ECE263

Lab 3

Spring 13

Real Time Interrupt
RTI

Revised: 3 Feb 2013 DAL

(intentionally left blank)

Object: This lab will demonstrate three concepts:

- breaking a program into a series of smaller tasks that are called by a scheduler.
- using the real time interrupt (RTI) to test interrupt processing
- doing bit banging to I/O ports so that a logic analyzer can be used to track the flow of the program as the tasks are called.

References:

MC9S12DP256.pdf

Background:

Scheduler:

When writing programs for complex systems, the best approach is to break the problem down into smaller pieces and write separate programs for each of the various operations. These routines would then be called and executed as they are required. Each routine should perform a single task and have explicit entry and exit conditions. For example, a routine might perform a floating point multiply, and when called, the multiplier and multiplicand would be passed to the routine and upon completion the result would be returned. Another example would be a routine that reads data from a sensor and stores the data into a specified memory buffer.

This approach for writing programs is especially useful for embedded systems which normally have real time computing requirements and typically interface to many I/O devices. Since I/O devices are usually very slow compared to the processing time of the CPU, they often have the ability to interrupt the program when they need service. These interrupts would be random events that must be handled as they occur without disturbing the flow of the program. It is also quite common to have some tasks that need to be initiated at a periodic rate or for tasks that must wait until one or more other tasks have completed before they can be started.

One simple way to handle this type of system is to have a main routine that acts as a scheduler or task manager and have it run in a loop constantly looking for events that need to be serviced and then call the appropriate routines (see Figure 1). A set of status flags or semaphores are used to indicate that service is required. When one task needs the services of another task, it would set the appropriate task flag. At some point, the scheduler would detect that the flag is set and then call the required routine. As part of the normal processing, the called task would clear the task flag so that the scheduler would not call it again until the task flag is set again.

Real Time Interrupt

This lab will be using the real time interrupt (RTI) peripheral of the MC9S12 microcontroller to generate a periodic interrupt. The program will respond to the interrupt by entering a program referred to as an interrupt service routine and doing some operations typical of embedded systems. The user must perform some I/O operations to the hardware before the RTI can be used. To enable the RTI and set the period of the counter, the program must write a value to RTICTL located at $003B (see Table 47 of the MC9S12DP256 manual). To enable the interrupt, a 1 must be written to RTIE (bit 7 of CRGINT located at $0038). After it is setup, the RTI will continually time out at a periodic rate and the RTIF (bit 7 of CRGFLG located at $0037) will be set to 1 and the interrupt will be generated.

Debugging via bit banging

When debugging programs, designers usually rely on single stepping and break points to follow the operation of the program, however, for systems that operate in real time, that method is not adequate. In order to track the flow of a program in real time, a programmer can do bit banging to

I/O port bits at strategic points in the program and then monitor those points using a logic analyzer. The captured timing traces will reveal the sequences and timing of the flow of the program at full speed.

For this lab we will use Port A and Port B for diagnostic bit banging. Bits of Port A will be toggled from 0 to 1 or 1 to 0 each time the program reaches a specific point as described in the program requirements section. Bits of Port B will be set when specific routines are entered and cleared when the routines are exited. Use the definitions below for the bit banging.

```
; Port A & B diagnostic bits
rti_diag:   equ   %00000001
t1_diag:    equ   %00000010
t2_diag:    equ   %00000100
t3_diag:    equ   %00001000
m_diag:     equ   %00010000
```

The commands you need to use to toggle, set, or clear the bits are as follows (note: Port A is to be used for toggle and port B is to be used for set/clear):

```
LDAB  PORTA
EORB  #t1_diag          ; toggle bit
STAB  PORTA

LDAB  PORTB
ORAB  #t1_diag          ; set bit
STAB  PORTB

LDAB  PORTB
ANDB  #(t1_diag ^ $FF)  ; clear bit
STAB  PORTB
```

Pre-lab: This lab will use the program(s) described at the end of this write-up. For the pre-lab, generate a flow chart and/or pseudo code that shows the design of the program and create the assembly source code for the program.

