Lab 8 User Manual

Sathvik Kanuri

CSE 369: Intro to Digital Design

December 2 2024

Introduction

The following manual provides an overview of how to play the tic-tac-toe game on the DE1 board with an LED display. It also provides a breakdown of each module and and submodule used in the design, with the simulation results.

How to Use the Design

Starting the game will show the game board on the LED display. Player X starts first. The current square that the player is hovering over is indicated by their symbol flashing in green. Pressing Key 3 acts as a next button, moving the cursor to the next square. Pressing Key 2 serves as a select option, locking in the users symbol into that square.

After the game ends, the result is displayed on the HEX display as either X wins, O wins, or a draw. Once a player wins, the game is locked. Pressing Key 0 acts as a reset button when needed, resetting the whole board.

I added two additional features. The first is a timer. The timer is indicated by the LEDs which slowly decrement. When time is out, all the LEDs are off and the users input is locked. This means they cannot select a different square, and must select the currently hovered square. Pressing select will select the square, change the player, and reset the timer. The second feature is to flash the winning line of symbols at the end of the game.

System Block Diagram

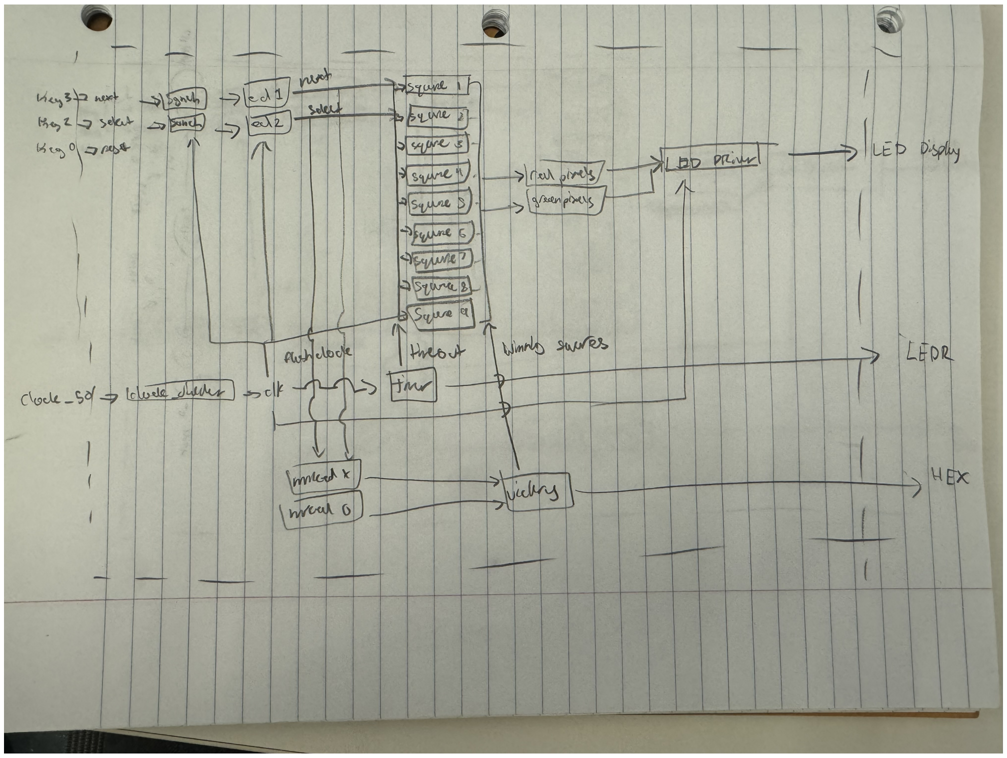

Figure 1: System Block Diagram

The users input is given in by Keys 3, 2, and 0 which control next, select, and reset respectively. The input is fed into a synchronizer and then an edge detector to ensure it only is high for 1 cycle. These inputs are given to 9 square modules which each control one 4x4 grid of LEDs on the board. The top level module sees over tacking which square module is currently active, and which squares have been marked. When select is pressed, the squares state changes, and the top level module moves to the next player. The victory module take in input from the top level module in regards to what squares have been marked, and checks the 8 win conditions for both players and outputs to the HEX display. Finally, the timer takes in input from a slower clock and outputs the time left on the LEDR field, and outputs timeOut when time has run out. When timeOut is high, the users input is locked until they reset the game or press select. The CLOCK_50 input is fed through a clock divider module and given to each appropriate module. Reset resets the whole game.

Module Descriptions

For each module in your design, describe its purpose and functionality. Include state diagrams for any modules with finite state machines (FSMs).

