**ECSE 324 - Lab 3 Report - Group 46**

Jessica Li (260665795)

Claire Liu (260654285)

**Discussion on Approaches to Each Section:**

Slider_switches.s:

Two subroutines are written for this section: read_slider_switches_ASM and read_slider_switches_4_bits_ASM. The first subroutine reads the value at the slider switch base address and returns it, the second subroutine reads the last 4 bits of the value at the base address and returns it.

LEDs.s:

Two subroutines are written for this section: read_LEDs_ASM and write_LEDs_ASM. The first subroutine loads the value at the LED base address into R0 and then branch exit to LR. The second subroutine stores the value at base address in R0 and then branch exit to LR.

HEX_display.s:

HEX_clear: The method takes an input of a HEX variable. The method will compare the input variable and the HEX varying from HEX0 to HEX5. Once the variable is confirmed to be a specific HEX, the method clears the content of the corresponding memory in the FPGA board by assigning 0x00 to the byte. The method will take a decrement counter, as the input may be more than 1.

HEX_Flood: The method is similar to the clear method. Instead of assign 0x00 to the address, we assign 0xFF to the address.

HEX_Write: Write takes 2 variables, the first one is the HEX and the second one is a char indicating the number that is waiting to be displayed. First, we try to identify which number to be displayed and assign corresponding number expression of the 7-segment HEX display. With the on-element of 1 and off-segment of 0. Then, identifying where to display by comparing the HEX address.

Pushbuttons.s:

For the data register, read_PB_data_ASM and PB_data_is_pressed_ASM are written. The read_PB_data_ASM subroutine loads the content of the data register into R0 and returns this value. The PB_data_is_pressed_ASM subroutine extracts the bit corresponding to the input pushbutton, compares it with 0 and 1 and return this bit value.

For the edgecapture register, three subroutines are written: read_PB_edgecap_ASM, PB_edgecap_is_pressed_ASM and PB_clear_edgecap_ASM. The read_PB_edgecap_ASM subroutine loads the content of the edgecapture register. The PB_edgecap_is_pressed_ASM subroutine extracts the corresponding bit in the edgecapture register according to the input pushbutton hot encoding and checks if the bit is 1. It returns 1 if the edgecap bit is on, otherwise it returns 0. The PB_clear_edgecap_ASM subroutine stores the value 0xF to the edgecapture register to clear the register.

For the interrupt mask register, two subroutines are written: enable_PB_INT_ASM, and disable_PB_INT_ASM. Their logic is similar. Basically the corresponding content of the interrupt mask content is extracted based on the hot encoding of the input pushbutton and the modified content is stored back to the interrupt mask register's address.

main.c:

In the main class, some test codes are written for the basic I/O, polling timer and interrupt timer.

For basic I/O, we first turned on all the segments of HEX4 and HEX5 and set the LEDs to be on whenever a corresponding switch is turned on. Inside the while loop, we constructed five if-statements to check the input of the pushbuttons and the slider switches. Then we used the HEX_write method to display the appropriate number or clear all the display segments accordingly.

For the polling timer, we first set all the attributes for the two timers according to the lab instructions. The first timer has increment of 10ms for the HEX display, the second timer has increment of 2ms and that deals with the pushbuttons. We then used division and module operations to calculate the correct number that should be displayed on each HEX_display.

For the interrupt timer, we first the set all the attributes of the timer according to the lab instructions. Only one timer is used in this section and it goes up in the increment of 10ms. We used the fpga_pb_keys_int_flag attribute to determine which pushbutton is pressed. We created a variable called 'active', it would become 1 and the stopwatch starts counting if the start pushbutton is pressed.

**Potential Improvements:**

1. Though our HEX_displays display the correct number, sometimes it would display some random segments on HEX0, we think this is due to the fact that we used STRB and LDRB for

modifying the HEX_display registers. If we had more time, we would use a mask to modify the registers and only use word expressions.

**Challenges Encountered:**

1. One of the biggest problems that we encountered was the writing of HEX_displays. We initially thought that the char argument of the HEX_write method should be inputted as HEX_write (HEX0, '3'), since a char value is usually quoted with single quotation marks. However, we were not getting correct displays with that input and we thought it is something wrong with our code. Therefore, we spent a lot of time (more than 2 days) trying to debug our HEX_display program but could not achieve anything. We then realized that we can simply treat the char argument as an integer and the program finally seemed to work out in the end.