Foundations of Computer Systems Design Lab (CS2310)
Lab 9: Counters and Tic-Tac-Toe

29 October 2024

In this lab, we will learn how to implement *D flip flops*, *counters* and a module to implement *Tic-Tac-Toe* game in Verilog.

Submission guidelines

- Use the same module name and port definitions as given in the question. Use the same filenames as given in the testbenches.
- Testbench files for all questions have been provided. Ensure that your Verilog code passes all testcases.

1 Designing a D flip flop **(20m)**

In this question you will implement a D Flip Flop in two different ways.

1. Implement a negative-edge triggered D flip flop in Verilog, in a Master-Slave architecture using two D Latches. Use the module signature `module D_FF_MS (input D, input CLK, input RESET, output Q)`. (10m)
2. Implement a D flip flop in Verilog, using the edge detector and one D Latch. Use the module signature `module D_FF_ED (input D, input CLK, input RESET, output Q)`. (10m)

Note that you must create and use D Latch module(s) and you are only allowed to use behavioral modeling inside the D Latch module. The reset should be Asynchronous.

2 Designing a ripple counter **(30m)**

Design a 4-bit ripple counter using D flip-flops with structural modelling. The module signature should be `module RIPPLE_COUNTER (input CLK, input RESET, output [3:0] COUNT)`. When the reset line is set to '1', reset the counter value to 0 asynchronously.

3 Designing a Tic-Tac-Toe module **(50m)**

In this question, we will design a Verilog module that implements a module to simulate a Tic-Tac-Toe game, maintaining the game state after each move by players and checking if any player has won the game. Note that we are not implementing a module to play against us, but simply to simulate the game between two human players.

To simplify the construction of this module, we have broken down the implementation into two steps,

1. Design a 'TCell' module that models a single cell of the 3×3 board.
2. Use the 'TCell' module to create a module that updates the cells based on user input.

Design a TCell **(10m)**

Design a synchronous module that takes a clock signal, set and reset lines and a symbol to store in the cell. The output of the cell consists of (i) a *valid* line that is set to 1 when the cell is filled and is 0 when the cell is empty (ii) a *symbol* line that indicates what symbol is filled in the cell. *Symbol* **value 1 indicates X and 0 indicates O**. Note that we don't care what the symbol value is when the valid line is 0.
The module signature should be `module TCell(input clk, set, reset, set_symbol, output reg valid, symbol);`

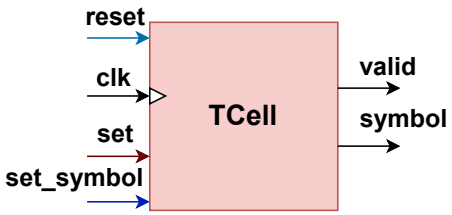

Figure 1: TCell module

- The cell should be empty initially.
- The cell should update it's value (triggered) on the positive edge of the clock signal.
- When the *reset* line is 1 at trigger, the cell should become empty (Synchronous reset).
- When the *set* line is 1 at trigger, the cell should store the value of *set_symbol* as it's new symbol. The *valid* line should indicate that the cell is filled.
- *reset* line takes precedence over *set*, i.e. when both *reset* and *set* are set to 1, the cell should reset.
- Once a symbol has been set in the cell, it should not be overwritten by another set operation in the next cycle. It should preserve the old symbol value until the cell is reset.

Design a Tic-Tac-Toe module (Take home assignment. Deadline: 3 Nov) (40m)

Use the TCell module designed in the last subsection, to create a module that implements Tic-Tac-Toe game. The module takes a 2-bit row and column number as input (**Valid Row and Col starts from 1**), along with clock, set and reset line. The output consists of valid and symbol lines corresponding to all 9 cells along with a two bit game_state (whose meaning is given in the table below).

game_state value	Meaning
2b'00	Game started
2b'01	X won
2b'10	O won
2b'11	Draw

The module signature should be `module TBox(input clk, set, reset, input [1:0] row, input [1:0] col, output [8:0] valid, output [8:0] symbol, output [1:0] game_state);`

- The board should be empty initially.
- The first player should always get the symbol X.
- The board should not be updated after a player wins or it is a draw, until the board is reset.
- The game_state indicates a draw only when all cells are filled and there is no winner.
- Note that it is possible to design the module without any sequential elements (except the TCell modules).
- To help you get started, we have provided a high level diagram of the TBox module in Figure 3.
- You can use the row_col_decoder module implemented in Lab 3 for implementing TBox.
- We have provided only few testcases in the testbench. You are expected to create more testcases and verify that all corner cases have been handled. The testbench contains helper macros to play a move, print the state of the board, etc.

