6

Maskable Interrupts

6.1 Objectives:

In lab 5, we used the polling technique for parallel I/O. Using interrupts can improve performance of the computer and make the software easier to organize. In this lab, you will learn,

- What is an interrupt?
- The difference between maskable and non-maskable interrupts.
- Response of TM4C to a maskable interrupt.
- How to generate delays using the System Timer system.
- How to write interrupt service routines (ISR) for the TM4C.

6.2 Related material to read:

- Interrupts: Chapter 12 in the text, pages 249 – 274.
- TM4C Data Sheet, Sections 3.3 and 3.4.
- Valvano Volume 1, Section 4.4 and Chapter 9.
- Startup.s: for a complete Interrupt Vector Map

6.3 Maskable and non-maskable interrupts:

Lab 3 briefly described the register model for the TM4C. This register model contained the Program Status Register (PSR). Eight of the bits in the PSR (bits 7:0) indicate which interrupt, if any, is currently being handled.
Another special register associated with interrupts is the PRIMASK register. The PRIMASK register contains only one bit (also named PRIMASK) that enables / disables most interrupts. The user cannot turn off a non-maskable interrupt.

A maskable interrupt may be turned on or off by the user under program control. Its corresponding interrupt masking bit (I) is set to logic 1 during system reset, which turns off the maskable interrupt system. The user can turn on this system using the Change Processor State Interrupt Enable CPSIE I instruction and turn off this system using the Change Processor State Interrupt Disable CPSID I instruction. The TM4C is equipped with several maskable interrupts, e.g., SysTick, Timer, Pulse accumulator, SPI, and A/D system. In this lab, we will consider only one maskable interrupt, i.e., the System Timer (SysTick). Other interrupts are discussed in future labs.

The System Timer generates an interrupt at a user specified interval. This interrupt is very useful in reminding the TM4C to perform a regular, repetitive task such as measuring the temperature in the controlled room at a regular interval.

A complete interrupt vector map for TM4C is listed in the file Startup.s available on our lab webpage. This map tells you where the interrupt vector the appropriate interrupt service routine is located for all possible TM4C interrupts.

6.4 Interrupt service routine:

An interrupt service routine or ISR is similar to a subroutine. The main program runs in an infinite loop, sometimes doing other useful work. When an I/O device is ready to service, it sets a flag, which causes the main program to turn the control over to the ISR. The ISR services the I/O device by clearing the flag, doing any data transfer, and any other work required. An ISR is similar to a subroutine but it responds to a hardware signal whereas a subroutine is called because of an instruction in the program.

6.5 Programming the System Timer (SysTick):

In the previous labs we used software-controlled delays. If accurate timing is required, the system timer in the processor can be used. Refer to Section 12.4 in the text for the detailed description of the System Timer.

There are three important control registers for the SysTick in the TM4C: STCTRL, STRELOAD, STCURRENT.

The Status and Control register (STCTRL) is used to enable the SysTick system (interrupt vector address 0x0000.003C) (see Figure 6.4). Bit 0 of this register is the System Timer Enable (ENABLE) bit. This bit when set to 1 enables the SysTick system, and when set to 0, disables the SysTick system. Bit 1 is the Interrupt Enable bit (INTEN) which enables interrupts for the SysTick system. When this bit is set to 1 interrupts are enabled and an ISR will be called when the timer reaches 0. Bit 2 is the Clock Source bit (CLK_SRC). The internal oscillator of the TM4C is not the only clock available to use. With additional setup, other more accurate oscillators can be used. However, for now we will simply use the Precision Internal Oscillator (PIOSC) divided by 4. The PIOSC runs at 16 MHz on the TM4C, so PIOSC/4 is 4 MHz. Hence the clock period is 250ns.
The SysTick Reload Value register (STRELOAD) selects the time-out period of the SysTick system (see Figure 6.5). RELOAD[23:0] is where the 24-bit starting value for the counter will be stored. The SysTick system will count down from this value to 0, subtracting 1 every clock cycle.