Module 1: DE1_SoC Module

- **Purpose:** The top-level module that manages the entire game.
- **Functionality:** The module takes in input from the user and runs it through an edge detector. It instantiates 9 Square modules for the grid and passes input to them, as well as other values such as which square is currently active, and which squares are marked by which player. It also creates a timer which locks user input after it runs out. It creates a victory module, which checks the 8 win conditions for both players, and the draw condition which is outputted to the HEX display. It routes all the output to the red pixels and green pixels array which is given to the LEDDriver to display.

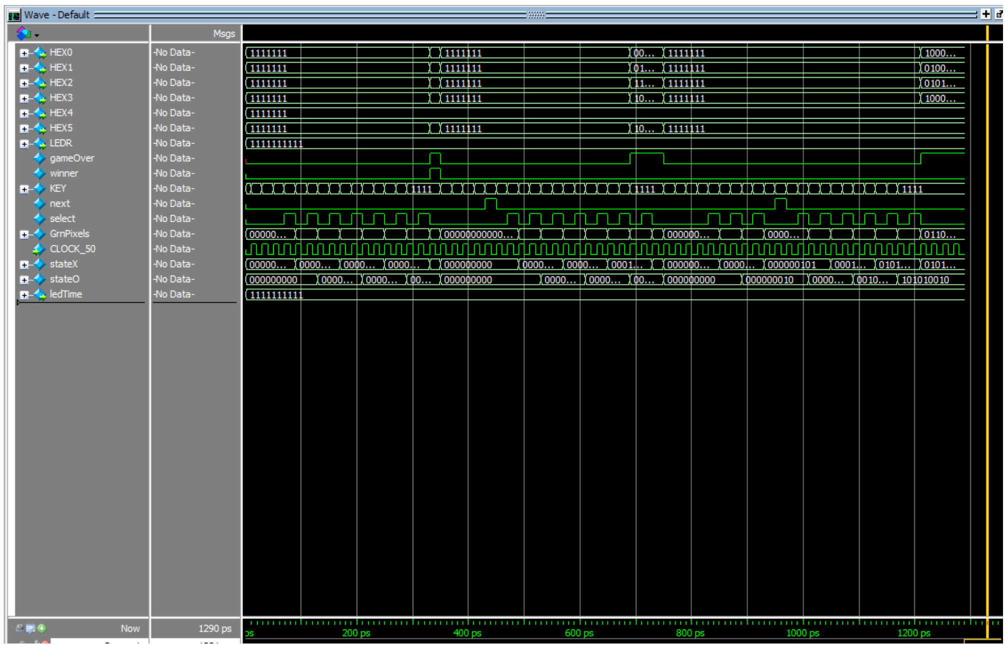

Figure 2: Simulation Result for Top Level DE1_SoC Module

The simulation goes through all three win conditions for the player, with the resets. The first showcases player X winning. After pressing select 7 times, a diagonal line is formed and the player X wins. This is indicated by the gameOver and winner signals being high. After this, reset is hit.

The second condition is when player O wins. Here, we first hit next, and then select 7 times. This results in a diagonal line for o. The game ends once again with gameOver signal being high, and winner being low indicated player O. Reset is hit once again after this.

Finally, the draw condition. The buttons are pressed in a manner that fills the board without a win and the gameOver signal goes high.

For each of these conditions, the HEX displays have different text that say either ”X wins”, ”O wins”, or ”Draw”.

Module 2: Square

- **Purpose:** This module represents a 4x4 LED square in the tic-tac-toe grid. With 9 of these, we are able to create a tic-tac-toe board. The module takes in input for what it should display, and outputs it for the display.
- **Functionality:** The module takes in the input for the select button, as well as an integer to determine if the current square is active (meaning it is being hovered over). If the square is active but not yet selected, it uses a slower clock to flash the current symbol as an indicator of the current selection. Once it is selected, it changes its state according to which player is currently making a move, and outputs the symbol to be shown on the board in a 16 bit output logic. It also outputs which player has currently selected the square, to be used by the top-level module when determining win conditions.

  On a reset, it sets its state back to empty, and resets any values indicating its selection.

