Karl Doumar 260733160 March 30th, 2018

He Qian Wang 260688073

Group 32

Lab 3: Basic I/O, Timers and Interrupts

**Introduction**

The objective of this lab was to introduce students to I/O interfacing and knowledge on how to implement subroutines to communicate with I/O devices, such as when to accept input from or write output to I/O peripherals. The input and output exist on the Cyclone 5 board with Hex displays, pushbuttons and slider switches. The second part of the lab consists of writing subroutines to be used in the main C program. The program in question is a timer using two methods, polling and interrupts.

**Part 1: Basic Input and Output**

**Part 1.1 Slider switches and LEDs**

Before writing any code, we were tasked with organizing our files into different folders that will be much easier to manage. We created a drivers folder which contains three different sections which are asm, inc and src. The ARM subroutine files will be placed in the asm folder, the C header files will be placed in the inc folder, and any C programs will be placed in the src folder. We will have a main program to run the different subroutines from the folders by making global variables.

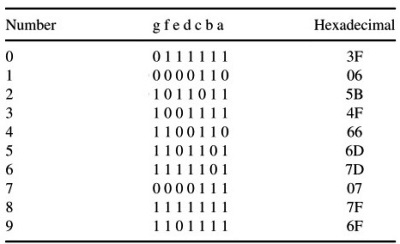
The first part of the lab was to create a subroutine that reads the data from the slider switches registers and if the slider switches are on, we write data to the LED registers to turn on the lights. This part is completed by using a subroutine file called **slider\_switches.s** and a C header file **slider\_switches.h** that will be used in the main program. Everything is placed inside the main method and in an infinite loop that will force our subroutines to be active until we turn off the program. The lights will turn on if the corresponding switches are activated.

**Part 1.2 HEX Display**

The HEX display component of the lab consists of writing a subroutine that writes our input to the hex display registers. Our subroutine must be able to clear a hex display, flood it and write a specific value to the display. The way we wrote the subroutine for the clear display, we know that R0 contains which hex display to clear and it will be given in a hot-one encoding manner. We perform a bitwise AND with R0 and #1 to see which display is concerned. If the AND returns false, we simply shift the R0 by one bit, increment our counter and then we check again.

After getting which hexdisplay to clear, we check whether it belongs in the first or second register. In order to clear the proper byte, we would do a bitwise AND with the data inside the register and #0xFFFFFF00. This will preserve the data and force the register to 0. Afterwards, we write the result into the concerned register.

Similarly for the flood hex display, we would simply do the same steps but instead of doing a bitwise AND, we would use ORR and 0x000000FF. This would be preserve the state of the other data and flood the hexdisplay that we wanted.

The HEX\_write\_ASM would be done similarly, but instead for using the ORR with 0x000000FF, the FF byte would be replaced by the one that corresponds to the character that we want to write. The character is given in the ASCII value and we decided to make about 15 different sections for each value from 0 to F. After figuring out which value, we write the corresponding value basing on a high active method.

We wrote a pushbutton file which is responsible to read pushbutton data, edge cap data and Pushbutton interrupts. When we read any data for pushbuttons, we simply read it from the corresponding registers and return the data in R0. When we need to check if a button is pressed, we simply do an AND to check if the result is 0 or 1. After accessing the edge cap data, we need to move any value in it to reset it to 0.

The hexdisplay is basically just clearing every hex display if switch 9 is active. Otherwise, Hex display 4 and 5 are flooded and then pushbutton 0, 1, 2, 3 are responsible to write hex values on the corresponding displays. The four first slider switches will tell us based on a four bits manner which is the value to display.

**Part 2: Timers**

This part of the lab required us to build a timer using a given HPS timer subroutines, pushbuttons and Hex displays. The timer must be able to display two digits of minutes, two digits of seconds and two digits of sub-seconds. The pushbutton 0 will be used to start the timer, button 1 is to stop the timer and button 2 is to reset the timer.

The HPS timer file is given and it has 3 different subroutines. The first one config takes a structure as an argument and it will be able to access the different data present in it. We configure the timer with values for timeout, LD\_en, INT\_en, enable. The second subroutine will be used to read the value of the current timer and the third subroutine simply clears the current timer.

The rest of the program is written in the main program. We have two different timers, one is to increment our timer for the timer and the second is to poll the pushbuttons. If the timerstart and the timer is on, we increment the timer. If the pushbuttons are active at the polling, we update the timer depending on which one is pressed. For example, we would press button 0 and we start the timer, button 1 would stop it by setting timerstart to 0 and button 2 would reset everything to 0 and write 0 to each display.

**Part 3 Interrupt**

The last part of the lab is build a stopwatch program like the first part, except, we use interrupts to run the program instead of polling. Every function of the program is the same as in part 3, except that we use interrupt flags for the timer and the pushbuttons. In the beginning of the program, we must enable the interrupts for the timer and pushbuttons using ID 73 and 199. Whenever we need to increment the timer, the interrupt flag will become active and we will increment the values in our timer.

The pushbuttons, when they are pressed, they will raise another flag for the program and if button 0 is pressed, we start the timer, if button 1 is pressed, we simply stop the timer and if button 2 is pressed, we would reset the timer. We updated two subroutines in the interrupt service routine files which are ISR.s and ISR.h. The subroutines are HPS\_TIM0\_ISR which clears the timer and raises the flag for our program. Similarly, for the pushbuttons ISR, we read the edge cap and set it in our interrupt flag. We have to clear the edge\_cap again if we want to make the interrupt active again.

This part of the lab has a lot of the logic from the Part 3 timer part of the lab.