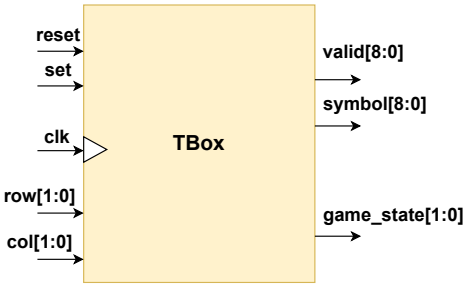

Figure 2: TBox module

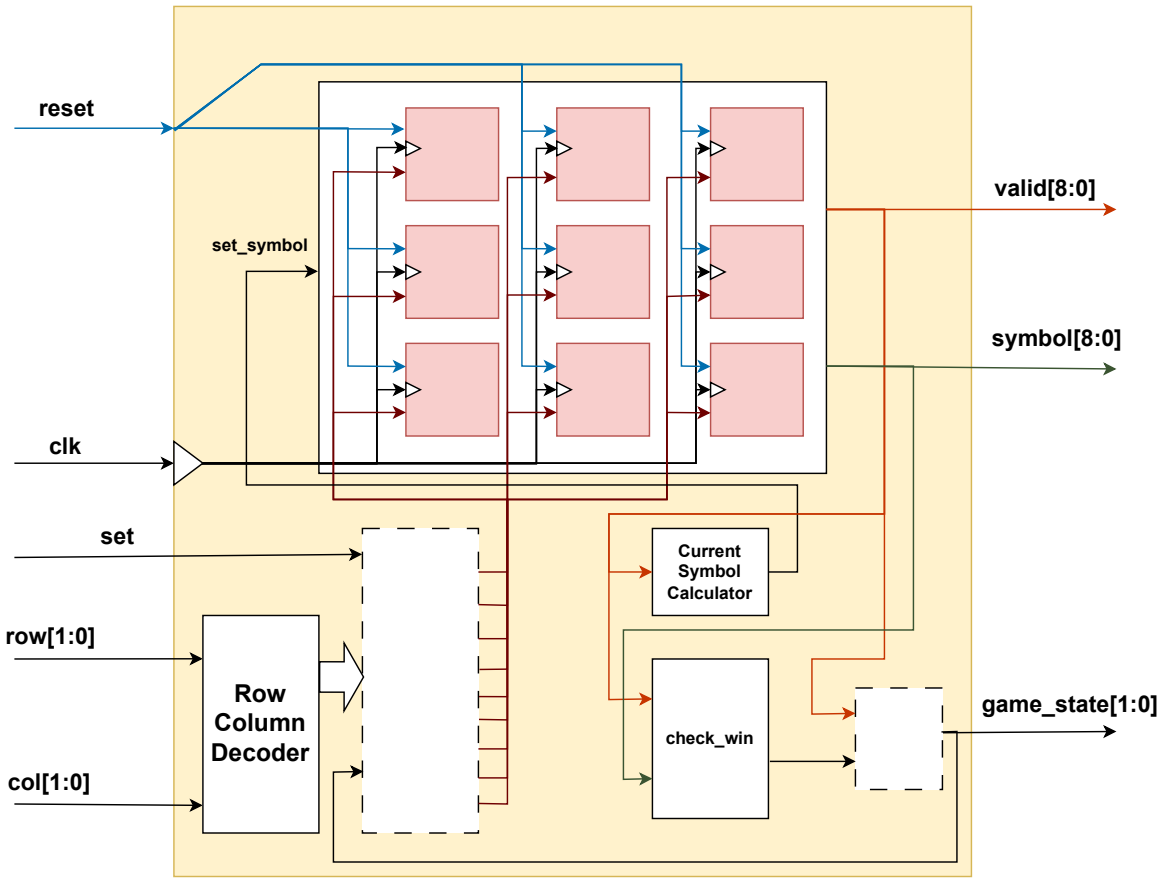

Figure 3: TBox module outline