Procedure: Working with your lab partner, complete the following steps:

1. Create a new project and enter your program as *main.asm*.
2. Set up the Logic Analyzer to display signals of Port A and Port B that will help you to debug the flow of the program and verify the timing. Make sure you label the traces with names like TASK1 and TASK2 to make the plots more understandable.
3. For the initial debug of the main routine, comment out the enable interrupt instruction (CLI) in the source code.
4. Generate the executable object file using the *Make* facility and then download the program on the Dragon 12 Plus.
5. Debug the main/scheduler program and verify that the variables and ports got initialized correctly and verify the flow of the program in the scheduler loop. (Note: with no interrupts enabled the program should stay in the loop and not call any tasks.)
6. Correct any program errors and repeat steps 4 – 5.
7. Use the logic analyzer to determine how long the scheduler loop takes.

8. Now restore the CLI instruction in the source code and reload the program.
9. Now verify that the program does the following:
    a. the RTI_ISR is entered and that it does the correct things
    b. each task is entered and does the correct things
10. Using the logic analyzer, verify
    a. the timing of the RTI_ISR
    b. the timing of the tasks
    c. the sequences of the task calls
11. Correct any program errors and repeat steps 9 – 10.
12. Print out logic analyzer plots to show the timing and flow of your program
13. Demonstrate the working program to the instructor or TA and get the Verification sheet signed.

Program Requirements: The program will consist of five separate routines: a main routine/scheduler, three tasks, and an interrupt service routine. The operation of each routine is described below.

Main/Scheduler

The main routine will follow the flow chart laid out in Figure 1. It will do all of the necessary initializations such as the setup of DDRA and DDRB, initialize the I/O ports, clear any/all variables. It will also initialize the real time interrupt (RTI) peripheral of the 9S12 so that it will generate an interrupt every 1.024×10^{-3} seconds as described above. After initialization is complete, the routine will drop into the test/dispatch loop which will be run forever. The main loop in the routine will examine the task flags which are actually individual bits of a variable located in RAM space, for example:

```
T_FLG:    DS.B  1          ; task flag storage
TSK_1:    equ   %00000001  ; Task 1 needs service
TSK_2:    equ   %00000010  ; Task 2 needs service
TSK_3     equ   %00000100  ; Task 3 needs service
```

The scheduler will load the variable T_FLG and test each bit of the byte one at a time. If the bit is a one, it will call the associated task otherwise it will test the next bit. The scheduler does not change the contents of the flag byte in any way. At the end of the loop, the *m_diag* bit of Port A should be toggled as explained above.

RTI_Interrupt Service Routine

The RTI interrupt service routine (RTI_ISR) will be used to maintain two counters that will be incremented each time the routine is entered. These counters will be variables located in RAM space and must be loaded each time they are used and written back to RAM afterward. One counter (CNTA) will count from 0 to 3 and then wrap back to 0. The second counter (CNTB) will count from 0 to 9 and then wrap back to 0. Each time CNTA is reset to 0, task flag 1 (TFLG_1) will be set to a 1 and when CNTB is reset to 0, TFLG_2 will be set to 1. Upon entry to the routine, *rti_diag* bit of Port A should be toggled and the *rti_diag* bit of Port B should be set to 1. You also have to clear the interrupt by writing a 1 back to RTIF (bit 7 of CRGFLG located at $0037). Just prior to executing the return from interrupt command, the *rti_diag* bit of Port B should be cleared to 0.

Task 1

Task 1 will be a very simple routine that is basically just a “place holder” for a function we will use in subsequent labs. This task was called because the TSK_1 flag was set therefore this routine needs to clear the bit. In order to simulate some processing time, you should load a register with a number and then enter a loop decrementing it to zero at which time you can exit the routine. For debug purposes, upon entry to the routine, the *t1_diag* bit of Port A should be toggled and the *t1_diag* bit of

Port B should be set to 1. Just prior to executing the return from subroutine command, the *t1_diag* bit of Port B should be cleared to 0.

.

