCBHighRdy,A1                          /* OSTCBCur = OSTCBHighRdy */
      MOVE.L     A1,OSTCBCur
      MOVE.L     (A1),A7                           /* SP = OSTCBHighRdy->OSTCBStkPtr */

      MOVEM.L    (A7),D0-D7/A0-A6   /* Restore ALL CPU registers from new task stack */
      LEA        60(A7),A7

      RTE                                                                 /* Run task */
}
```

4.3 Time Tick

The μC/OS-II, as the other real-time operating system kernel, requires a time tick, which is a periodic interrupt to keep track of time delay and timeout. The frequency of time tick depends on the time resolution of the application and must be betewen 10 Hz and 100 Hz. The higher frequency of the tick, the higher overhead of the application.

In this case, real time counter (RTC) module is used as time tick generator. OSTickInit() is used to initialize the RTC and starts the counter for time tick. TickISR() is the RTC interrupt service routine to implement

time tick function of μC/OS-II. It must be installed to the vector of RTC interrupt, which is in the vector table of MCF51QE128. TickISR() consists of two parts. One is the general routine of interrupt while the other, OSTickISR(), impliments the specific function of time tick.

```
void OSTickInit(void)
{
      SET_RTC_CLKSRC_LPO;

      SET_RTC_PRESCALER(11);

      SET_RTC_MOD_VALUE(9);

      ENABLE_RTC_INT;
}

asm void TickISR(void)
{

      MOVE.W     #0x2700,SR                                  /* Disable interrupts */

      LEA        -60(A7),A7                 /* Save processor registers onto stack */
      MOVEM.L    D0-D7/A0-A6,(A7)

      MOVEQ.L    #0,D0                                         /* OSIntNesting++ */
      MOVE.B     OSIntNesting,D0
      ADDQ.L     #1,D0
      MOVE.B     D0,OSIntNesting

      CMPI.L     #1, D0                                  /* if (OSIntNesting == 1) */
      BNE        _OS_My_ISR_1
      MOVE.L     OSTCBCur, A1                          /* OSTCBCur-<OSTCBStkPtr = SP */
      MOVE.L     A7,(A1)

      _OS_My_ISR_1:

      JSR        OSTickISR                                 /* OS_My_ISR_Handler() */

      JSR        OSIntExit                                        /* Exit the ISR */

      MOVEM.L    (A7),D0-D7/A0-A6          /* Restore processor registers from stack */
      LEA        60(A7),A7

      RTE                                             /* Return to task or nested ISR */
}

void OSTickISR(void)
{
      CLEAR_RTC_FLAG;

      OSTimeTick();

      OSIntExit();
}
```

4.4 Interrupt

Interrupt controller of ColdFire version 1 (CF1_INTC) is intended for low-cost microcontrollers. It maintains a vector table that can accommodate 256 exceptions. The first 64 are reserved for internal processor exceptions, and the remaining are for user-defined interrupt vectors. When an interrupt occurs, the CPU saves the SR and the PC onto the stack and executes the appropriate ISR from the vector table. Usually, the interrupt service routine must be written in assembly because the context of the interrupted task needs to be saved at the beginning of the ISR and to be restored at the end of ISR. Refer to the *MCF51QE128 Reference Manual* avaiable at www.freescale.com for more details on ColdFire V1 interrupt architecture.

The steps of each ISR under μC/OS-II is as follows:

1. Save the context of the interrupted task.
2. Call OSIntEnter() or increment OSIntnesting directly
3. If it is the first nested ISR, OS saves the task's stack onto the TCB.
4. Execute the C-level user code to service the interrupt
5. Call OSIntExit()
6. Restore the context
7. Return from the interrupt

TickISR() is a template of ISR under μC/OS-II. It only needs to change its name, invoke the function to service the specific interrupt, such as OSTickISR(), and locate it in the vector table according to its interrupt source.

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: AN3586
Rev. 0
01/2008

freescale™
semiconductor