ECSE 324: Computer Organization Lab 2: I/O

Hongtao Xu 260773785

Part 1: Basic I/O

Brief description:

Part 1 basic I/O is simple to understand and straightforward. The goal is to write a program that puts Switches as input, put LEDs and Seven-segment displays as output, and put the Push button as the on-off button. There are 10 LEDs in total which correspond to the 10 slider switches above them. The LEDs are virtual memory based on address 0*FF200000, and the Switches are based on the address of 0*FF200040. Both of the LEDs and the slider switches are embedded inside the simulator, which acts as the slaves on the IO system, while we are only using the 0-3 positions of them.

In specific, when we select the 0-3 digits on Switches, the corresponding lights' positions on LEDs will turn on, if we unselect the digits, the LED lights will turn off again. Also, after I select the digits on Switches, every time I press the 0 on Push Button twice, the corresponding Hex number will appear on the first position of Seven-segment displays. For example, if I select 1 and 3 on Switches, the corresponding lights on LEDs will turn on immediately, after I press 0 on Push buttons twice, there will be an A appear on the first position of Seven-segment displays (2^1+2^3=10). The max value we can select on Switches is 2^0+2^1+2^2+2^3=15, which plays F(15) on the first position of Seven-segment buttons, only the first digit of it is used in Part1.

Approach taken:

I am writing 2 drivers for HEX displays and Pushbuttons. The first one includes a clear function that could turn off all the segments of the HEX displays passed in the argument named HEX_clear_ASM, a flood function that could turn on all segments named HEX_flood_ASM, and a write function that displayed desired number on the selected display name HEX_write_ASM. There are 7 subroutines on the Pushbuttons driver used to read, remember and clear the state of the pushbuttons and provide information for the HEX display driver to display. In specific, read_PB_data_ASM returns the indices of the pressed pushbuttons; PB_data_is_pressed_ASM receives pushbuttons indices as an argument; ead_PB_edgecp_ASM returns the indices of the pushbuttons that have been pressed and then released; PB_edgecp_is_pressed_ASM returns 0*00000001 when pushbutton has been asserted; PB_clear_edgecp_ASM clears the pushbuttons Edgecapture register; enable_PB_INT_ASM enables the interrupt function for the pushbuttons by setting interrupt bits to '1'; disable_PB_INT_ASM disables the interrupt function for the pushbuttons by setting the interrupt mask bits to '0'. After compiling code in those parts, the HEX4 and 5 are always on and show the number 88, the hexadecimal numbers displayed in HEX0 after selecting numbers on Switches. The clear and flood functions in the HEX display driver implement that the addresses are passed to a register and then compared with the one-hot encoding of the displays from HEX5 to HEX0. The corresponding HEX display will be turned on/off after the 1st display address that is smaller than the past address is discovered, while the past addresses are also subtracted by the smaller address and sent to a smaller HEX address for comparing and searching for the following selected HEX display. The hexadecimal number through encoding table displayed by the write function. Next, the Pushbutton driver is about writing and reading information to the memory address, which includes the edge capture and the Pushbutton memory address. The driver includes functions for enable and disables interrupt function, which means 0/1 corresponding memory address in the Push buttons. In this way, the diver could clear the un-send bits in memory address by using BFC.

Challenges and possible improvement:

There is a hard process trying to find a way that displays the hexadecimal numbers. Firstly, I just turn on all the segments that I need while they did not function well. Then I realized that the displays for the hexadecimal numbers would not appear through passing specific address numbers to the HEX display memory address.

Meanwhile, I used BFI and BFC to display my numbers, which can clear unused bits in memory addresses and reset some register values.

Part 2-1:

Brief description:

This part introduces an I/O device called Cortex-A9 private Timer, it is primarily a down counter that takes an initial value in the load register to start with. This timer decreases the value with a frequency of 200MHZ and when it reaches 0, the F bit becomes 1. The purpose of this part is to build a timer count from 0 to 15(F) on HEX0, when it reaches 15(F), the timer back to 0 again and keeps counting.

Approach taken:

I used some functions from part 1 to build the ARM A9 private Timer. I used ARM_TIM_config_ASM to build the configuration of the timer firstly, then I used ARM_TIM_read_INT_ASM to read the interruption and used ARM_TIM_clear_INT_ASM to clear all the interruptions. Also, there is a subroutine named HEX_write_ASM, which is used to count from 0 to 15(F) on the HEX0 of Seven-segment displays. I set the initial count value to 20 million, and the timer can update its count value every one second by the interruption.