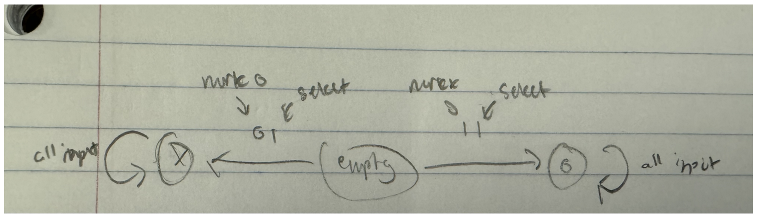

Figure 3: State Diagram for Square

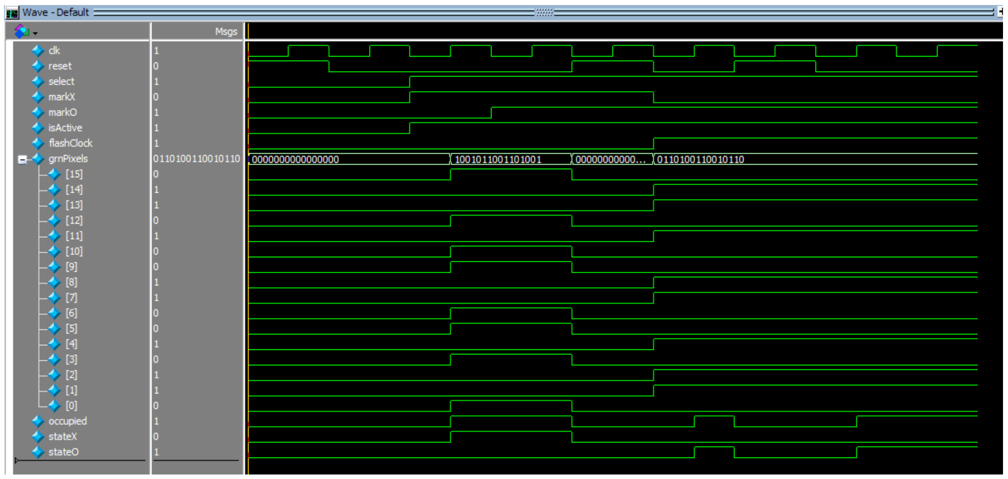

Figure 4: Simulation Result for Square

The above module shows the simulation for the square module. When the square becomes active, and is to be marked by X, pressing select sets its state to X, and sets the grnPixels to be the symbol for x. Resetting the square sets it back to empty. Then, when player O is to mark the square, pressing select will set the squares output to be O.

Module 3: Victory

- **Purpose:** This module checks for win/draw conditions, and creates an output for the HEX display.

- **Functionality:** The xMarks and oMarks logic is managed in the top module and as given as input into this module. The module checks for any of the 8 win conditions for both players, and outputs the appropriate text (”O wins” or ”X wins”) to the display. If the board is full and there is no winner, the game outputs draw. On any of these three conditions, the module also outputs a game over logic, that stops suer input until the game is reset.

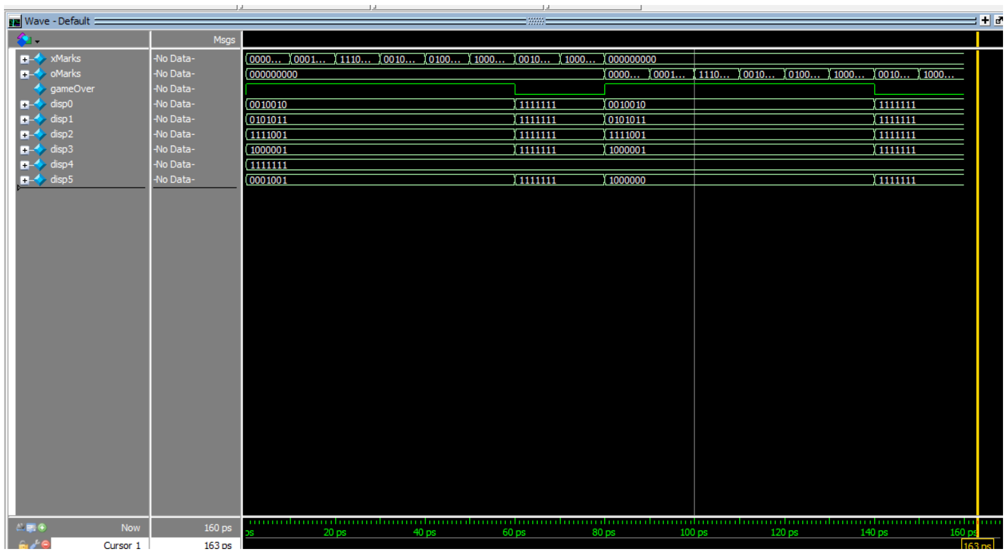

Figure 5: Simulation Result for Victory

The simulation shows all conditions for the victory module. It starts with o marks empty and runs through the 8 win conditions for X, and then repeats the same for player O. It then checks the draw condition. The output can be seen in the game over field, and the HEX displays.

