William Ma (Partner: Nicholas Petosa)

Professor Hyesoon Kim

Processor Design

15 December 2017

We completed the assignment in Verilog through the use of two always blocks, with one detecting and acting upon key presses, and the other working with one-hot encoded microstates representing the light pattern that is displayed on the FPGA's LEDs. There are 18 such micro states that are transitioned in the order specified in the assignment document. Every ¼ second, the state transitions to the next one. This speed of ¼ second is based off of an adjustable duration N*12500000 cycles, where N is a multiplier, 1 <= N <= 8, which is incremented using KEY0, and decremented using KEY1. The value of N is always displayed on HEX3, so that you can quickly and precisely determine the actual speed of the lights flashing.

The three macro states as described in the assignment document were split up into the 18 microstates, which each can be described as a 10-bit vector, describing the on/off state of each of the ten LEDs, 0 being off and 1 being on:

- State 1: Microstates 0,2,4, five most significant LEDs are on; 1,3,5 all off
- State 2: Microstates 6, 8, 10, five least significant LEDs are on; 7, 9, 11 all off
- State 3: Microstates 12, 14, 16, only most significant LEDs on; 13, 15, 17 least significant

Moving on to the challenges we faced, many of them were the challenges that you would expect with a first hardware project in this class. We first struggled a bit with assignments in our always blocks, as we were a bit unfamiliar with some of Verilog's syntax; even though we've had quite a bit of exposition in class, we hadn't quite had that practical experience to really sort of know exactly what we were doing with the code. In particular, it was a confusion of blocking and non-blocking assignments in the always blocks. However, this was not particularly difficult to sort out. It was only a matter of time before we got it down and other challenges came up.

The other challenge that we faced was identical to the one we once faced when writing input handling code for GameBoy games in our CS 2110 class. Luckily, from a high level, we knew exactly how to solve it, and once again the issue that really held us up was a syntactical one, and a problem-fitting one. Specifically, the problem is that when we press a button, due to the way the hardware polls, if we hold it down, our circuit will seem to receive multiple button presses, when in fact we do not want this behavior. The solution is to have states for whether buttons are pressed or not, and if one is pressed, to ignore successive polls of that button until all input has cleared. This way, we can more precisely control the button inputs, since given the speed of the hardware, not doing this would result in very rapid speed changes even for light presses, as what is a small amount of time on a human scale is extremely long for a circuit.