ECSE 324 – Computer Organization
Lab 3: Basic I/O, Timers and Interrupts

Ryan Servera 260741736
Gabrielle Doucette-Poirier 260738518

Part 1: Basic I/O

Slider switches and LEDs program:

Brief description

This program will let make it so that when you turn on one of the switches, the LED above it will light up.

Approach taken

In this program, the C file contains all the necessary import, which are all the Assembly subroutines required for the switches to communicate with the LEDs. For the LEDs there are subroutines for reading and writing to the address of LED_BASE. As for switches, there is only a read subroutine. When a hardware component is read, it takes the current value of the address and returns it. When you write onto a hardware component, you change the value in the address. For this program, the main file would have this loop which would be constantly writing LEDs with the switches read. Therefore, when a switch was toggled on the LED onto of it would light up.

Challenges

In this part, the challenge came from understanding how to components functioned. In this part it was important to comprehend that all of the LEDs are associate to one bit in a memory address and that if the hardware (LED or switch) was toggle on then it would have the value of 1 and a value of 0 when it is turned off.

Improvements

There are no improvements to be done.

Entire basic I/O program:

Brief description

This program would link all the hardware components (switches, LEDs, Pushbuttons and HEX Displays) together and allow them to communicate. In this program, the HEX display will change depending on the messages sent to it by the other hardware components.

Approach taken

In this program, the C file would contain imports for Assembly subroutines. These subroutines are for all the different hardware components, such as the HEX displays, the sliders and the pushbuttons. In this section, the segments could either be flooded (all the segments are on), or cleared (all segments are off), or written to, given a HEX value to display. To test, the entire I/O, we first flood all the displays, then for cleared two displays, but if a pushbutton was pressed then it would instead display a hex value which was determined base on the sequence of the first four switches.

Challenges

The main challenge in this section was understanding how to HEX displays functioned. A HEX display has seven segments, where each segment represents a bit. However, because the HEX display is part of a set of 8 bits in the address (which contains 32 bits) this detail made it harder, because we needed to be specific when it came to determine which set of 8-bits needed to be change. So first, we checked which displays were set on, then we performed shifts and logical operations to ensure that only desired HEX display was being altered.

Improvements

One possible improvement for our code, would be the use of less loops and use something simpler, because currently using a loop made it unnecessarily complicated when the HEX displays were in different memory addresses. Due to the different memory addresses, the last few displays (HEX_4 and HEX_5) required different branches with different behaviour, to keep in mind which set of 8 bits needed to be changed.

Part 2: Timers

Poling based stopwatch:

Brief description

This program will use the HEX display to emulate a stopwatch, this stopwatch will have the ability to start, stop and reset.

Approach taken

In this program, the main file will contain imports from the HEX Displays, the timers, and the pushbutton. The way the stopwatch was implemented was that is would be set to 10ms and then that will continuously count. In this case, the counts for each display are relative to one another, after every cycle, the count of the next display is incremented. As for the pushbuttons, pressing on them would trigger and event which was being a polled for twice as fast as the stopwatch. Pressing the first button, will toggle a flag that would set the state of the stopwatch to on. Pressing the second button, would set that flag off, turning the stopwatch off. And finally, pressing the third button will reset all of the counts in the stopwatch to 0.

Challenges

One challenge in this section, was using Edge capture for the pushbutton, because in or to use edge capture to set a flag, you need to record the moment at which the pushbutton is triggered. However, every time you recorded it, you need to clear the Edge capture for the pushbuttons. One source of error that we were getting was from how we were clearing the edge capture because unlike other hardware components edge trigger is cleared when something other than 0 is written to it.

Improvements

One possible improvement would use a method other than polling for event triggers (button presses) at a faster rate then the clock, because that still has the ability to not register. To avoid these types of issues it is best to use an alternative method such as poling which is discussed in the following section.

Part 3: Interrupts

Interrupt based stopwatch:

Brief description

This program is like the previous one, it is a stopwatch. However, unlike the previous one, this one uses interrupts to count as well as start, stop and reset the stopwatch.

Approach taken

In this program, the main file will contain imports from the HEX Displays, the timers, and the pushbutton and now interrupts. Also, just like the other one, the stopwatch was implemented with a 10ms timer. However, each time it triggered, the interrupt would set the flag and then allow the counts for the stopwatch to increment. Next for the pushbutton, now they set the flags based on interrupts flags specific to the pushbutton. These interrupts are trigger whenever there is a push button edge capture. With this there is no longer any need to have a second timer to poll the presses. To implement these pushbutton interrupts, we first had to enable the interrupts masks for the pushbuttons and then enable in the main file. In terms of button mapping all of them remain the same. Pressing the first button, will toggle a flag that would set the state of the stopwatch to on. Pressing the second button, would set that flag off, turning the stopwatch off. And finally, pressing the third button will reset all the counts in the stopwatch to 0.

Challenges

The main challenge for this section setting the flags for the pushbuttons. Since we needed to set the flags to change when a pushbutton edge case was triggered and then where the flag had to specific to the pushbutton pressed. One other challenge was that, we didn't realize that the interrupt mask needed to be enabled for the buttons to process interrupts.

Improvements

There are no improvements that come to mind, the interrupts seem like an optimal solution for the stopwatch application