Design Assignment #3 v20190521

(Deadline: 11:59PM PDT, Monday, June 03, 2019)

Name (Last, First):

Student Id #:

INSTRUCTIONS

NOTE: This design assignment is to be done individually using EDA Playground. https://www.edaplayground.com. You should have already gone through the startup guide.

SUBMISSION PROCEDURE: You need to submit your solution online via Gradescope. https://www.gradescope.com/courses/4278 (code: MDZ4EM). The website will automatically grade your submission.

DESIGN PROCEDURE: Note that for each Verilog design task a dassignX_X.v and optionally dassignX_X.tb are provided. The .v file contains the module you are to design. You are to fill in these modules for each design task.

The XYZ_tb.v files are test benches that tests your design. The test benches, led_fsm_tb.v, code_reg_tb.v, and dassign3_tb.v will be very similar to the code used for the autograder. Your code needs to work with them without any modification to the test benches. You are of course welcome to create your own test benches to do your own testing.

Note that these files have module names and interface signal names already defined. These MUST NOT be changed (deleted from or added to) in any way - doing so will in all likelihood result in a design that will not comply with the grading program and thus result in a zero score.

Only submit the files provided on gradescope. Do not change the filenames since the autograder is looking for specific filenames.

File provided: *dassign3.v, dassign3.tb.v, moves.txt.*

In this design assignment, you are tasked to design a TicTacToe game for two players. The 3x3 game board is shown below where $p_{8\text{-}0}$ corresponds to a square.

p_8	p_7	p_6
p_5	p_4	p_3
p_2	p_1	p_0

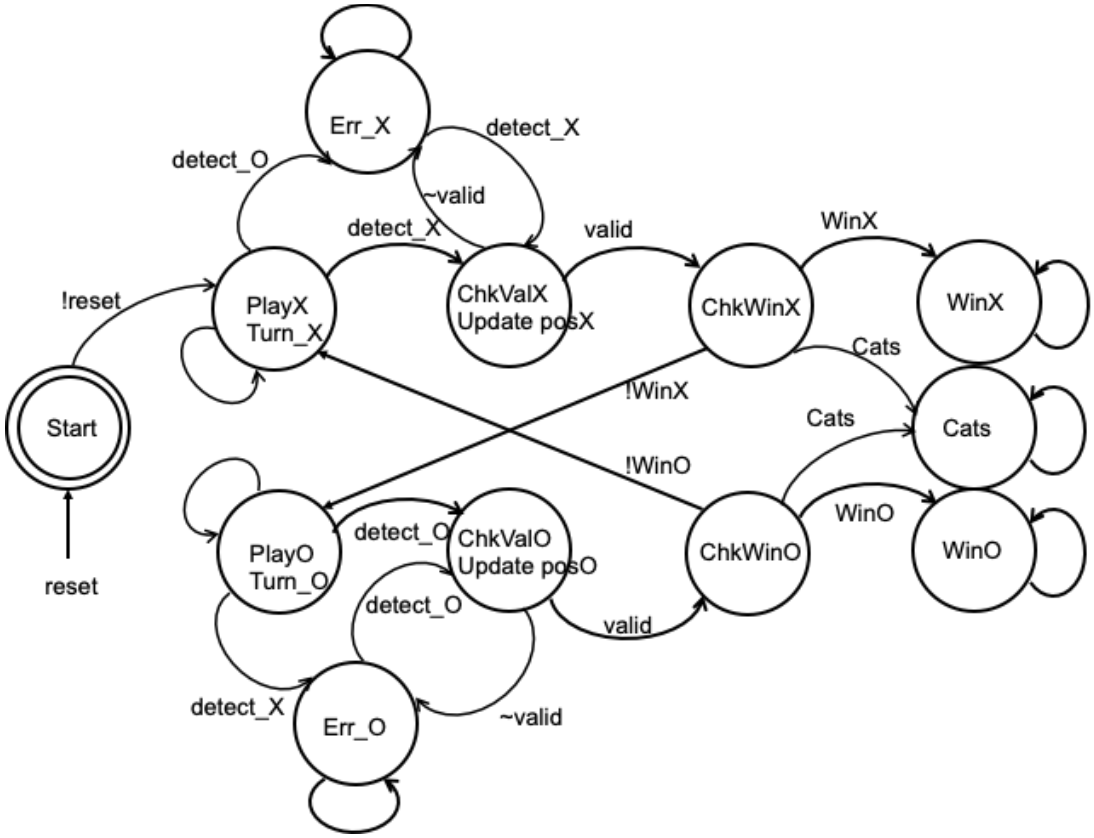
The game takes the following inputs:

