Spring 2019

California State University, Northridge

Department of Electrical and Computer Engineering

Lab Experiment 10: Interrupts

May 06, 2019

Professor: Neda Khavari

Authors:
Ridge Tejuco
Dan Paul Rojas

Introduction

The purpose of this lab is to understand and become familiar with interrupts, how to set them up and call them. The experiment includes three modes that handle interrupts which are SVC, IRQ, and FIQ.

The experiment also covers how to setup the stacks for each individual mode that can be used. Then the experiment covers writing the file handlers and entering these interrupts.

For FIQ and IRQ, the Vectored interrupt controller is set up and used for the external interrupts. The external interrupts are input through the joystick on the board which are p0.16 and p0.20.

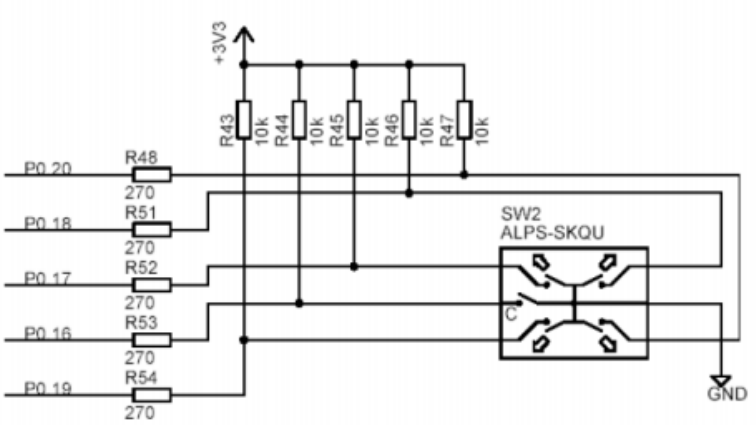

Figure 2. LPC2148 Education Board Schematic: Joystick-switch.

Procedure

First, the stack for each individual mode used in this experiment was set up in the Startup file with the following

```
USR_Stack_Size EQU 0x100
Len_SVC_Stack EQU 0x200
Len_IRQ_Stack EQU 0x300
Len_FIQ_Stack EQU 0x400
Mode_SVC EQU 0x13
Mode_IRQ EQU 0x12
Mode_FIQ EQU 0x11
SRAM EQU 0X40000000
Stack_Top EQU SRAM+USR_Stack_Size
Stack_SVC EQU SRAM+Len_SVC_Stack
Stack_IRQ EQU SRAM+Len_IRQ_Stack
Stack_FIQ EQU SRAM+Len_FIQ_Stack
     MSR CPSR_c,#Mode_FIQ+I_Bit+F_Bit
     LDR SP,=Stack_FIQ
     MSR CPSR_c,#Mode_IRQ+I_Bit+F_Bit
     LDR SP,=Stack_IRQ
     MSR CPSR_c,#Mode_SVC+I_Bit+F_Bit
     LDR SP,=Stack_SVC
     ;Enter User Mode with interrupts enabled
     MOV r14, #Mode_USR
     BIC r14,r14,#(I_Bit+F_Bit)
     MSR cpsr_c, r14
     LDR SP, =Stack_Top
```

The loop for SWIHandler branch was commented out and the SWIHandler branch file was imported.

```
     IMPORT SWI Handler
;SWIHandler B SWIHandler
```

For task 1 through 3, the SWI handler file was written. The address of the previous instruction was saved in R0 and then the top 8 bits were masked off, to retrieve the SWI command number.

```
     AREA swi_code, CODE, READONLY
     GLOBAL     SWIHandler
SWIHandler
     STMFD      sp!,{r0-r12,lr}    ; Store registers.
     LDR        r0,[lr,#-4]        ; Calculate address in R0
     BIC        r0,r0,#0xff000000    ;Mask off top 8 bits
```

R0 is compared to our expected SWI command values to branch to the right service.

```
     CMP R0,#0x1
     BEQ LED_ALL
     CMP R0,#0x2
     BEQ LED_TOP4
     CMP R0,#0x4
     BEQ LED_ALT
```

The following LED outputs were used and branched to.

```
LED_ALL
     ORR R1,R1,#0xFF00
     STR R1,[R2]
     B EXIT
LED_TOP4
     ORR R1,R1,#0xF000
     STR R1,[R2]
     B EXIT
LED_ALT
     ORR R1,R1,#0xAA00
     STR R1,[R2]
EXIT
     LDMFD      sp!, {r0-r12,pc}^
     END
```

In the user code, since, we are using LEDs in our interrupts. Pins 8 through 15 are set to GPIO. And the pins are set as outputs.

```
     AREA user_code, CODE, READONLY
     MOV R1,#0
     LDR R2,=PINSEL0; ALL pins in GPIO Mode
     STR R1,[R2]
     LDR R2,=IO0DIR
     LDR R1,=0xFF00; PINS 8-15 as output
     STR R1,[R2]
     LDR R2,=IO0PIN
     LDR R1,[R2]
```

