ECSE324 Lab 3 Group 47 – Nihal, Abdallah, Fouad

1 – Basic I/O

Description:

The first part of the lab we had to first ensure that when a slider switch on the board was pushed forward (enabled), the corresponding LED light above the switch would light up for easier use. The goal was to create a program where a user can enter a number in binary using the first 4 slider switches (SW0-SW3), hence numbers from 0-15, to display on the HEX displays. The user is supposed to enter a number, then if a pushbutton is asserted then the corresponding HEX display will show that number; we are only using the first 4 HEX displays as there are only 4 push buttons and the other 2 HEX displays are flooded with all the segments highlighted.

Approach and Challenges:

The first part we broke down was displaying the numbers where numbers above 9 are displayed as hexadecimal values (i.e. 10 displayed as A). To do this we considered the documentation of the board and checked the values of the segments and wrote down the matching encoding for each number for example the number 1 (0000 0001) would have the second and third segments light up so it would be encoded to (0000 0110). We wrote a function to do this in C using a simple switch statement which simplified things rather than doing it in Assembly.

The second part was writing the function for flooding the display. Since it takes in an argument with the one-shot encoding of which HEX display was selected, we had to loop through the bits by creating the number 0000 0001 and left shifting it every loop. The challenge here was that all the HEX displays are not all in the same register, so we chose to loop through the first 4 registers using the same loop, and then skip to another part where we check the values individually since they are just 2 values. We simply store the 8 bit number with all 1s into the memory location of the HEX displays which we also had to retrieve from the De1-SoC Computer document. The same approach was taken for the clear only some parts changed.

To read the slider switches we simply go to the location in memory and load it. Then we clear the rest of the bits so its only the 4 first bits showing and this is what is passed back to our function that converts the number to the encoding to be displayed for the segments.

For the main control of the code, we have a constant while loop that constantly calls a read to the slider switches and writes to the LEDs. It checks to see if the last slider switch is enabled so that it clears all the HEX displays and constantly writes to the HEX displays by reading the push buttons and sending the number created from the slider switches.

Improvements:

The first part we could have simplified was instead of writing two separate parts for the flood method, we could have just changed the pointer to where we want to store using a simple MOVEQ move to test to see if we reached the last two values. This would have cleaned up the code.

An improvement that could have been made is add conditionals to the main C code so that we keep the values of the HEX display and only when we read that the push buttons have been asserted then we take that value, clear the pushbuttons and apply the write, this would have allowed us to save the value onto the HEX displays!

2 – Timers

Description:

The second part is a simple stopwatch timer. The stopwatch counts in 10 milliseconds and uses the HEX displays to show the count as well as 3 pushbuttons to start, stop, and reset the stopwatch. The program is done using two HPS timers where the first timer is for the stopwatch and counts down 10 milliseconds, every time it is done counting the clock is updated. The second timer is used to poll the pushbuttons and we do this at a faster rate so that we capture the pushbutton being pressed.

Approach and Challenges:

The program is set up to loop constantly and check two conditions. The first check is for the push buttons timer, it will read the S bit to check to see if the timer has finished counting down from the 5-millisecond timer, if it has not it will skip it, if it has then it will first clear the timer by reading the end of interrupt register which is programmed to automatically reset the contents of the S and F bit to 0 so that the counter will start again. We then read the data from the PB register (we read the load data rather than the edge of the push button) and check to see if it's the PB3 we turn our Boolean start to true, if its PB2 we turn it to false, and if its PB1 we set it to false and reset the values of our minutes, seconds and milliseconds. This sets us up for the second condition statement that checks to see if the timer for the stopwatch has finished and the start is enabled.

Here we do the same wait to see if the 10-milliseconds have ended and if they have entered the condition, reset the timer by reading the end-of-interrupt register. We then increase the times of the minutes, seconds, and milliseconds depending on their previous status and display the output.

Improvements:

An improvement we could have made is to erase the contents of the pushbuttons (however this would not make a difference), what might make a difference is capturing the edge as this would capture the click faster than the press because it takes half the amount of time.

3 – Interrupts

Description:

The third part of the lab is the same as the second part except we are using interrupts instead of constantly polling to see if the timer has ended. We use only one timer and generate an interrupt when it has finished to increment the stopwatch, and also set up interrupts for the pushbuttons so that when one of them is pressed an ISR is called.

Approach and Challenges:

We do the same as the previous program, except in addition to configuring the first timer we must tell the processer that we want the specific parts of the hardware to generate interrupts which we do with the interrupt setup routine and pass the IDs for the hardware that we picked up from the computer document. We tell the hardware that it will generate interrupts by first telling the timer by setting the interrupt bit to 0, this tells it to generate interrupts. As for the push buttons, we do this by setting all the interrupt mask bits to 1.

Instead of checking to see if the timer has finished each time, the hardware will automatically call the ISR when the timer has finished counting down the 10 milliseconds. When it has done counting it will set a global variable to 1 to indicate to the main program that the timer has finished and we can increment our values. We then reset the value to 0 for the next time the counter finishes counting.

As or the push buttons, the ISR changes another global variable that the main can see called “key_pressed_number”. When a pushbutton is pressed an ISR is called to read the edge registers of the push buttons and clear the interrupt. It will assign the value of which push button is pressed to the global variable and the main decides if its start, stop, or reset and does the same as in part 3.