3DQ5 Lab 1 Exercise Report

Jack Wawrychuk 400145293

Minhaj Shah 400119266

Part 1 – LEDs and Switches

The behavior to control the green LEDs was implemented using combinational logic. An AND gate was used to determine if switches [7:0] were all on and turn on LED 0, and an OR gate was used to determine if at least one switch from [7:0] was on and turn on LED 1. A NOR gate was used to determine if all of switches [15:8] were off, as only all 0s at the input of NOR will give an output of 1 to LED 2. A NAND gate was used to determine if at least one switch from [15:8] was off, as any 0 in the input of a NAND will give an output of 1 to LED 3. An XOR gate was used to control LED 4, as the output will only be 1 if the number of 1s inputted is odd. The gates were chosen as they seem to be the most efficient and simple way to achieve the desired output in each case. A priority encoder was used to determine the least significant switch turned off. Nested if/else statements made sure that the least significant switch number has priority over the others. If a switch was low, the corresponding binary value was stored in a variable, and all other switches were ignored. That value was then stored in LEDs [8:5]. If all the switches are on, a value of 1'b0000 is displayed.

Part 2 – BCD Counter and Push buttons

The input of buttons 0, 1, and 2 are received in a dalways_ff block and control variables stop_count, up_count, and down_count respectively. Due to the fact that these variables can only be driven in one block at a time, the variables s_count, u_count, and d_count were used in the counting block. The corresponding values are set equal to each other at the beginning of each block, e.g. u_count <= up_count, down_count <= d_count etc. The actual counting logic is achieved by first checking if the counter has reached its limit, 59 for counting up and 00 for counting down, and if it is within limit it is increased or decreased according to the counting direction. Once a limit is reached, the variables controlling the counting direction are switched, e.g. if 00 is reached, u_count is set to 1 and d_count is set to 0. Four additional variables, s_count, pause0, pause1, and pause2 are also set to 1 when a limit is reached. These are used to implement the desired functionality of the counting direction not being changed, and counting not resuming, until button 0 is pressed. The values of the buttons and the counting logic will only update if s_count/stop_count are 0, so they will not update until button 0 is pressed. When values change, the value of that variable is not updated for one clock cycle to allow the corresponding value in the other always block to "catch up". If no buffering were to occur, anytime a value was changed the variables in each function would always be opposite, and since they update each other, they would keep switching back and forth. The variables p0,01,02 and pause0, pause1, pause2 are used to determine which always block needs to have its variable value held for a cycle, e.g. if p0 is 1 then stop_count needs to be buffered but not s_count, if pause0 is 1 then s_count needs to be buffered but, not stop_count. This results in the desired logic but also causes there to be a delay of one clock cycle before a variable change has effect, meaning if button 1 is pressed, the direction of counting won't be changed to up until 1 clock cycle after the button is pressed.