ECSE 324 - Lab 3

ARCHIT GUPTA (260720757)

KAI LUO (260479742)

Section 1 – Organising

The first section (organising the project) was important for understanding how the structure of this lab will work. Implementing this structure wrongly could mess things up big time. For proper compilation of the drivers, the assembly source files had to be organized properly. It was also easier to move between different files quickly.

Section 2 - Basic I/O

- Slider Switches and the LEDs

This is where the main part of the lab started. This section was not too hard to implement as we were given the header files and the sample C code. We had to read the value of the slider switches from the given memory locations. The value of each slider switch is automatically placed in memory at address 0xFF200040. We had to write a subroutine that reads the data from that address and returns it in register R0.

Next, we had to display the state of each switch on the LED's. We had to use two subroutines. The 1st one read the values of all the switches from the slider memory locations. The 2nd one wrote those values into the LED's memory location. We took inspiration from slider_switches.h in LEDs.h by defining a second function that takes an integer argument. The argument is passed in R0.

We did not face many challenges in this section. We faced a slight difficulty in working with the extern statements and understanding how they collaborate with the main C file to successfully light up the LEDs.

- HEX Displays

To implement this, we had to represent the number on the slider switches (which was essentially in binary) in hexadecimal on the LEDs display. We wrote functions for the 7-segment displays. These are the 3 main functions:

- HEX_clear_ASM function - Turns off all the segments of the specified display(s).
- HEX_flood_ASM function - Turns on all the segments of the specified display(s).
- HEX_write_ASM function - Displays the specified hexadecimal digit on the specified display(s).

Each subroutine creates an 8-bit block of values and then displays the value to the 7-segment display. We use a loop in the subroutines. We first create an 8-bit String of the required values and then taking one bit at a time, we check if the value should be displayed to the 7-segment display. If the display is in the inputList, we clear the appropriate byte with the AND operator and then input the byte into the same memory with the OR operator.

After we reach the end of the loop, we increment the counters.

This section was quite a challenge in this lab. Using the 'AND' and 'OR' operators was slightly tricky but effective at the same time. One inefficiency was that we clear the entire byte irrespective of what operation we are doing.

- Pushbuttons

For this section we wrote subroutines that remember the memory associated with the 4 push keys. We defined 4 functions

- Read functions that return the pushbuttons that are pressed.
- Read edge-capture functions that return the value of the edge-capture memory location.
- Clear edge-capture function that clear the edge-capture memory location to zero.
- Interrupt functions that disable or enable interrupts.

We didn't face much of a challenge because we reused quite a bit of code. According to us, there was not much to improve in this part.

Section 3 - Polling Stopwatch

- Assembly Code

We had to create a basic timer with 3 subroutines (Configure, Read, Clear). The subroutines have an outer loop which look at the timer and a script which identifies the counter's location in memory.

Configure subroutine is the main subroutine. We use a pointer which was placed into a register and clears all the things in front of it. We pushed the registers for the subroutine. After that, we initiated the outer counter. Then we indicated the timer that we would be configuring. We used a compare statement to check the if condition. If evaluated to true, the timer is passed on and we try to find the memory location in which the timer is located. The final part of the configure subroutine was to store all the parameters in the timer memory position.

The Read subroutine required fewer registers to be pushed initially. This subroutine was very similar to the configure subroutine. Just the last part was different as it pulled the correct s-bit from memory to load it into the right-most bit of R0.

The Clear subroutine for timers uses much of the same code from the first two subroutines and then clears the values in the selected timers.

- C Code (main)

Now we talk about the main c code. The thing that runs our program. We configure 3 timers initially. The first timer is the actual 'timer' that is displayed on the display. The second timer checks to see if the pushbuttons have been activated. If the timers are configured and timer start is activated (push button is pressed), then the timer begins to run. We display millisecond, second and minute counters onto the hex displays of the FPGA.

We also look at the values of the pushbuttons through timer 1. We write how the program is supposed to react on the pressing of different pushbuttons.

This program was not too tough to implement as we worked with some friends and together figured things out. The struggle that arose was mainly because of writing the main c file and understanding how to properly call the functions and subroutines which were defined in other files.

Section 4 - Interrupt Stopwatch

This section was very similar to the last section as the only difference was in the nature of the interrupts. Previously a separate timer was used to poll periodically to check for a pushbutton press. But now, pressing the button (and releasing it) starts, stops, or resets the timer. Much of the code was provided already for us to use. We had to write the ISR.s file and of course, the main c file to implement the interrupt based stopwatch.

In the ISR.s file, we reserved memory flags for timer flags and interrupt service routines (for timer selection) were written. In the main C file, most of the structure was directly copied from the previous C file used in Section 3, requiring only the replacement of the timer poller with an if statement that watches for the interrupt flag to be raised.

This part was easier to implement because we already had the implementation of the previous part. To improve efficiency maybe, we could have used the main C file so that only one slider switch can be used to change between the two types of stopwatch (Polling stopwatch and the interrupt stopwatch). This would considerably shorten the main C file.