Laboratory 3: Basic I/O, Timers and Interrupts

ECSE 324- Winter 2020

Lab Section 002 - Group 43
Dafne Culha (260785524)
Majd Antaki (260731626)

Introduction:

This lab introduces the basic I/O capabilities of the DE1-SoC computer which are the slider switches, push-buttons, LEDs and 7-Segment displays. After writing assembly drivers that interface with the I/O components, timers and interrupts are used to demonstrate polling and interrupt based applications written in C.

Part 1: Basic I/O

1.1 - Slider Switches

For this part, the LED display was expected to keep checking the state of the slider switches in an infinite loop. We were expected to be able to read and write to and from the LED displays as well as the slider switches.

To help us get started, code for the slider switches driver has been provided to us. We used that code as a template for writing future driver code. It was very simple to understand as it only required to load the value at the base address of the slider switches on to a register using two LDR instructions. Setting up the header file and the assembly file was fairly simple as well.

1.2 - LEDs

This function was very similar to that of the previous part. Read function was exactly the same as the last part but this time the base address of the LEDS was used. To make the LEDs turn and off when the corresponding switch is toggled, we write to the LEDs what is read from the slider switches in an infinite loop. To read from the slider switches, we load the memory location of them into a register and then load the contents of this memory location into R0.

Putting this part together was pretty straightforward with instructions provided as well.

Part 2: Slightly more advanced: Drivers for HEX displays and pushbuttons

2.1 - HEX displays

As in previous parts, we created two files HEX displays.s and HEX displays.h and placed them in asm and inc folders respectively. We were asked to write the assembly code to implement the three functions listed in the header file.

For clear and flood subroutines, since the displays are passed as one-hot encoded values, we shift the register. Next, we compare it with the input register to figure out which one needs to be flooded or cleared. Clear stores all bits as 0s at the addresses that correspond to the displays. Flood does the opposite and stores 1s instead. For write subroutine, instead of setting it to just 1 or 0, it takes a parameter from 0 to 15 and writes it to the specified displays. The inputs correspond to 16 possible outputs from 0-9, A-F respectively. We decode the input to convert them to the correct HEX character. Other than cycling through every output, write subroutine is very similar to clear and flood subroutines.

2.2 - Pushbuttons

Next, like the last parts, we created two files pushbuttons.s and pushbuttons.h and placed them in the correct folders. We implemented the functions that would access and manipulate specific information about the pushbuttons. This part contains methods to return the outcomes of pushbuttons: returning the value at the edge capture register, setting the edge capture register to 0, and enabling / disabling interrupts. We implement them by assigning appropriate memory locations to specific registers and accessing / writing to the values at those memory locations.

2.3 Integration

Next, we modified the main.c file to create an application that integrates all of the drivers created so far to use them in a useful program and see each subroutine relating to I/O ports works correctly. The program reads the slider switches register and writes their state to the LED register which results in lighting up the LEDs. Furthermore, it also writes to HEX displays when the pushbutton with the same corresponding number as the HEX display is pressed. For example, if the KEY0 is pressed, HEX0 is written. Since there are no push buttons to correspond to HEX4 and HEX5, constantly flood these displays. Finally, asserting the slider switch SW9 should clear all the HEX display. We do this by checking if the binary number it read from the slider switches is greater than 511. If it is, the HEX displays are cleared by using clear.

Part 3: Timers

For this part of the lab, we were asked to create a stopwatch using the HPS timers, pushbuttons, and HEX displays. The stopwatch was expected to count in increments of 10 milliseconds. And we were expected to use a single HPS timer to count time, display milliseconds on HEX1-0, seconds on HEX3-2, and minutes on HEX5-4.

The outlines of the subroutines to be used were provided to us. The configuration subroutine takes a base address for the timers, reads the S bit of each of the timers. Clear subroutine is used to reset the timers. The timers are configured by looping through the timers the same way we did for the HEX displays.

We initialized 2 timers. The first timer increments the count with the rising edge of the clock and the second one polls the first three push buttons to see if they have been activated. If a button is pushed, the edge capture is set to 1 so we can check the edge caps of each button to see if it has been pushed or not. To start the stopwatch, when the 1st push button is pushed, we set the flag variable to 1 and clear edge caps. To stop the stopwatch, when the 2nd push button is pushed, we set the flag variable to 0 and clear edge caps. And to clear the stopwatch, when the 3rd push button is pushed, we set the flag variable to 0, clear edge caps and also clear all the counts.

There is also a while loop that increments the count with the rising edge of the non-polling timer. We check if the int values associated with each hex display should return back to 0 or not after they reach a certain number and write them to their respective displays.

Part 4: Interrupts

Lastly, we had to design a stopwatch which was functionally the same to the one designed in the previous section but we were asked to use interrupts in its implementation.

The interrupts allow us to remove the second timer we needed to use in the previous section whose job was to poll and check if any of the push buttons have been pushed.

Similar to the last part, we still have a while loop that increments the count with the rising edge of the non-polling timer. We check if the int values associated with each hex display should return back to 0 or not after they reach a certain number and write them to their respective displays.

However, unlike the last part, the timer has different values for interrupts activated from different buttons. We have a while loop that checks the status of the interrupt values and responds accordingly. To stop, start and clear the stopwatch, we perform the same operations described in the previous section.

Challenges Faced / Improvements:

The first parts of this lab was fairly simple and easy to understand. For the last part, we were fairly new to interrupts and understanding how they worked and setting flags accordingly to the respective interrupts was very challenging.

One of the biggest improvements we could have fixed was that our board was showing weird segments even though the code was fine and we were able to implement all tasks. We could spend more time on this and fix this issue by checking the hardware or altering the underlying settings.