Spring 2022

Haroutun Haroutunian
Andres Lopez

California State University Northridge
Department of Electrical and Computer Engineering

Experiment 10

Interrupts

Professor Flynn

May 6, 2022

Introduction

In this lab experiment, we will learn about interrupts, how to write interrupt service routines, and how to configure external interrupt inputs. There are different types of interrupt service routine, which include Software Interrupt (SWI), Interrupt Request (IRQ), and Fast Interrupt Request (FIQ). In this lab, we will be writing Assembly programs to make Software Interrupt calls to provide different services to light up LEDs as interrupts are an important mechanism for large applications to handle multitasking. We will also learn how to configure one of the pins as an external interrupt input pin to generate an IRQ interrupt signal to the processor.

Procedure

Before students begin causing interrupts, the startup code must be modified to accept external interrupts and run through the SWI handler. The startup code is added-on from lab 9's code with the sensitivities and stacks, as shown in figures 10.1. The files from lab 9, LCD screen, are implemented in this experiment as well. As shown in figure 10.2, the main code to run the LCD screen is duplicated.

```
;SETTING OF THE EXTERNAL INTERRUPTS
    LDR R0, =PINSEL1
    LDR R1, [R0]
    BIC R1, #0X3
    BIC R1, #0X300
    ORR R1, R1, #EINT0 ;SETS 01 VALUE AT BITS 1:0 FOR EINT0
    ORR R1,R1,#EINT3 ;SETS 11 VALUE AT BITS 9:8 FOR EINT3
    STR R1,[R0]
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;EXTMODE SETTING EDGE-SENSITIVITY
    LDR R2, =EXTMODE
    LDR R1, [R2]
    ORR R1,R1,#UPEDGE
    STR R1,[R2]
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;EXTMODE SETTING HIGH ACTIVE
    LDR R2, =EXTPOLAR
    LDR R1, [R2]
    ORR R1,R1,#UPEDGE
    STR R1,[R2]
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;EXTINT SETTING FLAGS AT 0X9 SILENCING INTERRUPTS
    MOV R1, #0X9
    LDR R2, =EXTINT
    STR R1,[R2]
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;ENTER SUPERVISOR MODE
    MOV R14, #SUP_MODE
    ORR R14,R14,#(I_BIT+F_BIT)
    MSR CPSR_c, R14
;INITIALIZE THE STACK, FD
    LDR SP, =SUP_TOP
;LOAD THE START OF ADDRESS OF SUP CODE INTO PC
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;ENTER IRQ MODE
    MOV R14, #IRQ_MODE
    ORR R14,R14,#(I_BIT+F_BIT)
    MSR CPSR_c, R14
;INITIALIZE THE STACK, FD
    LDR SP, =IRQ_TOP
;LOAD THE START OF ADDRESS OF SUP CODE INTO PC
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;ENTER FIQ MODE
    MOV R14, #FIQ_MODE
    ORR R14,R14,#(I_BIT+F_BIT)
    MSR CPSR_c, R14
;INITIALIZE THE STACK, FD
    LDR SP, =FIQ_TOP
;LOAD THE START OF ADDRESS OF SUP CODE INTO PC
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; Enter User Mode with interrupts enabled
    MOV     r14, #USER_MODE
    BIC     r14,r14,#(I_BIT+F_BIT)
    MSR     CPSR_c, r14
```

Figure 10.1: Mystartup clock settings and stacks

```
        GLOBAL user_code
        IMPORT  LCD_pinss
        IMPORT  LCD_inits
        IMPORT  LCD_clear
        IMPORT  LCD_cmds
        IMPORT  LCD_strings
        IMPORT BRANCHBACK
            AREA mycode, CODE, readonly

IOOPIN EQU 0XE0028000

user_code
        BL LCD_pinss
        BL LCD_inits
        LDR R0,=IOOPIN
        MOV R1, #0X40000000
        LDR R2,[R0]
        ORR R2,R2,R1
        STR R2,[R0]
        MOV R6, #0X80000000
        BL LCD_cmds
        BL LCD_clear
        MOV R6, #0XF0;CURSOR SET AT LOCATION 00 LINE 1
        LDR R8, =STRING1
        BL LCD_strings
        ;BL LCD_clear
        MOV R6, #0XC0;CURSOR SET AT LOCATION 40 LINE 2
        LDR R8, =STRING2
        BL LCD_strings
        B  BRANCHBACK

STRING1 DCB "BEST",0
STRING2 DCB "PROFESSOR",0
        ALIGN
        END
```

