Jesse Chen, Ojan Thonrycroft

Assignment 1

The first part of this assignment was implemented similarly to the sample code that was given for the Timer module. Like the Timer, the x-mas lights we made kept track of its current state out of eighteen total sequential states (six microstates for each of the three macrostates). Each state blinked a sequence of lights on the FPGA board according to the assignment specification. Initially, we simply had it switch states every 0.5 seconds to ensure that basic functionality worked properly. Since our clock speed is 50MHz, this meant that 0.5 seconds was equivalent to 25 million clock cycles. So, in order to get the timing right, we kept a count for the number of clock cycles elapsed since the previous state change, moving to the next state after 25 million clock cycles. Once the final state was reached, the board would loop back to the first state and start over.

To complete the second part of this assignment, we had to look up the Verilog mappings to the physical components on our FPGA board. This was done so we could assign the correct keys to perform their correct functions. Namely, KEY[0] would slow the lights' blinking speed by 0.25 seconds to a maximum of 2 seconds, KEY[1] would speed up the lights' blinking speed by 0.25 seconds to a minimum of 0.25 seconds, and RESET_N would reset the state of the board to its original state – that is, reset both the blinking speed to 0.5 seconds and the lights' blinking state to the first state. As such, we checked for key presses on each clock cycle and assigned their respective functionalities. Finally, we had to use one of the seven-segment displays to display the lights' current blinking speed. The blinking speed was an integer from 1 to 8 that corresponds to blink times ranging from 0.25 seconds to 2 seconds, in 0.25 second increments. This blinking speed was simply incremented or decremented when a key was pressed to either slow down or speed up the blinking, respectively.

The biggest challenge we faced during this project was the implementation of the reset key. While we had the project working with just KEY[0] and KEY[1], adding a third keypress seemed to break everything. We faced several different compilation errors and warnings that were related to attempting to write the same registers within multiple different always blocks using different edge triggers. We had to reorganize a large portion of code to work with a single always block that relied solely on the clock. There were also some issues involving how to multiplex the three different keys. Originally, we had used XOR with KEY[0] and KEY[1] to ensure only one input was received at a time, but once a third input was added, XOR did not behave as anticipated. Instead of checking for exclusively one input, we found it simply checks for an odd number of inputs, meaning it would still return true if all three keys were on. To fix this, we simply used a series of if/else if blocks to check for and address each keypress individually.