ECSE 324 , Group 17

Mohammad Noor 260782491

Menglin He 260863025

Lab 3 Report

Part 1.1 : Slider switches and LEDs program

- **Slider Switches:**

  Firstly, we created a new assembly file called ***slider_ switches.s*** in which we had to write a subroutine ***read _slider_ switches _ASM*** which read the value at the memory location designated for the slider switches data into the R0 register, and then branch to the link register. We used *.global* assemble directive to make the subroutine visible to other files in the project.

  Next, we created a header file called ***slider_switches.h***. This header file provided the C function declaration for the slider switches assembly driver.

- **LEDS:**

  We created a new assembly file called ***LEDS.s***. Within this file, we wrote two subroutines - ***read_ LEDs_ ASM*** and ***write _LEDs_ ASM*****.** The ***read_ LEDs_ ASM*** subroutine loaded the value at the LEDs memory location into R0 and then branched to **LR.** The ***write _LEDs_ ASM*** subroutine stored the value in R0 at the LEDs memory location, and then branched to **LR**.

- **Main.c:**

  We then created a ***main.c*** file. The file included the header files for both the drivers using import statements. We then wrote a main method that sent the switches state to the LEDs in an infinite while loop. In summary, if a slide switch was pushed up, its corresponding LED would turn on.

Challenges faced

- No challenges faced in this section as it was relatively straight forward to understand and implement.

Possible improvements made/could have made

- No further improvements could have been made as the code was simple and short.

Part 2 : HEX displays

- **HEX displays:**
  As in the previous parts, we created two files ***HEX_displays.s*** and ***HEX_displays.h*****.** We had to write the code to implement three functions – ***HEX_clear_ASM*** , ***HEX_flood_ASM*** , and ***HEX_write_ASM*****.**
  (1) ***HEX_clear_ASM*****:** Turned off all the segments of all the HEX displays passed in the argument
  (2) ***HEX_flood_ASM*****:** Turned on all the segments of all the HEX displays passed in the argument
  (3) ***HEX_write_ASM*****:** Took a second argument ***val***, which is a number between 0-15. Based on this number, the subroutine displayed the corresponding hexadecimal digit (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) on the display(s).
- ***Main.c*****:**
  We just wrote three function calls in the main method. We tested the three functions one by one by adding random arguments that were valid into the function call. The C programming code is very straight forward.

Challenges faced

- The main challenge we faced was how to approach the problem of turning off/on a given set of HEX displays. The concept was easy to understand as it was all about having a 0 or a 1 in the locations of all the segments of each HEX display.

Possible improvements made/could have made

- We think that our code could have been shortened using more loops.

Part 3 : Entire basic I/O program

- **Pushbuttons:**
  We created two files ***pushbuttons.s*** and ***pushbuttons.h*****.** In the ***pushbuttons.s*** file, we had to write the implementation of 5 functions. Of the 5 functions, 2 functions only accessed the pushbutton data register, 2 further functions only accessed the pushbutton interrupt mask register, and lastly 3 functions only accessed the pushbutton edgecapture register.
- **Main.c:**
  We then created an application that used all the drivers created so far. As before, the state of the slider switches was mapped directly to the LEDs. Additionally, the state of the right four sider switches SW3-SW0 was used to set the value of a number from 0-15. This number was displayed on a HEX display when the corresponding pushbutton was pressed. For example, pressing KEY0 resulted in the number being displayed on HEX0, pressing KEY1 resulted in the number being displayed on HEX1. Since there no pushbuttons to correspond to HEX4 and HEX5, we switched on all the segments of these two displays. Lastly, we used slider switch SW9 to clear all the HEX displays.

Challenges faced

- Part 3 built upon Part 2, so the main challenges we faced was in Part 2.

Part 4 : Polling based stopwatch

Here we had to create a simple stopwatch using HPS timers, pushbuttons, and HEX displays. The stopwatch counted in increments of 10 milliseconds which is determined by the working frequency of the board. We used a single HPS timer to count time. Milliseconds were displayed on HEX1-0, seconds on HEX3-2, and minutes on HEX5-4. Pushbuttons PB4, PB3, and PB2 were used to start, stop and reset the stopwatch respectively. Another HPS timer was used and set at a faster timeout value to poll the pushbutton edgecapture register.

A part of the code looks at the input pushbutton values. The code evaluates if any of the push buttons are being pressed and how to react to any of these push buttons being used by checking the default value associated with each pushbutton and the global variable timerstart to control the timer.

The stop watch logic, which was written inside ***main.c***, is easy to understand. Reset the corresponding hex displays when they hit a certain number, i.e., 60 for second and minute. The other chunk of if statement simply check the default value and the global timer, which acts as a Boolean variable to monitor the behavior of the stop watch.

Challenges faced

- Once we understand the whole logic of the polling-based stopwatch, the implementation is relatively easy. However, the assembly code in driver takes time to understand, but with the lab description, we are able to do the implementation.

Possible improvements made/could have made

- The possible improvement is actually part 5 of this lab by implementing the stopwatch with interrupt.

Part 5 : Interrupt based stopwatch

We modified the stopwatch application from the previous section to use interrupts. We enabled interrupts for the HPR timer used to count time for the stopwatch. We also enabled interrupts for the pushbuttons. Furthermore, we determined which key was pressed when a pushbutton interrupt is received. The flags are set in the ISRs.s file, with the example given in the lab description for timer, the flag for pushbutton shares the same logic of implementation.

Here a button being released shoots an interrupt which starts, stops, or resets the stopwatch. The difference between the two timers is that the polling stopwatch acts as soon as the button is pressed, while the interrupt timer acts once a button is released.

Besides the interrupts, part 4 and part 5 share the same idea during the implementation.

Challenges faced

- The challenge in this part was how to determine whether the interrupt is working since there is no obvious differences have been noticed

Difference between Part 4 and Part 5

During the demo, no big difference was noticed between part 4 and part 5, both parts share the same idea during the implementation. However, with the implementation of the interrupt, the program is more efficient. In part 3, the system itself continuously checked whether there is an action going on with every step, while the interrupt handles the continuous part by setting flags for the timer and pushbuttons. The board detects if there is an action going on, then goes to the ***ISRs.s*** file and looks for the corresponding flag value, if the two values (the flag values with the action) match with each other, the timer will react to this particular action.