Figure 10.2: LCD screen program from lab 9

Within the startup code vectors, the SWI_addr and SWI_HANDLER and added, which loads the address of the SWI_HANDLER into the Program Counter, as seen in figure 10.3 line 34. Once the Program Counter is "pointing" to the SWI_HANDLER address, the handler file is ran, which figures out which SWI # is called and does the tasks accordingly.

```
30 VECTORS
31
32     LDR         PC,Reset_Addr
33     LDR         PC,Undef_Addr
34     LDR         PC,SWI_Addr
35     LDR         PC,PAbt_Addr
36     LDR         PC,DAbt_Addr
37     NOP
38     LDR         PC,IRQ_Addr
39     LDR         PC,FIQ_Addr
40
41 Reset_Addr      DCD         Reset_Handler
42 Undef_Addr      DCD         UndefHandler
43 SWI_Addr        DCD         SWI_HANDLER
44 PAbt_Addr       DCD         PAbtHandler
45 DAbt_Addr       DCD         DAbtHandler
46                 DCD         0
```

Figure 10.3: mystartup SWI_HANDLER implementation

The code for running the LCD and initializing it are also transferred from previous experiments. In summary, experiment 9 required students to initialize the LCD screen before writing strings to it, as well as saving registers and flags throughout every subroutine. Any LCD code is run through as a software interrupt caused by the programmers and nothing has been changed from the previous experiment.

Firstly, within the SWI_HANDLER file, EQU's are set similarly to the experiment with the LEDs to light up the LEDs within SWI #1 and SWI #3. On lines 14 and 52 in figure 10.4 are the "box" layout for any interrupt program, which saves the registers and LR. Line 15 requires students to move the Link Register 1 instruction back to read which SWI # was called, then its' address is stored into Register 0, while Register 1 contains the actual SWI #. Line 17 clears the upper 8 bits and line 18 tests if Register 1 contains the #1. If SWI #1 was called, the program would branch link to LED_LIGHT (lines 33-42), which lights all 8 LED's. The code is then ran through a delay and repeats the process from line 18 to test whether SWI #2 was called. SWI #2 would run through the LCD subroutine to display "BEST // PROFESSOR" on separate lines. Finally, line 28 would check whether SWI #3 was called, which turns off all 8 LED's if they were on by branch linking to line 43.

```
1        GLOBAL SWI_HANDLER
2        IMPORT user_code
3                AREA SWIHANDLER, CODE, READONLY
4
5    SWI_HANDLER
6    PINSEL0 EQU 0XE002C000
7    IO0PIN  EQU 0XE0028000
8    IO0SET  EQU 0X4
9    IO0DIR  EQU 0X8
10   IO0CLR  EQU 0XC
11   LEDMASK EQU 0xFF00
12   CLOCK   EQU     12000000
13   DELAY1S     EQU     (CLOCK/4)
14           STMFD SP!, {R0-R6, LR}
15           SUB R0, LR, #4
16           LDR R1, [R0]
17           BIC R1, #0XFF000000
18           TEQ R1, #1
19           BLEQ LED_LIGHT
20           LDR     r0, =DELAY1S
21   LOOP3   SUBS    r0, r0, #1
22           BNE     LOOP3
23           TEQ R1, #2
24           BLEQ user_code
25           LDR     r0, =DELAY1S
26   LOOP4   SUBS    r0, r0, #1
27           BNE     LOOP4
28           TEQ R1, #3
29           BLEQ LED_OFF
30           LDMFD SP!, {R0-R6, PC}^
31
32
33   LED_LIGHT
34           STMFD SP!, {R0-R6, LR}
35           MRS R0, CPSR
36           STMFD SP!, {R0}
37           LDR R3, =IO0PIN
38           MOV R2, #LEDMASK ;turn LED's on
39           STR R2,[R3,#IO0CLR]
40           LDMFD SP!, {R0}
41           MSR CPSR_f, R0
42           LDMFD SP!, {R0-R6,PC}
43   LED_OFF
44           STMFD SP!, {R0-R6, LR}
45           MRS R0, CPSR
46           STMFD SP!, {R0}
47           LDR R3, =IO0PIN
48           MOV R2, #LEDMASK
49           STR R2,[R3,#IO0SET];TURN ALL OFF
50           ;LDMFD SP!, {R0}
51           ;MSR CPSR_f, R0
52           LDMFD SP!, {R0-R6,PC}^
53           END
```

