LAB 3

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.

1 Basic I/O

1.1 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.

1.2 LEDs

This function was very similar to that of the slider switches, for the Read function it was exactly the same of Slider Switches however this time the base address of the LEDS was used. For the Write function, a STR instruction was used as we were to accept an argument and store it at the base address of LEDs. These two functions were fairly simple and gave no difficulty when writing.

2 Slightly more advanced: Drivers for HEX displays and pushbuttons

2.1 HEX displays

Here it was expected to write 3 functions; HEX_clear_ASM(HEX_t hex), HEX_write_ASM(HEX_t hex) and HEX_flood_ASM(HEX_t hex, char val). Clear is meant to store all bits as 0s at the addresses that correspond to the displays. Flood does the opposite and stores all 1s. Write takes an additional parameter that is the character to write to the specified displays. Initially we started writing the whole code by only storing bytes, which confusingly resulted in weird errors. We then rewrote the code to read the whole word then edit it and then store the whole word again in memory.

The two of the subroutines, clear and flood are basically identical in execution and we wrote the flood routine first. The displays are passed as one-hot encoded values and to decode we shift the register so that we can keep our mask the same. Comparing the mask and the input register that holds the desired displays is enough to figure out which one needs to be flooded or cleared. Two separate subroutines were used to clear/flood the upper and lower sets of displays.

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. Other than choosing the specific pattern to be written with this method, the rest of the write subroutine is very similar to clear and flood. One of the most challenging aspects of this section was determining that the storing bytes did not work as expected. It is not clear what the cuase of the error was but random segments would turn on and sometimes whole 7-segment displays would turn on at the same time as other hex displays. For example, display 5 almost always turned on when display 1 turned on. The other challenge was the debugging features of the IDE did not work at all. Pausing the code after execution resulted in a frozen IDE that had to be force quit, breakpoints caused similar errors. Disassembly was not very helpful without the use of breakpoints.

2.2 Pushbuttons

While the lab document specifies a list of functions to be used for pushbuttons, in practice we only found the need to utilize the read data function that we implemented. The other functions PB_data_is_pressed_ASM(PB_t PB), read_PB_edgecap_ASM(), PB_edgecap_is_pressed_ASM(), PB_clear_edgecap_ASM(), enable_PB_INT_ASM(), disable_PB_INT_ASM(PB_t PB) were not implemented in the interest of time and because they were judged to be unnecessary. The basic form is the same as the corresponding subroutine from the LED sections

2.3 Integration

The next part was to create a program in c that integrated all the existing parts into one basic I/O program. It would highlight that each subroutine was implemented correctly and did not have any extraneous effects. The integration had to maintain that at any time when a switch was flipped that the LED next to the switch was also turned on. This is simple to do by mapping the reading of the sliders to the setting of the LEDs. Then the system has to read the input from the first four switches as binary and write it the HEX display that corresponds with the pushbutton that is pressed. This is implemented by truncating the read data from the switches to the first 4

bits only. Then use the HEX write function and pass the pushbutton that was pressed so that the switch data could be displayed on the correct hex display. The third part is to constantly flood the final two displays, which is again trivial by just calling the HEX flood method and passing those two displays. Then it is necessary to clear all the displays when switch 9 is flipped. Dividing the switch read data by 512 or shifting right by 9, then using an if statement to call HEX_clear on all displays.

The only difficulty faced in this section was making sure that none of the code was blocking the execution. It is not clear what caused this blocking but it was resolved. The rest of the compilation of tasks was straightforward.

3 Timers

Leaving basic I/O behind and focusing now on timers. 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, read 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 the same was we did with 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 to the timers address.

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 count, 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, and the same for the other parts too. It is also where we set the timer to count to the next value and check if it has reached that value yet.

Like the previous parts the difficulties start with the lack of resources provided for this lab. The IDE was again an issue that prevented us from debugging effectively. And pausing was not possible which meant we could not inspect and edit the values in memory and the registers during runtime. The timers are also a complex subject and attempting to understand them from solely the DE1 manual with no professor or TA guidance was extremely difficult.

4 Interrupts

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.