CECS 361 Spring 2021 Final Project

Tic-Tac-Toe

By

Sabrina Hsia & Jackie Huang

Our design uses the clock from the Nexys A7-100T to drive all of the sequential circuits we will be using to implement the tic-tac-toe game. There will be a total of about eight modules; Tic_Tac_toe, position_tracker, position_decoder, winner_detector, illegal_move_detect, noSpace_detect, and fsm_turns. Each of these modules will be handling different aspects of the game.

The top module called *Tic-Tac-Toe* will be used to connect all of the other modules together into one concise module that will simulate a tic-tac-toe game.

The *position_tracker* module is used to keep track of the nine positions on the 3x3 grid that have a mark placed on them already and prevents the placement of a new marker there if one already exists. There is a reset input that will load all nine positions with their default values and clear any previous values. By default, all values are loaded with the 2-bit value of 2'b00. Each position holds a 2-bit value that is used to differentiate between an empty space and either player 1 or player 2 mark on each of the positions. Player 1 has the 2-bit value of 2'b01, while Player 2 has the 2-bit value of 2'b10. Each of the players have a 9-bit enable single that is used to determine which position they want to place their mark. With a 1 in the n-th position to determine which position the players want to place their mark. However, we also have to create a set of outputs that have inverted logic. This is because the RGB LEDs turn on with a logic 0 and turn off with a logic 1. In this case, we want all of the LEDs to initially be off which is why we have a second set of outputs with the initial values of the position set to 2'b11.

The module *position_dec* will be used to decode a 4-bit decimal number input from the users into a 9-bit binary number that will be used to determine which of 9 positions on the board the user wants to mark. This module makes it easier to choose the location where the player wants to place their marker on the grid. Instead of having to memorize which 9-bit position correlates with the corresponding position on the grid, the player can just input 0-8 to choose from the 9 locations on the 3x3 grid.

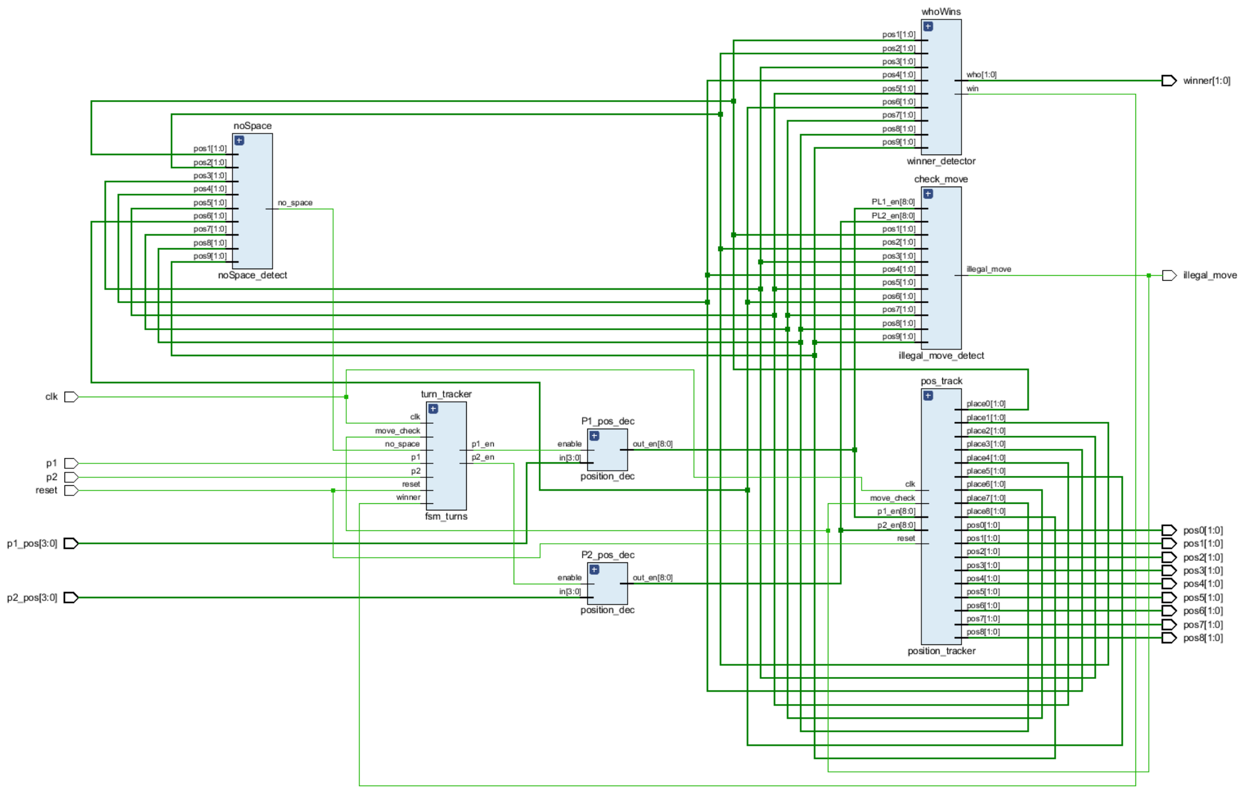
The *illegal_move_detect module* is used to check if the position that the player wants to place the mark in is a valid position or not. To do this it takes a look at all nine positions and at the enable signals of the players in those corresponding positions to see if the position already has a marker from either of the players. If the module does detect an invalid input from a player it will output a 1 preventing any that move from happening. Otherwise, it will output a 0 and allow the move to happen.

