ECSE 324

LAB REPORT 3

INTESARUL KHAN – 260579991
SHIRAAZ DILAWAR - 260577486

Abstract:

ECSE 324 Lab 3 is mostly focused around basic functions of I/O, input and output. The DE1-SoC computer was used to run the ARM code that combined switches, pushbuttons, hex displays, timers, and interrupts. Unfortunately, our group was not able to implement the interrupt component of this lab.

PART 1

Slider Switches (slider_switches.s):

This assembly file contains one method *read_slider_switches_ASM*, which reads the value at the memory location for the slider switches. For this part of that lab, the code was provided for us. 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.

LEDs:

This part is similar to the first part. The assembly file contains methods, *read_LEDs_ASM* and *write_LEDs_ASM*. *Read* returns the value at memory for the LEDs. *Write* takes an integer as argument and uses it to light up the LEDs.

Hex Displays:

HEX_clear_ASM clears all displays. We first loop through memory location HEX3_to_HEX0, and store the value 0. We use a register, which is left-shifted every time, to AND with our value. We then repeat the same process for location HEX5_to_HEX4. We had a problem with display HEX2, as it did not always clear properly. We never managed to solve this problem, but it did not play a major role in the execution of the code.

HEX_flood_ASM floods all displays. This is the same logic as the *clear* function, however we store the value 255 instead of 0. This switches on all 7 segment displays.

HEX_write_ASM displays the passed parameter to the 7 segment display. We issue many checks to determine what the value of the passed parameter is. Once we have figured that out, we must store into a register the binary string corresponding to what will light up the 7 segment display. After this is done, we must write this value to memory at locations HEX3_to_HEX0 and HEX4_to_HEX5.

Pushbuttons:

read_PB_data_ASM simply reads the value in memory at the pushbuttons data location. *PB_data_is_pressed_ASM* must first determine what the value of the parameter passed is. Once we find this, we can return the push button pressed. If value = 8, it is the last button, if 4 it is the third button if 2 it is the second and if 1 it is the first.

read_PB_edgecap_ASM simply reads the value in memory at the push-buttons edgecapture location.
PB_edgecap_is_pressed_ASM behaves similarly to *PB_data_is_pressed_ASM.* It compares the argument with the decimal values 8, 4, 2 and 1 and returns the associated button.

PB_clear_edgecp_ASM loads the memory location for edgecapture, and stores 0xFFFFFFFF in its place.
enable_PB_INT_ASM loads memory from the interrupt memory location. If it is 0, we can simply store R0. If not, we must make changes so that we do not affect the other buttons' interrupt status. We perform an ORR operation with the argument and the memory location. The result is then stored.
disable_PB_INT_ASM loads the value at memory. We then take the complement of the argument passed. These two are then ANDed together. This is done so that we do not replace unwanted values. We can then store this result.

Main program:
The main program for this part was fairly simple. It had one main while loop to keep repeating the instructions. We would first light up the LEDs according to which slider switches are on. We then flood displays HEX4 and HEX5. After this, we read if a button has been pressed, and write the value to the corresponding Hex display. We also clear all other displays. The value is calculated by *read_slider_switches_ASM.*

PART 2

Timers:

The stopwatch is implemented by using HPS_TIM.h driver that were provided for this part of the lab. The main program for the stopwatch first initializes the counters for milliseconds(count 0, count 1), seconds(count 2, count 3) and minutes(count 4, count 5) to 0. The buttons to start, stop and reset the watch are also initialized for both 5 ms timeframes. HEX 5 to HEX 0 are all written to 0 at the start. The count happens in a while loop which starts when the program starts. Count 0 starts incrementing until it reaches 10, at which point it resets to 0 and increments count 1 by 1. This controls the milliseconds. Once Count 1 hits 10, it resets to 0 and increments count 2 by 1. Count 2 also resets to 0 when it hits 10 and increments count 3 by 1. Count 3 resets to 0 once it reaches 6 since we now reach count 4 which is in minutes. Count 4 resets at 10 and Count 5 at 6 and together they represent minutes. Once the clock reaches 60 minutes it resets again. HEX_write_ASM is used to write the values of the counters onto the HEX displays. The buttons to start, stop and reset the stopwatch are also configured. These are PB0, PB1 and PB2. stop and start are controlled by the value of the start variable. Reset just sets all the counters to 0 and writes them onto the HEX displays.

PART 3

Due to the many challenges faced in this lab we were unable to develop the interrupt code at all. There is nothing else to say with regards to interrupts.

Problems Faced:

We were unable to work out the lab in time this week for which we failed to demo lab 3. Lack of resources provided for this lab made it extremely difficult to follow through. This lab was fairly complex and attempting to understand them solely from the manual with no professor or TA guidance proved to be extremely difficult.