- A *reset* signal starts the game. As long as the *reset* = 1'b1, the game is in reset.
- A *clk* signal for running the game state machine.
- A *flash_clk* signal to flash a signal as explained below.
- 9-bit signal, *sel_pos[8:0]*, to indicate which square is selected as the move. This signal is 1-hot.
- 2 signals (buttons) to indicate a player making the move. Signal, *buttonX*, indicates player X placing the X in the square indicated by *sel_pos*[8:0]. Signal, *buttonO*, indicates the similar move by player O. Only 1 button can be pushed at any given time. You can assume that either signals are only asserted for 1 clock cycle.

The game module outputs the following signals:

- 2 signals to indicate the player's turn, one for X's turn, *turnX*, and one for O's turn, *turnO*.
- Two 9-bit signals, *occ_square[8:0]* and *occ_player[8:0]*, to indicate whether a square is occupied and which player is occupying the square. For an occupied square, a 1'b1 in the corresponding *occ_player* is marked by 1'b1 to indicate an X, otherwise it remains a 1'b0. So, each square is essentially represented by 2 bits, a bit to indicate whether it is occupied, and a bit to indicate which player (if it were occupied and otherwise it is 1'b0).
- An additional 9-bit signal to represent the playing board, *occ_pos[8:0].* This signal combines the information of the 2 9-bit outputs above and indicates when a winning position is detected. Each *occ_pos* signal is 1'b0 to indicate unoccupied, 1'b1 to indicate occupied by X, and flashing 1'b1 at a rate of ½ *flash_clk* to indicate occupied by O. If a winning position is detected, the winning row/column/diagonal flashes at a rate of *flash_clk*.
- 8-bit signal, *game_st[7:0]* (an ASCII character), to indicate the status of the game. The character "X" to indicate the winner is X's player, "O" to indicate the winner is O's player, "C" to indicate a tie ("Cats-Game"), and "E" to indicate an error detected. An "n" is indicated when none of those conditions are present. An error, "E", is indicated if a player tries to place and X or O on an occupied square or if an X or O is being played during the other player's turn.

To help you get started, the game state diagram is provided below. The figure is intended to show the basic states and does not show all possible logic conditions for the transition arcs. You can choose to implement this another way if you choose. You should use modules for (1) a state machine that determines behavior of *occ_square*, *occ_player*, and *game_st signals*, (2) logic that checks a move for validity of a move for an Error, and (3) logic that checks a Win or Cats-Game.



EE89 Design Task

Adapt this design so that it can be embedded in a Xilinx FPGA. You should use toggle switches [8:0] to set the sel_pos[8:0]. To test your logic, start by using LEDs led[8:0] indicate the position. The led[15] indicates X's turn, and led[14] indicates O's turn. The middle button is the reset. Up button indicates X-move and down-button indicates O-move. You will need to synthesize the design using Vivado (Xilinx FPGA Design Environment) and compile the design for the Basys3 board.

Several additional tasks are required for EE89. You need to account for some of the realities of working with the Basys board.

(1) A button-press can easily last for longer than a clock cycle. Holding down signal should not continuously indicate such a move is being made. To avoid a problem with this, a debouncing function should be implemented to detect a 0-to-1 transition of either signal.

(2) Similarly, it is possible that a player may incorrectly set multiple bits of the toggle switch for *sel_pos[8:0]* leading to multiple bits to be asserted instead of being one-hot. Under such a condition, either an error, “E”, should be indicated in the *game_st[7:0]* or such signals should be ignored when the player tries to play the move.

(3) The ASCII output that indicates the state of the game should drive the 7-segment display.

(4) Lastly, an 6x6 LED array will be provided with 7 inputs to display the board. Instead of using 9 LEDs to indicate the position, you should replace the module to use the LED array. The array is “charlieplexed” as we designed in Design Assignment #2. So, you should design the logic to set the LED array to have the following behavior. Every 2x2 portion of the array corresponds to a square. Of each 2x2 array, the top 2 lighting up indicates X, bottom 2 lighting up indicates an O. Recall from design assignment #2, in order to light up the entire array properly, the logic you design will need to cycle through each of the LEDs quickly with the right setting. You need to indicate a “Win” with a slow flashing of the light of the corresponding row/column/diagonal.