ECSE324 COMPUTER ORGANIZATION

Lab3 Report

Group 29

Yi Zhu

260716006

Shaluo Wu

260713923

2018-10-31

This lab consists of four main parts. First part is the introduction of how to create a project in Intel FPGA Monitor Program. The second and third parts introduce some basic I/O features on the FPGA board, including the slider witches, LEDs, pushbuttons and 7-segment displays. In these two parts we are also asked to write both ARM language code and C language code to test how to control these basic I/O features on the FPGA board. The last part of the lab is to learn how to use the interrupt driver service to control a stopwatch.

1. Creating a project in the Intel FPGA Monitor Program

In this part we are asked to properly structure a project. It is relatively important that in any project the driver code relies on the project structure and the code will not compile or run if the project structure is not well organized. We followed the instructions provided on myCourses and followed it step by step.

Since the procedure is straightforward, we did no meet any challenges in this part of the lab. It is an introduction and learning process part, so there is nothing more we can improve.

2. Basic I/O

2.1 Slider Switches and LEDs

In this part of the lab we learned that there are 10 slider switches on the FPGA board using 10 wires connected to the bus which each carry either a logical ‘0’ or ‘1’. The value for these switches can be written and saved in the memory at SW_BASE, which is 0xFF200040 in this case. The code and procedure are provided on myCoureses, so we followed it step by step to finish it. Then we moved on to the LEDs part. We are asked to display the state of each switched on corresponding LED on the board. We referred to the sample code provided in slide switches and wrote our own read_LEDs_ASM and write_LEDs_ASM methods. The general working mechanism of these methods is the same as the sample codes in slider switches that to read the value in the memory address of the LEDs which is 0xFF200000 and then write the value that we want into the memory address of the LEDs. Only the write_LEDs_ASM method is slightly different that it will have to accept an integer input from return value in the read_slider _switches_ASM method. After all setup, we put them together in the mian.c class. This method can read the value we received from slide switched and then write it in to the corresponding LED lights. Then we tested it on out FPGA board and everything went well.

The whole process is straightforward and relatively easy to be achieved. We did not encounter any challenges. Except we may consider using some PUSH and POP method for these registers that we will use in the read_LEDs_ASM and write_LEDs_ASM methods. We do not think there is further methods that can be improved.

2.2 HEX Displays

This part of the lab asks us to implement three methods for out 7-segments display on our FPGA board: HEX_clear_ASM, HEX_flood_ASM and HEX_write_ASM. We also need to use enumerations in C language. To implement these methods, we first need to figure out how the 7-segments display works. It consists two memory addresses for displaying number that from HEX0 to HEX3 use 0xFF200020 and from HEX4 to HEX5 use 0xFF200030. After referring to the DE1-SoC Computer manual and recalling the materials we learned in ECSE222 class, we generated a reference table that have target display decimal numbers and corresponding binary and decimal data in the data register for 7-segments display. The table is shown as below:

Decimal	Hex	7-Segments Display B	7-Segments Display D
0	0	0b00111111	63
1	1	0b00000110	6
2	2	0b01011011	91
3	3	0b01001111	79
4	4	0b01101101	102
5	5	0b01101101	109
6	6	0b01111101	125
7	7	0b00000111	7
8	8	0b11111111	127
9	9	0b01101111	111
10	A	0b01110111	119
11	B	0b11111111	127
12	C	0b00111001	57
13	D	0b00111111	63
14	E	0b01111001	121
15	F	0b01110001	113
Flood		ORR 0xFFFFFFFF	
Clear		AND 0x00000000	

Table 1 7-Segments Display Reference

After generated the reference table, the next step is to implement the logic to display flood, clear and specific number on the 7-segments display. Since all these outputs depend on the input of which HEX display we want to use, we will check the HEX display first. In the HEX_clear_ASM method, we first checked the input register R0 that whether HEX0 is included, if not we will jump to check HEX1. If yes, then well use AND 0xFFFFFFF00 to clear HEX0. We will do it step by step for each HEX display and at the end we will store the final result in the memory address. In the HEX_flood_ASM method, the general logic is the same except instead of using AND 0xFFFFFF00 to clear, we will use ORR 0x000000FF to flood HEX0. In the HEX_write_ASM, it is slightly different that we will check the input number that we want to display in R1 and then convert it to the number we will use for 7-segments display according to our reference table. Then we will display it on the target 7-segment display using the same step as clear and flood.

