Patrick Ghazal
Kartik Misra

ECSE324
Lab 3 Report

Introduction

This lab's purpose was to showcase the I/O aspect of the DE1-SoC computer, specifically the use of slider switches and push-buttons for input, and LEDs and 7-segment displays for output.
Assembly drivers were designed to interface with those components, then timers and interrupts were used to demonstrate polling- and interrupt-based C applications.

1. Slider switches and LEDs program

This basic program has two key subroutines:

- **read_slider_switches_ASM** : collects the slider switches data
- **write_LEDs_ASM**: writes the data onto the LED display.

The main program bringing them together simply takes the output of the **read** subroutine, and passes it as input to the **write** subroutine, and does so continuously using a while loop. Therefore, whenever a change is detected in the slider switches, the LEDs are updated.

2. Entire basic I/O program

This program implements three other subroutines:

- **HEX_clear_ASM** : clears the entire HEX display
- **HEX_write_ASM** : writes the input data to the HEX display
- **read_PB_data_ASM** : reads the data from the push buttons to know where to display

The main still reads the slider switches data and writes into onto the LEDs, however it additionally checks whether slider switch 9 is on, as this clears the HEX display. To do so, the switches data is bitwise-ANDed to the value 512 (0b1000000000), which effectively discards every bit except the left-most one.
If it is not on, the program further retrieves data from the push-buttons to determine which HEX display should be lit. A similar technique as above is used where the value 15 (0b1111) is used to isolate the four right-most bits of the switches and buttons data.

We have had some difficulty with some residual segments on the display staying lit when they shouldn't, but overall, the code functions as it does and the display is accurate.

3. Polling based stopwatch

For the first stopwatch program, we needed to use the HPS times, the pushbuttons class and the HEW display to show the numbers. The implementation consisted in making the HPS_TIM_config subroutine in order to make sure the correct timer was being loaded into the register, we then disabled the E bit timer, as instructed. The timer was then configured, we had to load and configure the timeout, loading INT_en shifting twice for the I bit, loading enable then storing all of these into the control.

The next step was to build the read and clear subroutines which where written in a very similar manner. As the name implies, the counter's job is to make sure that the correct timer was being used, after which the logic for read and clear was implemented. For the clear subroutine for example, reading the F bit cleared everything and for read, the S-bit was loaded.

We had no issue making the pushbutton class work with our program, the inputs were implemented correctly for start stop and reset, however the milliseconds display wasn't accurate, the timer was going way too fast in general. This is due to an issue where the intervals in which the stopwatch counted was wrong. We had a very hard time trying to pinpoint the problem, changing a few constants names in the main C file didn't help, we tried changing the ms_count variable's value from 100000 to something else but that didn't work either. We did the same with hps_tim.timeout but it too didn't prove successful.

4. Interrupt based stopwatch

This second approach to the stopwatch program required us to use interrupts as opposed to polling. This meant using the HPR timer to "count time" for the stopwatch. We had to implement interrupts for the button input (pushbuttons) so that the system knows an action needs to be performed and respond accordingly (as seen in class). This is all in theory however.

You might notice in our code submission that there is nothing for this section of the lab. The reason for that is due to the complete failure of our program. There was nothing we could do, it just didn't work so we didn't bother leaving the code in. We do however understand the theory behind the interrupt stopwatch's implementation. We had to utilize the c code just like in the previous section but then use INT_ASM() to check the S bit of each timer and check the interrupt flag of each timer. Our attempt to implement that change had failed and we don't fully understand why it did so. We were confused on what purpose the M bit and the EN bit had in the status register of each timer, which might have been the root of the issue.