Karl Rouhana 260948648

ECSE 324 Lab 2 Report

Part 1: Basic I/O

1. My approach

   In this part, I had to read the switches and turn ON the appropriate LEDs, I also had to check if the SW9 is has been pressed so I can reset the HEX displays.
   I check if the edge cap register of the Push buttons is not 0. If it isn't, that means we need to write on the HEX displays. Once done, I clear the edge cap so that I don't read the same thing twice.
   Since HEX0-3 has the same address, and HEX4-5 is a different one, I have to figure out where I am when I want to write-clear-flood the HEXes. To do that, I used the value of R0 (passed as an argument to the HEX write, flood, clear function with the index) and AND it with a temp variable, which is multiplied by 2 after every iteration of the loop to find where we should edit. I'm doing that because HEXes are encoded using one-hot encoding. Once we are at the correct index, I edit the bit of the HEX. I rotate the bits of the HEX, to get the bits I want to change as the LSB of the address. The amount I need to rotate is the *index of HEX * 8* since each 8 bit is one HEX display. I AND the bits at the HEX address with 0 at the end of the number to clear those LSBs, and depending on the subroutine, I either keep them cleared, flood them or call a function that finds the number I want to write. After doing one of the above, I rotate the bits back to their original place, and store back the new value of the HEX.
   To find the right bit combination in HEX_write_ASM, I call a function, which will compare R1 (passed as argument with the value to display) with all the number from 0 to 15, and I store the appropriate combination of bits to write them later in the function. I finally use HEX_flood_ASM, to fill HEX4 & HEX5. The call to this subroutine is in the main loop to facilitate turning them back ON after they were turned OFF by SW9.
   To check SW9, I AND the value received from reading the Switches with #0b1000000000, because there's 9 switches and they are one hot encoded, so if the result is not 0, that means SW9 is ON. In that case, I clear all the HEX and continuously loop in Nine_Pressed until SW9 is turned OFF.

2. Challenges faced

   I honestly thought this was going to be the same level of difficulty as lab 1, but I was unfortunately wrong. There was a lot of information coming my way : between Lab manual, the Simulator, the Assembly instructions, and the ARM manual, I quickly felt overwhelmed. So, I took notes of each step to try and find the best approach to this part. I also did not expect to deal with that many areas in the code: The memory, the disassembly, the registers table, LR, the IOs and the addresses of the IO.

3. Possible improvements

   Well first my code is a bit flawed; I sometime must press multiple time the button so it can register that I pressed it and display the correct value. I think this may have to do with the edge cap being cleared more than it should, but I just cannot figure out where or how to fix it. Also, I think my routine to find the index of the HEX to edit is not the best and I'm sure there's a better way to do it. Finally, there could be some error handling done here, for example if a value bigger than 15 is inserted with the buttons.

Part 2: Timers

1. My approach

   In this part, we have to create a stopwatch that has the ability to Stop, Reset and Resume on the push of different buttons.
   I started by configuring the ARM Timer, by giving it a load that will specify what will be the incremental timeframe, which was 2,000,000 for the stopwatch. Since the frequency is 200MHz, the timer will be incremented every10ms for the stopwatch. I also enable the ARM Timer by setting A & E in its Control Register to be 1. I first wait in a loop until the first start button is pressed. Once it is, the stopwatch starts. I store all my stopwatch digits in the memory and update them in a subroutine. In it, I add 1 to the lowest digit (HEX0), until it reaches 10, at which point I increment HEX1 by 1 and bring HEX0 back to 0. I use the same instructions and logics for the remaining HEXes, with the seconds and minutes, and hours.
   Once the stopwatch is updated, I go back to the main loop to check for any Push button pressed. We have three possibilities. One, Stopping the stopwatch (PB1 pressed) will set the E bit of the ARM Timer to 0 to stop the counter. Two, Resuming the stopwatch (PB0 pressed) will re-enable it by resetting E to 1. And three, Resetting the stopwatch (PB2 pressed) will set all digits of the Stopwatch in memory to 0 and write '0' in all the HEX displays. At the beginning, I have to clear the Edge cap of the PBs.

2. Challenges faced

   The biggest challenge I faced was understanding the logic of the ARM Timer and its registers. I also had a lot of problems with clobbered registers, and this took some time to find a fix.

3. Possible improvements

   This part, the code works perfectly. As far as I can tell, the buttons are registered correctly, and I don't have to press them multiple times. Since I'm using the same write HEX method, this part can be improved. Also, my temp variable for the stopwatch are loaded and stored at each iteration of the loop, I know memory accesses are expensive, so this can be something than can be improved by using the same variable registers each time, but I had problems with clobbered registers so I just used the memory method instead.

Part 3: Interrupts

1. My approach

   In this part, we have to create a stopwatch that has the ability to Stop, Reset and Resume on the push of different buttons, but using interrupts.
   First, I needed to update the Interrupt subroutines given. I needed to add the ARM Timer's part. So, in SERVICE_IRQ, I added a check for the if Interrupt ID is 29, value of the ARM Timer's ID. Else, we go into UNEXPECTED. Otherwise, we just call KEY_ISR or ARM_TIM_ISR, to update PB_int_flag & tim_int_flag, since PB_int_flag is updated in KEY_ISR and tim_int_flag is updated in ARM_TIM_ISR. Finally, I also added a couple of lines in CONFIG_GIC to say that we are also looking at the ARM Timer.
   For the application, I start by configuring the ARM Timer (same as previous part) and enter a loop that will wait until the Start button is pressed (PB0).
   Then, I get in IDLE, where I continuously check if tim_int_flag and PB_int_flag are non-zero. If tim_int_flag is 1, it means that the counter has reached 0 and consequently 10ms have passed. We should now increment the stopwatch. The subroutine for the stopwatch are exactly the same, except that they clear PB_int_flag instead of the edge cap.
   KEY_ISR is called when the Edge cap are non-zero and stores in memory the index of button was pressed. ARM_TIM_ISR is called when the counter of the ARM Timer reaches 0 and F becomes 1.

2. Challenges faced

   I copy pasted the code from the manual, and this part did not go so well because it commented code I needed to use, and I did not notice that until later. Also, I really had to read and inform myself with all the information about interrupts and how it works in the different manuals provided.

3. Possible improvements

   Since I'm using the same write HEX method, this part can be improved. Also the variables for the stopwatch are also stored in memory, so this can be improved.