Module 4: Edge Detector

- **Purpose:** This module is used to make sure input is only processed once per button click.
- **Functionality:** The module takes in the input from the button, and runs it through a simple 3-state finite state machine to determine whether to output 1 or 0. If the state is not selected, it will move to the selected state for one cycle outputting one. For the next clock cycle, if select is active it will go to a second state and loop there until it is unselected. The third state is the default state that is navigated to when the select input is 0.

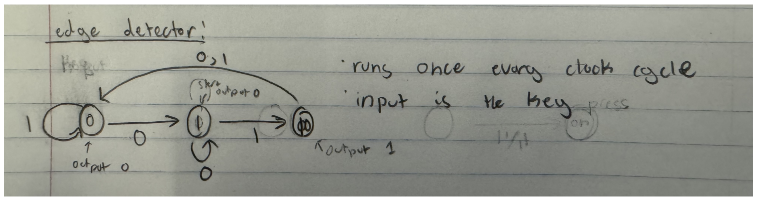

Figure 6: State Diagram for Edge Detector

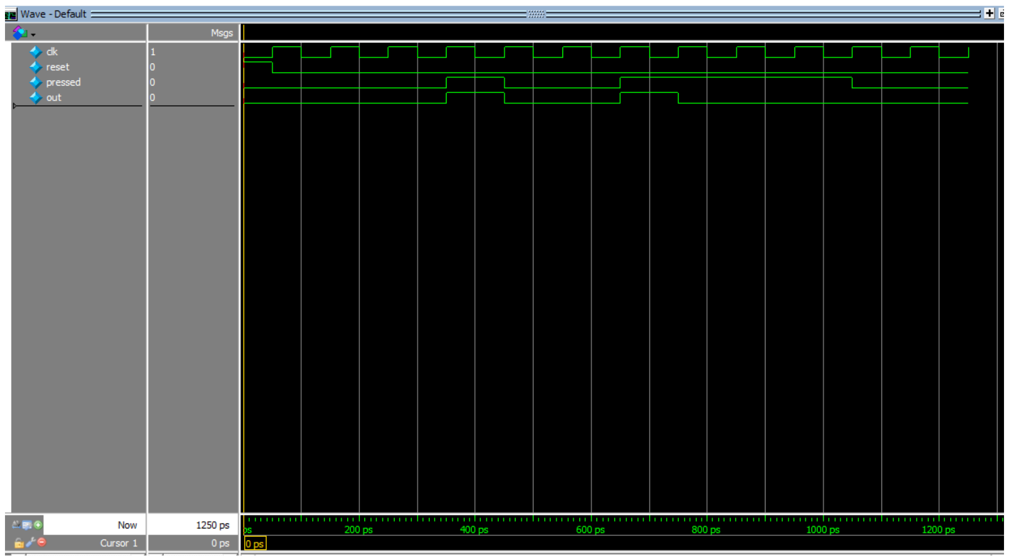

Figure 7: Simulation Result for Edge Detector

As can be seen in the simulation, when pressed is active, it is only high for one clock cycle, before going back to 0, even if pressed is held.

Module 5: Synchronizer

- **Purpose:** This module is used to deal with any metastability issues in the input.
- **Functionality:** The module runs input through a dual flip-flop to synchronize input.

