ECEN 2350: Digital Logic

Assignment #12

Make a combination lock with your Boolean Board, using pushbuttons.

1. [15 points] Make a reliable Finite State Machine (FSM) in Verilog to take button-press inputs from `btn[3:0]` in the particular sequence `btn[1]`, `btn[2]`, `btn[1]`, `btn[3]` only. While taking input, your state machine should light the `RGB0[2:0]` colored LED to indicate its state:

   In the "No Buttons Pressed" IDLE/initial state the `RGB0` colored LED should light up Red.

   Upon pressing any buttons in the correct sequence, the `RGB0` LED should light up Blue. Once all buttons have been pressed in the correct sequence, arriving in the "Success" state should light the `RGB0` LED Green (unlocked). Thence, pressing any button should reset the FSM to the locked (IDLE) state.

   Pressing a wrong button should return return the state machine to a "No Buttons Pressed" (IDLE) initial state and light the `RGB0` colored LED Red. It should then require the entire sequence to be entered again.

   IF you use the buttons as clocks (i.e. `always @(posedge btn[0])`), you will have to adjust your constraints file to permit `btn[3:0]` to be used as clocks. Consult the instructions in Lab 9, e.g.

   ```
   set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets btn[0]];
   ```

   If you use the 100MHz system clock (`mclk, clk` or a derivative), you may build a sub-FSM to handle button-press events to avoid the interpretation of a button being held down as "button press" every 10ns. An actual "button-press" would consist of comparing the *current* button state {pressed or not pressed} to its *prior* state (saved in a `reg [3:0] btnOld;`). In Verilog, since the buttons are a `[3:0]` vector you can do this in parallel:

   ```
   pressed = btn & ~btnOld;
   ```

   Tips to make your state machine more stable & reliable:
   - Use a Moore machine. Make your outputs a combinational function of your state.
   - Use a Verilog `case` statement to determine the next-state from the current state.
   - Use a synchronizer, as shown in lecture, when you have unpredictable (e.g. Human) inputs. This avoids instabilities caused by setup/hold time violations.

   Submit your solution as the file `lab12-q1.txt`

2. [10 points] Alter the state machine to process the button sequence { 2, 3, 2, 0, 1, 0 }. Add the use of the `RGB1[2:0]` colored LED to light up to indicate the lock's status: Red for locked, Green for unlocked. A correct button sequence input should toggle the locked/unlocked status. This behavior is similar to many push-button safes.