For task 1 - 3, the user code contained the calls for SWI interrupts with delays in between them.

```
     SVC #0
     LDR R6, =0x0008FFFF
DELAY_ONE
     SUBS R6,R6,#1
     BNE DELAY_ONE
     SVC #1
     LDR R6, =0x0008FFFF
DELAYs
     SUBS R6,R6,#1
     BNE DELAYs
     SVC #2
```

For task 4, the p0.16 and p0.20 were set up as external interrupts in the user code by wring 0x301 to PINSEL1.

```
SET_PINSEL1
     LDR R0,=PINSEL1
     LDR R1,[R0]
     BIC R1,R1,#0x2
     LDR R2,=0x301
     ORR R1,R1,R2
     STR R1,[R0]
```

Then p0.16 and p0.20 were set to high and edge-sensitive

```
SET_EMOD
     LDR R0,=EXTMOD
     LDR R1,[R0]
     ORR R1,R1,#0x09
     STR R1,[R0]
SET_EPOLAR
     LDR R0,=EXTPOLAR
     LDR R1,[R0]
     ORR R1,R1,=0x09
     STR R1,[R0]
```

Then the Vectored interrupt controller was set up. EINT3 was set to FIQ by writing ‘1’ to bit 17 and EINT0 was set to IRQ by writing a 0 to bit 14.

```
     LDR R0,=VICINTSELECT
     LDR R1,[R0]
     ORR R1,R1,#0x20000; EINT3 is FIQ; EINT0 is IRQ
     STR R1,[R0]
```

Then the vic interrupts were enabled. The program enters a loop waiting for external interrupts.

```
     LDR R0,=VICINTENABLE
     LDR R1,[R0]
     ORR R1,R1,#0x24000
     STR R1,[R0]
stop b stop
     end
```

In the IRQ Handler,the interrupt flag is cleared. Then the LEDs are set and then the program returns to the last instruction.

```
     GLOBAL IRQHandler
     AREA irq_handle,CODE,READONLY
IRQHandler
     STMFD SP!, {R0-R12,LR}
     MRS R8,CPSR
     STMFD SP!,{R8}
CLR_EINT
     LDR R0,=EXTINT
     LDR R1,[R0]
     ORR R1,R1,#0x1
     STR R1,[R0]
LEDS
     LDR R0,=IO0PIN
     LDR R1,[R0]
     BIC R1,R1,#0xFF00
     ORR R1,R1,#0x4800;  LEDS = '01001000' eg. 123
     STR R1,[R0]
Return
     LDMFD SP!,{R8}
     MSR CPSR_f,R8
     LDMFD SP!,{R0-R12,LR}
     SUBS PC,LR,#4
```

Lastly, the FIQ handler is very similar to the IRQ handler. The interrupt flag is cleared and the LEDs are set to a new pattern. By clearing the flag, it stops the interrupt from looping infinitely.

```
EXTINT EQU 0xE01FC140
IO0PIN EQU 0xE0028000
     GLOBAL FIQHandler
     AREA fiq_handle,CODE,READONLY
FIQHandler
     STMFD SP!, {R0-R12,LR}
     MRS R8,CPSR
     STMFD SP!,{R8}
CLR_EINT
     LDR R0,=EXTINT
     LDR R1,[R0]
     ORR R1,R1,#0x8
     STR R1,[R0]
LEDS
     LDR R0,=IO0PIN
     LDR R1,[R0]
     BIC R1,R1,#0xFF00
     ORR R1,R1,#0x1200;  LEDS = '00010010' eg. 321
     STR R1,[R0]
RETURN
     LDMFD SP!,{R8}
     MSR CPSR_f,R8
     LDMFD SP!,{R0-R12,LR}
     SUBS PC,LR,#4
     END
```

Results & Conclusions

For task 1, looking through memory, a full descending stack is setup. For task 1, the goal of setting up a Full descending stack for each individual mode used throughout the experiment was successful.

For task 2 and 3, the SWI interrupts display the correct LED patterns outputted when the interrupts are called in software. In task 2 and 3, the program successfully runs through all SWI calls. The experiment was successful in calling the SWI from user code and entering the SWI handler. Each swi code was successfully branched to, and then returned from.

For task 4, when pressing the joystick down, the LEDs output a "01001000" which is the expected value. And when shifting the joystick down-right, the LEDs output a "00010010" which is the correct value. In task 4, the VIC and external interrupts were successfully set up. Using the VIC, the IRQ and FIQ interrupts were successfully handled.

Questions

1. If MOV is used instead of MOVS, SPSR_svc will not be copied back to CPSR.

2. The signal type used in the lab is high and edge sensitive for a rising-edge.

3. If the flags are not cleared, the interrupt will loop infinitely to the handler.

4. The address of the subroutine call will be put in the Link register, therefore the link register needs to be restored back manually.