Figure 10.4: Checking Software Interrupt calling instructions

Within the mycode file lives the calling software interrupts and the initiation of the LED's by setting the respective pins as outputs and turning the LED's off. From lines 13 through 22 in figure 10.5, the LED's are initialized and set off for software interrupt 1, which lights all LED's. Line 23 creates Software Interrupt 1, which branch links to the SWI_HANDLER file, checking through which SWI # was called and does its corresponding task. A delay is implemented after each SWI # calling instruction. Line 31 was implemented to branch back from the LCD file to the main mycode file to create the calling SWI #3.

```
1        GLOBAL SUBRTS
2        GLOBAL BRANCHBACK
3            AREA mycode, CODE, readonly
4    SUBRTS
5    PINSEL0 EQU 0XE002C000
6    IO0PIN  EQU 0XE0028000
7    IO0SET  EQU 0X4
8    IO0DIR  EQU 0X8
9    IO0CLR  EQU 0XC
10   LEDMASK EQU 0xFF00
11   CLOCK           EQU     12000000
12   DELAY10U        EQU     (CLOCK/400000)
13           MOV r2, #0
14           LDR r3, =PINSEL0
15           STR r2,[r3]
16           LDR R3, =IO0PIN
17           LDR R4, [r3,#IO0DIR]
18           LDR R2,= 0xFF00
19           ORR R4,R4,R2
20           STR R4, [r3,#IO0DIR]
21           MOV r2,    #0x0000FF00;turns LEDs off
22           STR r2, [r3,#IO0SET]
23       SWI #1
24           LDR     r0, =DELAY10U
25   LOOP    SUBS    r0, r0, #1
26           BNE     LOOP
27       SWI #2
28           LDR     r0, =DELAY10U
29   LOOP2   SUBS    r0, r0, #1
30           BNE     LOOP2
31   BRANCHBACK
32       SWI #3
33   STOP B STOP
34       END
```

Figure 10.5: mycode file containing LED initialization and SWI calling instructions

Discussion

For SWI, we made a separate SWIHandler subroutine that gets called when SWI is called. It also decodes the SWI number in order to properly handle the SWI subroutine as needed. For our SWIHandler, we had implemented four cases which correspond to four codes. We had three codes that are hard coded and one case that is the default code for all of the other codes that weren't implemented.

For task 2, SWI #1 is hard coded to light all 8 LED's which are initialized to be off within the main mycode file. Within the SWI_HANDLER file, the SWI # is decoded to figure out which interrupt number was called and runs through their corresponding tasks. Task 3 implements the previous LCD screens experiment codes, which lights the LCD screen and displays "BEST // PROFESSOR" on separate lines. Task 3 also asks students to implement LED's, so SWI #3 would turn off all 8 LED's that were toggled within task 2.

For IRQ task 4, most of the code was implemented in the myStartup code. After talking with professor Flynn, we should have instead implemented the code in a different subroutine much like what we did for the SWIHandler subroutine.. In order to configure our IRQ properly, we had to initialize a lot of registers in order to implement it the way we want it. In order to use IRQ, we must first initialize it in the startup code.

Conclusion

For the Software Interrupt, the most important instruction that is needed to be included is that the interrupt handler would need to return the control to the calling program. The steps for returning the control to the calling program includes copying LR to PC and copying SPSR to CPSR. For the external interrupt inputs initialization, the necessary steps are configuring the pin function, selecting the edge/level of sensitivity, choosing the signal polarity, and clearing the EINT flags. Lastly, we also need to decide the interrupt type generated by the EINT inputs, whether it is a normal interrupt request (IRQ) or a fast interrupt request (FIQ).