Task 2

Task 2 will also be a very simple routine that will be a "place holder" for another function we will use in subsequent labs. This task was called because the TSK_2 flag was set therefore this routine needs to clear the bit. In this routine you will maintain another counter that is located in RAM space and counts from 100 down to 0. Each time you enter the routine you need to decrement the counter by one and save the data back to RAM. Each time 0 is reached you will set TSK_3 to a one. For debug purposes, upon entry to the routine, the *t2_diag* bit of Port A should be toggled and the *t2_diag* bit of Port B should be set to 1. Just prior to executing the return from subroutine command, the *t2_diag* bit of Port B should be cleared to 0.

Task 3

Task 3 will represent a routine that is initiated by another task. This task was called because the TSK_3 flag was set therefore this routine needs to clear the bit. In order to simulate some processing time, you should load a register with a number and then enter a loop decrementing it to zero at which time you can exit the routine. For debug purposes, upon entry to the routine, the *t3_diag* bit of Port A should be toggled and the *t3_diag* bit of Port B should be set to 1. Just prior to executing the return from subroutine command, the *t3_diag* bit of Port B should be cleared to 0.

Interrupt Vectors

The interrupt vector table needs to be modified in order to use the RTI interrupt. Insert the following code at the end of your program.

```
 ; catch-all isr

 DFLT_ISR:   NOP
             RTI

 ;*************************************************************
 ;   Interrupt Vectors

             ORG   $FFF0
             DC.W  RTI_ISR        ;rti
             DC.W  DFLT_ISR       ;irq pin
             DC.W  DFLT_ISR       ;xirq pin
             DC.W  DFLT_ISR       ;swi
             DC.W  DFLT_ISR       ;non_inst
             DC.W  DFLT_ISR       ;cop fail
             DC.W  DFLT_ISR       ;clk fail
             DC.W  Entry          ;Reset Vector
```

Memory Map:

Reg/Mem	Memory Address
PTA	$0000
PTB	$0001
DDRA	$0002
DDRB	$0003
CRG_FLG	$0037
CRG_INT	$0038
RTI_CTL	$003B
RAM	$1000
RAM_END	$3FFF
FLASH	$C000

Lab Report : In addition to the normal information as described in the Lab Guideline, include the following:

1. Answers to any/all questions
2. Logic analyzer plots showing the flow and timing of the program

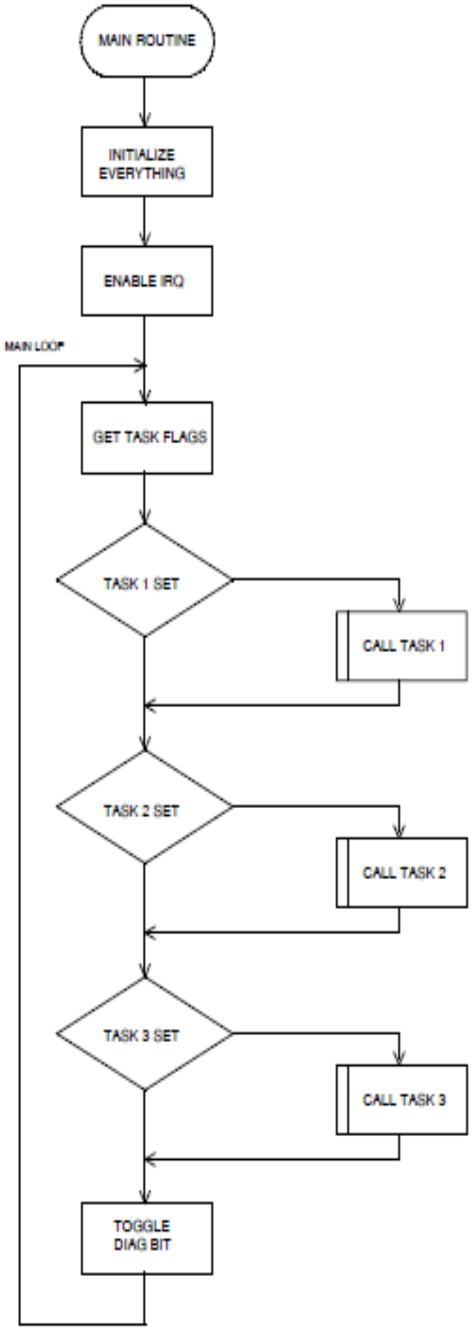

Figure 1: Flow Chart – Main Routine