ECSE 324: Computer Organization
Lab2 Report

Part 1: I/O
Part 2: Timers
Part 3: Interrupts

Grey Yuan (260857954)

Part 1

Approach

Following the beginning of the lab manual, I first write the subroutines of read_slider_ switches_ASM and write_LED_ASM. Then, I continue to write HEX_write_ASM with two loops to write HEX_MEMORY_1 and HEX_MEMORY_2. Then, using loop1 and loop2 that have already been written I can use any of them to implement HEX_clear_ASM and HEX_flood_ASM. Then I implement all the 7 subroutines for pushbuttons in order to control them.
In the main method, I call the functions implemented above to accomplish the desired feature of this part. First, I have to read the slider switches input using read_slider_switches_ASM from the input register that I stored the input in _start. Then, we store the input to initialize the register that we used to display with HEX. We directly branch to display in that case. If SW9 is on, we set R0 = #512 and HEX will be 880000. Next, we need to check whether pushbuttons have been asserted. To do that, we need to call PB_edgecp_is_pressed_ASM to check and clear PB with PB_clear_edgecp_ASM after each of that happening. We keep track of the input register and the display register. When the input register (in my case R0) is stored with #1, #2, #4, #8, we will call "display" and go back to "read" to continue reading slider switches.

Challenge

Since there are many subroutines in this part, we have limited number of registers, we need to keep overwriting them. I solved this problem by using caller-callee conventions (push and pop) in my subroutines.

Possible Improvements

I currently do not see how we can improve the code more. For coding style, I think I have done more subroutines to simplify my code. It is also possible to also use subroutines to increase the readability of my main method.

Testing

After we compile and run the program, we should immediately get "88----" with HEX4 and HEX5 are both lighted up. After we press and release PB0, we should see "88---9" with HEX4 and HEX5 are all taken and HEX0 is 9.

Part 2

Approach

At the start of the main method, we initialize the time port by loading 2000000 to R0. As polling has suggested, the program will iterate to check the status of pushbuttons. We need to use to clear up the register of the edge capture with PB_clear_edgecp_ASM. In "read", we judge if Pushbutton 2 is pressed: if yes, we will reset the load value and call HEX_write_ ASM to set all the display register to zero and write them to the HEX. Similarly, if Pushbutton 0 is pressed, we will set the register to enable the timer to start and set the register to 0 to stop the timer. Then, we will keep iterating to check F using ARM_TIM_read_INT_ ASM and update HEX if we have F=1.
Since HEX displays are from HEX0 to HEX5, we would use 6 registers to pass to subroutine "display". Before we display the values, we first need to add 1 to the register corresponding to the right-most digit and set the register to 0 when it is equal to 10. Also, we need to remember to set the register correspond to the left digit of it to 1. This processing procedure is also applied to all other HEX displays. After we finish processing all HEX display registers for this loop, we proceed to display all values in the registers with HEX_write_ASM.

Challenges

The process of learning about how timer works in ARMv7 is tough, but once we learned how the parameters are passed and displayed, it becomes easier later to write the subroutines and use them to achieve the timer.

Possible Improvement

We could automatically reset all HEX to 0 if we have reached the maximum display time for the timer. We could add a condition check to simply achieve this. There might also be simpler way to write the "read" subroutine described in "Approach".

Testing

After compile and run the program, we should see "000000" displayed on the LEDs. After pressing and releasing PB0, we will get the timer to start running. If we press PB1 when the timer is running, we will pause it. If we press PB2 when the timer is paused, we will reset all HEX display to 0. We could keep doing the process as long as the program is running.

Part 3

Approach

The purpose of using interrupt instead of polling is avoid iterating to check pushbuttons, so the request will only be handled when there is an interrupt (PB input). The main logic of interrupt is that, if we receive 29 in Interrupt_check, the timer will run normally and the way we use the subroutines to change HEX display is similar to "Approach" written in the previous part. Something different is that, in IDLE, we use PB_int_flag and tim_int_flag to check PB input. Otherwise, if we receive 73 in Pushbutton_check, we will check which PB is pressed, and start, stop, or reset the timer accordingly.

Challenge

Most of the subroutines are provided in the lab manual and the previous part. After following the steps in the lab manual and understanding the concept of interrupt, I do not have too many problems with implementation.

Possible Improvement

Similar to "Possible Improvement" in Part 2.

Testing

Similar to "Testing" in Part 2.