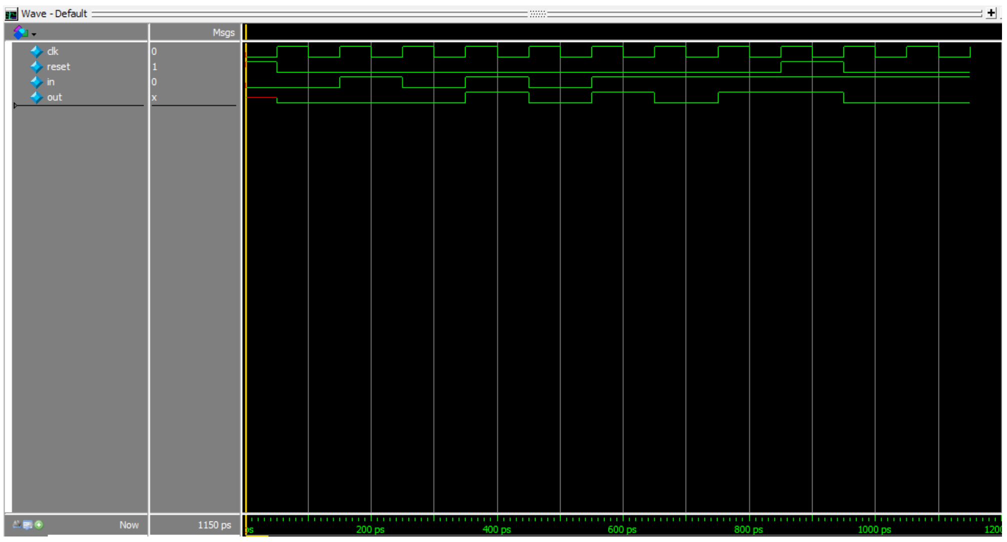

Figure 8: Simulation Result for Synchronizer

The ModelSim for the synchronizer module. The module is used to deal with metastability in user input. Runs the input through a dual flip flop to synchronize.

Module 6: Timer

- **Purpose:** The module used to track the time left for the users turn.
- **Functionality:** The module takes in a slower clock and counts down the time. When the user runs out of time, the timeOut field is set to true until they press the select button, which resets the timer. When timeOut is true, the player cannot change their selection, and will be forced to select the currently active square. The tiem elft is indicated by the number of LEDs left lit up on the board.

  Reset turns all the lights back on, and resets the count.

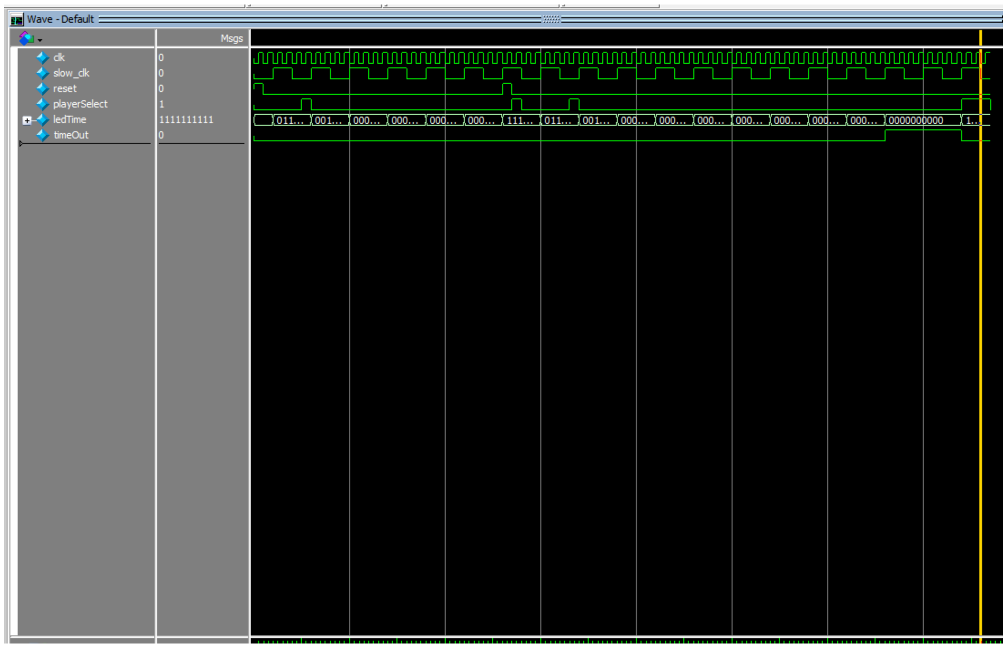

Figure 9: Simulation Result for Timer

The simulation shows the results of letting the timer run for a little bit and then pressing select or resetting which turns all the lights back on. When the timer reaches 0, timeOut becomes one until the player hits select.

Testing Methodology

To test the system, I first planned out how I was going to build my game, and I built it in a step by step manner, verifying each step worked before moving to the next. The first thing I setup was lighting up the board and being able to move the cursor. After that was accomplished, I setup a system to track which squares were chosen and being able to select them with the button. After that worked, I setup the victory module to check the win conditions and output to the display. I then moved to extra features such as the timer module and lighting up the row that won the game. For each step, I made a test bench to check the results, as well as actually deploying to the board to make sure it worked.

One challenge I faced was with synchronizer the timer module and the user input, as they would run on different clocks. I had to do some problem-solving to come up with a different way to lock the players input after the timer ran out that didn't directly control their input.

Finally, I also had others play test my game to try to find any faulty logic or interactions.

Estimated Time Spent

It took me an estimated 7-8 hours to complete this lab in its entirety.