ECSE 324
Computer Organization
Fall 2019

Date: November 11 2019

Lab 3 Report

Group 1

Preyansh Kaushik : 260790402
Elie Elia : 2607959306

Lab 3 introduces the basic I/O capabilities of the DE1-SoC computer - the slider switches, pushbuttons, 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.

1. **Basic I/O**

Slider Switches and LEDs:

In this program, we match the ten LEDs to the ten slider switches below the LEDs. The LED light turns on when the switch is high. We use two assembly subroutines. First is “read_slider_swtiches_ASM” that reads from the hardware address which switches are set to high and returns an int denoting the state of the slider switches. Next is “write_LEDs_ASM” that turns on the LEDs depending on the integer address passed as an argument. In our main function, within a continuous while loop we call the method that writes LEDs with the method that reads the switches as an argument. This effectively continuously maps the LEDs to the switches below then and turns them on when switches are set to high.

Challenges:

This was our first hardware based program so the syntax required to reference specific parts of the hardware took some time to understand.

HEX Displays:

This program starts by flooding the first two hex displays and initializes the remaining four to be cleared. The first four switches (SW0-SW3) are then used to encode a number ranging from 0 to F in hexadecimal using 4 bit binary representation with switch states. By pressing one of the four buttons the encoded binary value from the switches is displayed on the respective 7bit display in hexadecimal. As an example, if the last four switches were set to 1001 and the rightmost (KEY 0) was pressed, the value nine would be displayed on the right-most hex display.

The left-most slider “SW9” is used to clear all four displays. The typedef enum within “HEX_displays.h” maps every HEX display to an integer ranging from 0x00000001 to 0x00000020. Within main there is a continuous while loop that begins by checking if the “clear switch” has been triggered, if not it checks whether a button has been pressed. If a button has been pressed our program takes the binary value of the first four switches (SW0-SW3) and uses the HEX_displays_write_ASM to display the hexadecimal value on the respective display. The write method is different than the previous two because instead of setting it to 1 or 0 it writes a specific pattern corresponding to the input number. The input can be between 0-15, which then has to be decoded to convert those numbers to the correct pattern that displays a HEX character, 0-9,A-F. There are 16 possible inputs corresponding to 16 possible outputs so the program has to cycle through each one to find the correct output.

Challenges:

We had issues with the HEX_displays_write_ASM as we tried to simplify and shorten it but it began complicating the logic of the program and extremely difficult to debug. In the end this file was very long but the logic was understandable and sound.

3. Timers: Simple Stopwatch

The task was to create a stopwatch that uses the HEX displays to output the current time for the watch. The outlines of 3 subroutines, HPS_TIM_config_ASM(), HPS_TIM_read_ASM(), and HPS_TIM_clear_INT_ASM(), were provided to us that interface with the HPS timer drivers that are in the DE1 SoC, this formed the base of the timer system. The configuration subroutine takes a base address for the timers, reads the s bit of each of the timers, and the timers are reset by the clear subroutine. Clear and read also both accept an enumeration that corresponds to the timer being requested. The timers are configured by looping through the timers similar to the loop for the HEX displays. The timer is set up by altering the data stored at its address. To alter that data it first has to be read, the time is loaded, as well as the E, I, and M bits individually. They each have to be shifted so that they can be added together in the form that is specified in the manual. The compiled data is then stored back in the address of the timers. The first timer is used for the stopwatch and the second timer is used as the poller. The first timer has a timeout of 10ms and the second has a timeout of 5ms. The three variables corresponding the the milliseconds, seconds, and minutes are initialized to zero. The buttons are continuously polled to check whether the timer should increment, pause, or reset and clear the displays. The while loop is the section that configures the counting features, resetting milliseconds to 0 when it reaches 1000 (indicating that one second has elapsed) by continuously checking to see whether the counter has reached that value yet.

Challenges:
The timers were a complex and time-consuming subject to comprehend and apply solely from the DE1 manual.

4. Interrupts: Interrupt-Based Stopwatch:

This stopwatch functions identically to the previous stopwatch so all the same button mappings outlined in the previous timer apply to the interrupt-based stopwatch but is implemented in a different fashion. This uses interrupts that are initiated by a button press (initiating a flag) rather than checking continuously for if a button is pressed. We begin by setting up the specific interrupt devices and creating an integer to hold the value of each digit of the hexadecimal display. We then initialize the timer and enable the interrupts for the specified buttons. An indefinite while loop is then used that checks for the flags and uses the timer to update the digits in the display.

Challenges:

Our biggest challenge in the section of the lab was understanding the role of an interrupt in a timer and finding the correct syntax to implement the logic we desired.