The SysTick Current Value register is a 24-bit register containing the current value of the count down from the RELOAD value.

The SYSPRI3 register configures the priority level, 0 to 7 of the SysTick exception handler by placing this level into the TICK field (bits 31:29). The lower the value of this field, the higher its priority. A priority level of 1 or greater is required to enable SysTick interrupts. This register is byte addressable.
Figure 6.9 shows a program code that can be used to initialize the SysTick system. Lines 20–22 first disable the SysTick system during setup by clearing the ENABLE bit in the STCTRL register. Lines 23–25 load the number of clock cycles to count down from into the STRELOAD register. Lines 26-28 clear the CURRENT register by writing any value to it. Lines 29-31 set the priority of the SysTick to 2. Lines 32-34 finally enable the SysTick system and allow interrupts. Lines 43-49 is the SysTick Interrupt Service Routine (ISR). Note the EXPORT directive on line 43 to make it available to Startup.s. The CPSIE I instruction on line 37 enables the maskable interrupts. Thus, about every second, SysTick system will issue an interrupt and will print “Hello from SysTick” on the screen.

1. NVIC_ST_CTRL EQU 0xE000E010
2. NVIC_ST_RELOAD EQU 0xE000E014
3. NVIC_ST_CURRENT EQU 0xE000E018
4. SHP_SYSPRI3 EQU 0xE000ED20
5. RELOAD_VALUE EQU 0x0003C000
6.
7.
8.
9. ;*********************************************************
10. ; Data area Readonly Memory
11. ;*********************************************************
12. hello DCB "Hello from SysTick",13,4
13.
14. ;*********************************************************
15. ; Program area
16. ;*********************************************************
17. AREA |.text|, CODE, READONLY, ALIGN=2
18. THUMB
19.
20. __main LDR R1, =NVIC_ST_CTRL
21. MOV R0, #0
22. STR R0, [R1]
23. LDR R1, =NVIC_ST_RELOAD
24. LDR R0, =RELOAD_VALUE
25. STR R0, [R1]
26. LDR R1, =NVIC_ST_CURRENT
27. MOV R0, #0
28. STR R0, [R1]
29. LDR R1, =SHP_SYSPRI3
30. MOV R0, #0x40000000
31. STR R0, [R1]
32. LDR R1, =NVIC_ST_CTRL
33. MOV R0, #0x03
34. STR R0, [R1]
35. ; SysTick is now set up. Main program follows.
36. ;
37. CPSIE I
38. wait WFI
39. B wait
40. ;*********************************************************
41. ; SysTick ISR
42. ;*********************************************************
43. EXPORT SysTick_Handler
44. SysTick_Handler
45. PUSH (LR)
Figure 6.9: Program that initializes SysTick system and prints “Hello from SysTick” after each timeout period of around 1s.

6.6 Procedure:

1. Before lab, create a flow chart for the digital alarm clock below. This flow chart should include all the subroutines by name from the following instructions. Also, write the code for the setme routine (see procedure 2b) and test it with a main program designed for that purpose. Show this flow chart to your TA at the beginning of your Lab 6 period, and be prepared to hand it in if requested.

2. The objective of your next task is to create a simple digital alarm clock consisting of a TM4C interfaced to an LED (that serves as a “silent” alarm indicator). After prompting the user to set the current time and the alarm, the clock will run until the board is reset. Once the alarm goes off (activating the LED), it will stay on until any key is pressed on the keyboard. The alarm setting and current time should be displayed in the terminal emulation window as follows (note that the time should be displayed once each second). You can use the following steps for creating this digital alarm clock. Show the working program to the TA.

a) Write the main program. After initializing the LED alarm indicator port, main should prompt the user for the CLOCK setting followed by the ALARM setting (making calls to the setme routine to initialize the clock and alarm variables). After initializing the clock and alarm variables, main should start the clock (by calling the systick_ini routine) and then remain in a loop waiting for any key to be pressed (which, when pressed, should turn off the alarm indicator LED).