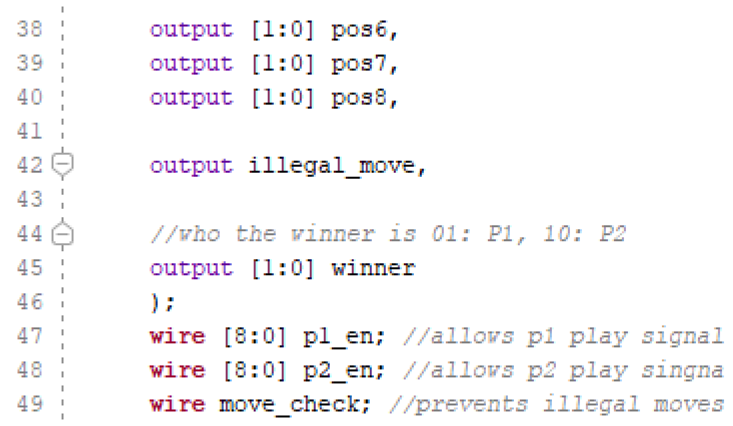
The *noSpace_detect* module is used to detect if there are no open spaces left on the board. If no winner is reached before this module outputs a 1 it means that the game ends in a tie since there are no more positions left to play.

The *winner_detect* module is used to compare 3 positions of the board with each other and outputs a 1 if all 3 positions have matching markers. This module only compares 3 positions with each other; it doesn't check for win conditions.

The *winner_detector* module is used to tell who the winner of the game is. It checks all eight ways that someone can win the game to see if any of them contain only the marker from one of the players. If none of the eight ways contain only the markers from a single player the module will not output a 0 which means that no one won the game.

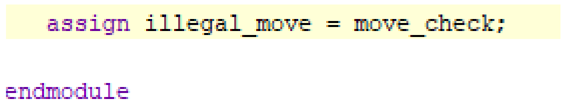
The *fsm_turns* module is an FSM machine that will handle determining whose turn it is and prevent someone from going more than once per turn. At the same time, the FSM will prevent any illegal moves from happening and allow the player to try another move until they make a valid one. The last job of the FSM is to determine when someone has won the game and to then prevent any more moves from happening since the game is already over. If the game reaches the point of where there are no spaces left and there is no winner the FSM will then also end the game and prevent anymore changes of the positions on the board until the game is reset.



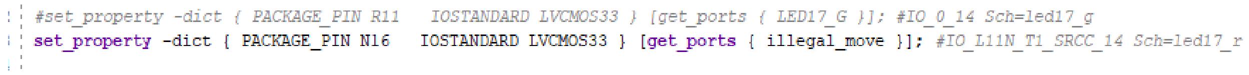


```
    output [1:0] pos6,
    output [1:0] pos7,
    output [1:0] pos8,

    output illegal_move,

    //who the winner is 01: P1, 10: P2
    output [1:0] winner
    );
    wire [8:0] p1_en; //allows p1 play signal
    wire [8:0] p2_en; //allows p2 play singna
    wire move_check; //prevents illegal moves
```

```
                        .p1_en(p1_play
```



```
  assign illegal_move = move_check;

endmodule
```



```
#set_property -dict { PACKAGE_PIN R11   IOSTANDARD LVCMOS33 } [get_ports { LED17_G }]; #IO_0_14 Sch=led17_g
set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { illegal_move }]; #IO_L11N_T1_SRCC_14 Sch=led17_r
```