Lab 3 Report
ECSE 324

Slider Switch

In this part we created .s and .h files and a main file to turn on and off the LED lights on the FPGA with the slider switched. The header file contained the method header read_slider_switches_ASM() and the .s file contained our assembly code which loaded the address of the base register of the slider switches to register R1 and then we load the contents of register R1 into register R0. In LEDs.h we have the method headers to read and write to the LEDs. In the main method, we turn on the LED if the slider switch is 1.

Hex display and pushbutton

For this part we had to display the hex characters on the FPGA 7 segment displays. We were given subroutines to clear, flood and write to the displays. The clear and flood methods were almost identical. One had to clear all of the segments and the other had to turn on all of the segments of the displays.

From the FPGA user manush we found that the address of the HEX displays are divided into two addresses. One containing the address of the first four displays and the other holding the address of the last two. We had to implement a counter that switched pointer locations after checking the first 4 displays.

R0 stores the contents of the slider switch base. We initialize a register counter and initialize another register to 1 for bitwise comparison. We do a logical bitwise AND

to the contents of R0 with R3 (R3 holds 00000001). This is the initial case. To check the other displays we do a logical shift to the left of R3 to get 00000010 and repeat the same procedure. If the AND returns a zero we go to the next locations, if it does not we move zero to that location to make it empty. We implemented flood in a similar way.

For HEX write again we stored the address of the two parts of the hex displays. Similar to clear and flood we keep one register for bitwise comparison and another register as a counter. We compare the inputs using R1 (initially stores the value 0). The we have subroutines to check each number and then compare that to the value of R1. Say if the input is 15 we check all the subroutines one by one until we finally stop at check F. Then we write F in the first display. We load the hex value from a pointer which is offset by the inputted integer value.

Push buttons

For this part we first load the address of the data, edge capture and interrupt mask registers for the read methods. Then we had to poll the conditional flags of the button locations. Subroutines were implemented to manipulate the memory associated with the push keys. The read edge capture function returns the value of edge-capture memory location. The clear_edgecap clears the edge captured in memory location by setting it to 0.

In each of the subroutines we first access the appropriate memory locations and then reading and writing a value. This was relatively easy to implement and there is not much room for any further improvement.

Polling

In this section we implemented a stopwatch which continuously keeps on displaying values. We were also introduced to C structures, data types that allow the grouping of several variables. The variables were accessed using pointers. All of the HEX displays were utilized to display the stopwatch.

The correct units of time like milliseconds, seconds, minutes were incremented after that time has passed. The push-buttons were used to start-time (push-button 1), stop-time (push-button 2), and reset the stopwatch (push-button 3).

The subroutines READ_INT, CLEAR_INT, CONFIG all depend on an outer loop counter which examine which timer we should be looking at.

The CONFIG subroutine serves as a basis for the other two subroutines. It takes a pointer since there are multiple inputs to the subroutine. So another part was included which placed the pointer into a register and cleared everything ahead of it. The other register needed in the subroutine were also pushed here, this is the outer counter. Next there is a compare statement which indicated if we need to configure the timer and also what timer we need to configure. There is a compare statement which indicated if a specific timer needs to be configured. Lastly all of the parameters are stored in the appropriate timer memory position.

The READ_INT used the idea of the first two parts of CONFIG. The finals part of this subroutine gets the correct s-bit from memory and stores it into the rightmost bit of R0. This part checks the conditional bits of the timers from locations in memory.

The CLEAR_INT subroutine clears the values of the selected timers.

In the C code we increment sec after ms reaches 1000, we increment min when second reaches 60 and we reset min to 0 when min is greater or equal to 60.

Then the values are displayed on the display. HEX0 and HEX 1 contained the milliseconds, HEX2 and HEX3 contained seconds and the last two HEX displays displayed the minutes.

Interrupt Stopwatch

This section was similar to the previous section. The only difference being instead of polling a separate timer to check if a button has been pressed, in this section we use interrupt when a button id released, the subsequently starts, stops or resets the timer. The polling timer starts as soon as a button is pressed while interrupt timer acts once a button is released. The code first checks if an interrupt flag has been sent to the program (when the flag is greater than 0). We then determine which button was pressed. In the ISR.s file, spaces in memory were allocated for the flags of each of the timers.