b) Write the clock/alarm set routine setme. At entry, a general register should point to the start of the time (or alarm) array. The setme routine should prompt the user for entry of hours, followed by minutes, seconds, and either “A” or “P” (for AM or PM, respectively). While it would be “nice” to include, no error checking is required on the entry of hours, minutes, or seconds (i.e., a “malicious” user could enter an invalid time). If neither “A” nor “P” is entered for AM/PM; however, the user should be re-prompted for a valid entry.

c) Write the clock/alarm display routine cdisp. At entry, a general register should point to the start of the time (or alarm) array. The cdisp routine should display the contents of the specified array as “hh:mm:ss X”, where hh is hours, mm is minutes, ss is seconds, and X is “A” or “P”.

d) Write the alarm monitor routine almon. This routine should compare the time and alarm arrays and, if an exact match occurs, the LED alarm indicator should be activated.

e) Write the LED alarm indicator port initialization routine led_ini. The LED will be interfaced to one of the pins of Port B. This routine should set up the appropriate GPIO configuration registers for Port B and initialize the LED to the “off” state.

---

^1 From the lab manual for EE362 course in University of Purdue.
f) Write the clock update routine `clock`. This routine should increment the current time by one second, as well as handle any “roll-overs” that occur to minutes, hours, and AM/PM.

g) Write the SysTick interrupt service routine `systick_isr`. This routine should call the `clock` routine, call the `almon` routine, and call the `cdisp` routine.

h) Write the SysTick initialization routine `systick_init`. This routine should enable the SysTick subsystem for a 1s interrupt rate and clear the “I” mask bit in the PRIMASK.

Optional procedure for more practice:

3. Connect the 6 DIP switches to an input port and connect 6 LED’s to an output port, as you did in the previous lab on parallel I/O interfaces. Draw a flow chart and write a program that uses the SysTick system to reflect the status of the DIP switches to the LED’s connected to the output port after every 15 seconds. Show the working program to the TA.

6.7 Questions:

1. Consider the program written in Figure 6.10. This program makes use of the SysTick system to generate a 10 second delay. The program tries to show the ‘START’ message at the start of the program. Then it waits for 10 seconds before displaying the ‘STOP’ message on the screen. The program will not work, as there are errors in the program. Locate the errors in the program and modify the program so that it will work properly.

```assembly
1. EXTERN SysTick_Init
2. EXTERN OutStr
3.
4. RELOAD_VALUE EQU 0x000008FF
5. NVIC_ST_CTRL EQU 0xE000E010
6. NVIC_ST_RELOAD EQU 0xE000E014
7. NVIC_ST_CURRENT EQU 0xE000E018
8. SHP_SYSPRI3 EQU 0xE000ED20
9.
10. ;***********************************************
11. ; ROM area
12. ;***********************************************
13. AREA rdata, DATA, READONLY, ALIGN=3
14. starttxt DCB "Start",13,4
15. stoptxt DCB "Stop",13,4
16. delsec DCD 10
17.
18. ;***********************************************
19. ; Program area
20. ;***********************************************
21. AREA |.text|, CODE, READONLY, ALIGN=3
22. THUMB
23.
24. EXPORT __main
25. __main
26. LDR R1,=NVIC_ST_CTRL
27. MOV R0,#0
28. STR R0,[R1]
29. LDR R1,=NVIC_ST_RELOAD
30. LDR R0,=RELOAD_VALUE ; load reload val into R0
31. STR R0,[R1]
```
2. What are the differences between the initialization of the maskable and non-maskable interrupts?

3. List the similarities and differences of interrupt service routines over the subroutines.

4. Which delay routine is more accurate, the routine that uses the software program or the one that uses the SysTick system? Explain why.

**Figure 6.10:** Program that tries to implement a 10 second delay using SysTick system.
6.8 Lab report:

For the lab write-up, include

1. Flowcharts and programs that you wrote before the lab.
2. A copy of your working .s files.
3. A brief discussion of the objectives of the lab and the procedures performed in the lab.
4. Answers to any questions in the discussion, procedure, or question sections of the lab.