ECSE324 Lab2 Report - G66

Gengyi Sun , Liang Zhao

November 2019

1 Creating the Project in Altera Monitor Program

Before getting our hands on the coding part, folders with correct naming and hierarchy must be created. As a priority to compile and execute the program, the folders must also contain the correct file as well. In a high-level language IDE, the folders are usually created as the project is created. Then a C file named main is placed under the top-level folder. All the assembly files should be placed in folder asm, and all header files should be placed in folder inc.

2 Basic I/O

2.1 Slider Switch and LED

This part requires the corresponding LED is turned on when a slider switch is turned on. This part consists of a main.c file, an LED.s file, a LED.h file, a slider_switch.s, and a slider_switch.h file. Main.c should include all the header files(.h file). The header files should specify subroutine names, inputs, and outputs. When main.c calls read_LEDs_ASM() subroutine, read_LEDs_ASM() will load the values at LED memory location into R0 and then branch to LR. The wirte_LEDs_ASM() will store value passed to R0 and then branch to LR. The word 0xFF200040 is holding slider switch status in the memory. The word 0xFF200000 is storing LED status in the memory. Since the bit alignment of slider switches and bit alignment of LED display are perfectly corresponded, the status of slider switches can be loaded to LED memory location directly. This is the easiest part of the lab because the codes are provided.

2.2 Drivers for Hex Display

The 6 HEX displays are controlled by 2 words in the memory 0xFF200020 and 0xFF200030. Each HEX display is taking 8 bits in the word but only 7 of them are controlling the display. First we decided the digit displays from 0 to F. Then compare the input number "val" deciding the display. Since HEX is encoded with one-hot encoding, deciding with HEX display should be turned

on can be done by a series of subtraction starting from HEX5. Therefore no interference will occur during execution. When the two elements are decided, register contains value of "val" is shifted to the corresponding HEX memory location. An AND operation is performed to clear the right HEX memory, then an ORR operation is performed to load the correct value. As for the clear operation, we use AND operation with 0s to clear the corresponding bits. For the flood operation, we use ORR operation with 1s to turn all digits on.
After finishing the lab, we consider the codes in this section could be provided

```
HEX_write_ASM:
                        //R0, HEX, R1 val
                        PUSH {R4-R7}
                        LDR R5, =LED_SEG1       //init address
                        LDR R6, =LED_SEG2
                        LDR R2, [R5]
                        LDR R3, [R6]

                        MOV R7, #0b00111110 //DEFAULT U: indicates error
                        CMP R1, #0
                        MOVEQ R7, #0b00111111

                        CMP R1, #1
                        MOVEQ R7, #0b00000110
```

Figure 1: Segment of deciding value to be writed into HEX display

as study materials, or at least the number display value should be provided. This can help with writing codes in the timer section because figuring out the correct digit to display a number is not a learning experience compare to writing timers. HEX display section is also adding working hours of the lab.

```
HEX0_C: MOV R3, #0b11111111111111111111111100000000
        SUB R0, R0, #0b1
        AND R1, R1, R3
        STR R1, [R5]
        POP {R4-R6}
        BX LR
```

Figure 2: Segment of clear all HEX displays

```
HEX0_F: MOV R3, #0b00000000000000000000000011111111
            SUB R0, R0, #0b1
            ORR R1, R1, R3
            STR R1, [R5]
            POP {R4-R6}
            BX LR
```

Figure 3: Segment of flood all HEX displays

2.3 Drivers for Push-buttons

Read_PB_data_ASM: the memory values representing the push-buttons is returned as a binary sequence.
PB_data_is_pressed_ASM: check if the indicated buttons are pressed.
PB_edgecap_is_pressed_ASM: check if the indicated buttons are pressed.
Read_PB_edgecap_ASM: the value of the edge-capture memory location.
PB_clear_edgecap_ASM: clears the edge-capture memory location.
Enable_PB_INT_ASM: enables interrupts.
Disable_PB_INT_ASM: disables interrupts.
The coding part is the same to the codes in HEX display: load or store memory sequence in the specified location. Thus we had no difficult completing this part.

```
read_PB_edgecap_ASM:    //
        PUSH {LR}
        LDR R0, =PUSH_edge
        LDR R0, [R0]
        POP {LR}
        BX LR
```

Figure 4: Segment of read push-button memory

3 Timers

There are four timers built in the FPGA board, with frequency at 100MHz, 100MHz, 25 MHz and 25 MHz respectively. The assembly codes have configuration, clear and read subroutines. The subroutine being called by the C program takes an input struct, then the assembly code checks for which timer is being used according the first element in the struct. Then it will update time

based on the time_out(the second element in the struct). Since timers have different frequencies, the constant that is determining timing interval should be altered accordingly. Finally we load I, M and E bits into the correct memory locations.
Despite the similar operations among configuration, read and clear, the subroutines have differences as follows:
HPS_TIM_read_INT_ASM: reads the correct s-bit in memory then loads it to the lower bits of R0.
HPS_TIM_clear_INT_ASM: clears the values in the chosen timer.

4 Interrupts

```
hps_tim0_int_flag:
        .word 0x0

hps_tim1_int_flag:
        .word 0x0

hps_tim2_int_flag:
        .word 0x0

hps_tim3_int_flag:
.word 0x0

pb_int_flag:
        .word 0x0
```

Figure 5: Segment of setting interrupt flags

In polling, the timer is updated in a while loop by calling the subroutines repeatably, for which the simple updating operation consumes the most resources. Polling is also not sensitive enough to react correctly when push-buttons are pressed rapidly since it depends on the second timer to poll the status of the push-buttons.
With interrupts, the signals indicating whether an event occurs are sent to the processor directly. This reduces the time delay we had in polling timers. When the signal is received, the code pauses its execution and handles the event. Thus an Interrupt Service Routine must be implemented to handle the event. Similar to all assembly routines, a .s file and a .h file are placed in the correct level folders.