The biggest challenge we encountered during the process is to find how to use logic shift to check which HEX display we want to use. The process of generating the reference table is long but the method is straightforward. Another problem we met is how to move the 7-segment display from HEX0 to HEX3 address to another HEX4 to HEX4 address. We solved it by using additional register R5 to store the value we want to display for HEX4 to HEX5 display. Lastly, we found another trick hat in the HEX_write_ASM method, we need to clear the display before writing the value. Otherwise, the display will crash and show strange numbers.

We believe there is still space for improvement in the HEX_displays method that currently we are using many loops to achieve checking numbers and HEX display. There should be another way which is similar to the 'for loop' in C language that can be implemented. Since the time is limited we cannot generate this method.

2.3 Pushbuttons

This part of lab asks us to implement functionality of pushbuttons on our FPGA board. We considered it same as the method we implemented in slider switches and checked the method provided on API. Then we wrote our code in the pushbuttons ARM language file. The read_PB_data_ASM function will return the pushbuttons that are pressed and the PB_data_is_pressed_ASM will return the status whether the button is pressed. The function is same as read_PB_edgecap_ASM and PB_edgecap_is_pressed_ASM. We followed the API to finish all other methods in this file.

There are some difficulties when we started writing these codes in the file. After reviewing related sections on DE1-SoC Computer manual and API online, the process is straightforward. We do not think we can have better way to improve it.

2.4 Combining

The next step of the lab is to combine these parts we wrote together to achieve a push-display-number function. In this function we will receive the value that we want to display from the slider switched and the HEX display we want to use by pressing corresponding pushbuttons. Since all relevant I/O implement methods have been done in previous parts of the lab, we will simply write the logic in the main.c file. We used multiple 'if statement' to achieve the logic we want. First, we receive the number we want to display from slider switches, then we check whether this number is zero, if it is then only flood HEX4 and HEX5. If it is not, then we check whether it is less than 16 since we can only display number less 16 on the 7-segments display. If it is greater than 16 we will display nothing from HEX0 to HEX3, otherwise we will check which pushbutton is pressed. If any button is pressed, then the number will be written into corresponding 7-segments display. Lastly, if the left most slider switch is on, which means the number we received will be larger or equal to 512, we will clear all the displays.

The process of implementing such logic in C language is relatively easy. The only slight difficulty we encountered is to figure out that the signal of slider switches can be represented using binary or corresponding decimal numbers.

3. Timer

In this part of the lab we are asked to build an HPS timer driver and implement a stopwatch in the main.c file. Both the ASM file and header file of HPS_TIM ASM are provided, and we also have a sample timer program, the process is relatively clear. We first implemented a timer generally same as the sample provided. Instead of clearing the display when timer reaches 16, we use the standard timing in the real world. We also initialized a list of variables that we will use for counting time. One additional variable is 'enable', which is used as a Boolean value to let the timer to keep going. Since it asks us to implement the timer moving in milliseconds, we reduced the timeout to be 1000, which means each time the million second will be increased 1. We display 10 milliseconds on HEX0 and 100ms on HEX1 and seconds on HEX2 and HEX3 and minutes on HEX4 and HEX5. Every time there is an overflow in any time unit, there will be a shift. When the minute reaches the limit, everything on the display will be zero and the timer will restart. Then we implemented another timer for stop the previous one. In the second timer, we added the pushbutton for more functions on our timer. We also used 'if statement' in C language to implement the conditions. If the PB0 is pressed and the timer is stopped, it will start the timer. If the PB1 is pressed and the timer is working, then let the enable to be zero so stop the timer. If the PB2 is pressed, reset everything to zero.

The greatest challenge in this section we met is to get the algorithm of timer working mechanism. Since the implementation is also straightforward, we think the best improvement is to use interrupt for the stop timer to utilize processor resources.

4. Interrupt Stopwatch

This part is generally similar to the 3rd part of the lab. The only difference is that in polling timer the action acts as soon as the button is pushed but in this part the action acts once the button is released. Since almost all files are provided via myCources, the only code we need to add is in ISR.s file and main.c file. To active the pushbutton interrupt method, we added code in the FPGA_PB_KEYS_ISR, which has the interrupt ID to be 73. We set the flag is up when the pushbutton is pressed and when the interrupt is done, clear the interrupt flag. In the mian.c file, we set almost the same thing as in the previous section. Instead of using two timers, we only use one timer and the interrupt method in this section.

The challenge we met in this section is to find out the working mechanism of interrupt. But after when we checked the DE1-SoC Computer manual and examples online, we figured it out and we believe it's the best way and no more to be improved.

5. Conclusion

From this lab, we generally understood how basic I/O on the FPGA board works and how to program and control them. We also have a better understanding on polling and interrupt.