Part 2-2:

Brief description:

This part is to design a polling-based stopwatch that takes the same features used in the previous part, while this part is to design a stopwatch that has six digits to compute with. In the frontal 3 digits on the Seven-segment display (HEX0-HEX2), once we reach 10, we reset this digit to 0 and increase the upper digit by 1, while for the 4th digit (HEX3), we increase the next digit when we reach the number 6 instead of 10, and for the HEX4, we reset and increase next digit once we reach 10 again. All 6 HEX displays are used to representing the time, the frontal two digits are used for 10 and 100 milliseconds respectively, the next two digits (HEX2 and 3) are used for representing seconds, and the last two digits are representing for minutes. There are three buttons used in the Push button. Button 0 is for starting/restarting the stopwatch, button 1 is used to stop the stopwatch, and button 2 is to reset the stopwatch.

Approach taken:

For this part, I used all the three subroutines from part 2-1: ARM_TIM_config_ASM, ARM_TIM_read_INT_ASM, ARM_TIM_clear_INT_ASM and HEX_write_ASM. At first, we initiate the timer as usual by storing the control register with a variable. We take R0 to R5 on the register to record the value presented by the HEX display from HEX0 to HEX, and we make sure that all register digits start from 0. We then check the F bit from the timer. If it is 1, I record the leading result and increase the least bit of the stopwatch. Then I clear the interrupt F bit by using a clear subroutine. I wrote subroutines for each HEX (HEX_write_ASM_HEX0 to HEX_write_ASM_HEX5) that records the value for each HEX and increases the value of the next HEX if one arrives 10/6. The status of the pushbuttons is checked after updating the displays. There are also some other subroutines, the subroutines begin is used to begin the stopwatch, the subroutine next is to release the stopwatch, and the subroutine re_count is to restart the stopwatch. Overall, this part is the assembly of the frontal 2 parts of the lab.

Challenges and possible improvement:

The challenge I met here is to understand the configuration structure of the stopwatch. To be specific, I spend lots of time doing research online and read the DE1-SoC_Computer_ARM to find out how the stopwatch functions work. When I compute it, I make mistakes about the F bit, I don’t know the F bit is updated by itself while I tried to change the F value by myself. Also, I face challenges with displaying the numbers for the stopwatch. When

displaying the numbers, I tried to compute all 6 HEX numbers in one subroutine.it is very hard because I barely using the push and pop register. To fix the problem, I write the subroutines for each HEX separately instead of one large subroutine. The possible improvement is deleting useless steps and complex expressions on my program. It would make the whole program shorter and clear. To be specific, I am not familiar with Methods enough so that I have to write some long subroutines to represent some simple computation, like I write subroutines for each HEX. Next time, I will be more familiar with using those Methods and write a clearer program.

Part 3:

Brief description:

In this part, the process and logic are pretty similar to parts 2-2. I need to use GIC to represent interrupt while designing the stopwatch. The main body of the GIC code has been provided on the lab manual, in this way, the interruption would send to GIC and branch the pc to the SSERICE_IRQ. I would shorten this part because we have a similar goal and idea with part2-2.

Approach taken:

After reading the provided GIC code, we need to use the CONFIG_INTERRUPT subroutine to configure the interrupt from both the timer and pushbuttons, which could trigger GIC interrupt and let SERVICE_IRQ handle the interrupt meanwhile. In the SERVICE_IRQ subroutine, the decision is made by comparing the interrupt's entrance and its port number. If the interrupt comes from the pushbutton, branch it to KEY_ISR while branching it to ARM_TIM_ISR if coming from the other way. By contrast, the ARM_TIM_ISR branch gets value from the F and stores it into the memory and reset the interrupt. While the KEY_ISR stores the value from the edge-capture register into the memory and clears the register. The main program locates in IDLE. The logic is pretty similar to part2 while I used some Push and Pull methods in part3, and also, I used loop in part3. I tried to use some more methods to shorten my program. There are two memories named PB_int_flag and tim_int_flag to observe if the interrupt happens or not.

Challenges and possible improvement:

The challenge I meet here is to think about how to combine the provided code and the logic I had in part2 together. Also, there is another challenge is to use more efficient methods in part3, such as the push and pull method, the loop method. I tried to use a new method to shorten my program and it did work out compare to part2. While there is a bug I am facing, every time I press button 1 in the Push button to stop the stopwatch, the stop value was not quite right. I think the possible improvement is to fix this problem by myself in the future.