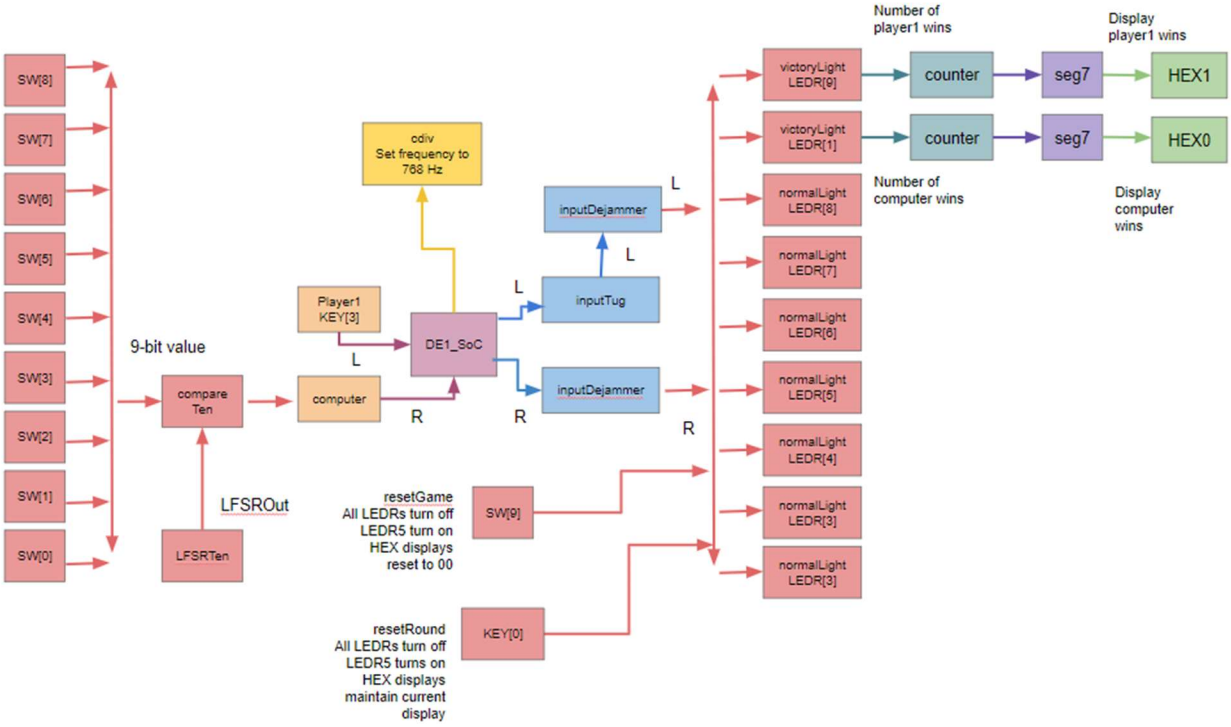

Fig 1. Block Diagram of System

This diagram illustrates how the two player inputs, KEY[3] and the randomized number as well as the switch combination for the computer, are used or transferred into each module to control the LEDRs.

Design Problem

Currently, Tug O'War is best known as pulling a rope between two groups or individuals with the goal of pulling the center of the rope completely over to one side of players. The side the rope is pulled over are the winners.

A digital version involving the DE1-SoC board allows for one player and a computer competitor to press (or, for the computer, simulate pressing) buttons to move a light from the center over to one side completely. Repeatedly pressing the button represents tugging on the rope. If the light moves completely over past the other side, the player on that side is victorious. The player's win count is displayed on the HEX display on the De1-SoC. The count increases with each round won, with a maximum of seven wins. The left hex represents player 1 wins with the right hex representing player 2, the computer, wins. Player 1's side is controlled by KEY[3] button and Player 2 is controlled by a computer. The LEDRs from 1 through 9 are the playfield, where center light LEDR5 is pulled left or right depending on which player successfully "tugs" the light to their side. If player 1 presses their button while player 2 does not, the light will shift to their side by one (from game start: LEDR[5] to LEDR[6]). If player 2 presses their button while player 1 does not, the light will shift to their side by one (from game start: LEDR[5] to LEDR[4]). If both buttons are pressed at the same time, there is no movement. Players must tap the button repeatedly to try to get the light fully past their side. Players are also prevented from pressing the button continuously.

Methods

System Design

The system inputs and outputs are controlled by the leading edge of a 768 Hz clock. Button presses are counted only if the button was pressed over a clock edge. The light will only ever move on a clock edge. This control over timing also ensures the button cannot be held down over multiple clock edges to count for multiple button taps, but instead just one. Additionally, player input is run through two flip-flops to ensure stable inputs (and therefore outputs).

When the light is moved off of the playfield from LEDR[1], player 2 is the winner. The same occurs for player 1 but for LEDR[9]. The win count for the winning player is then increased by one on the HEX display. Upon round reset, which is pressing KEY[0], the win count is maintained but another round can be played. The game is reset with SW[9], which will set the win count to 0 for both players. Upon both resets, LEDR[5] is lit until a player presses their button on a clock edge.

Computer Input

Below are the circuit designs and state diagrams for 3-bit and 4-bit LFSRs. These were first analyzed to later apply and design a 10-bit LFSR in Verilog. The 10-bit LFSR was then used to generate random numbers for the computer player.

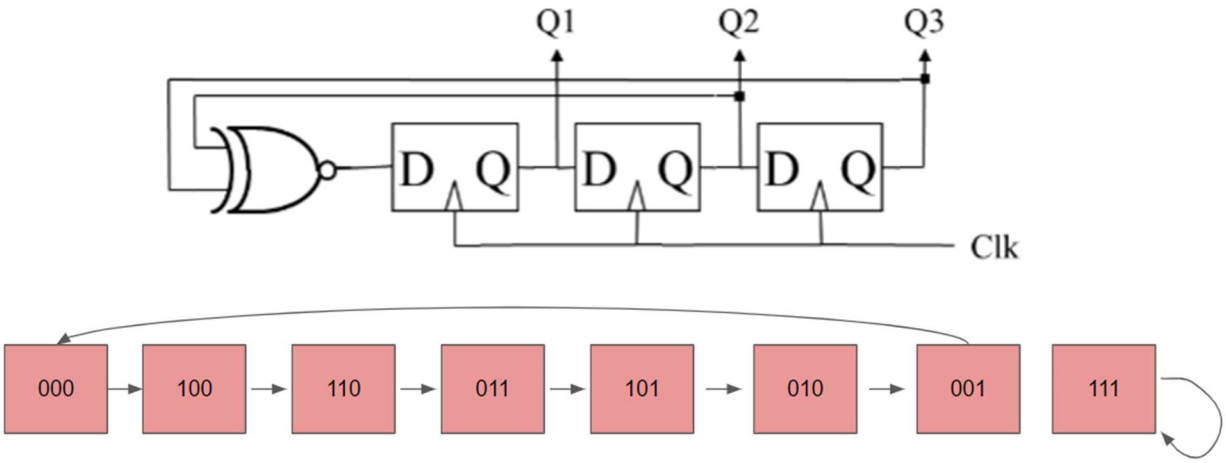

Fig 2. 3-bit LFSR circuit design and state diagram

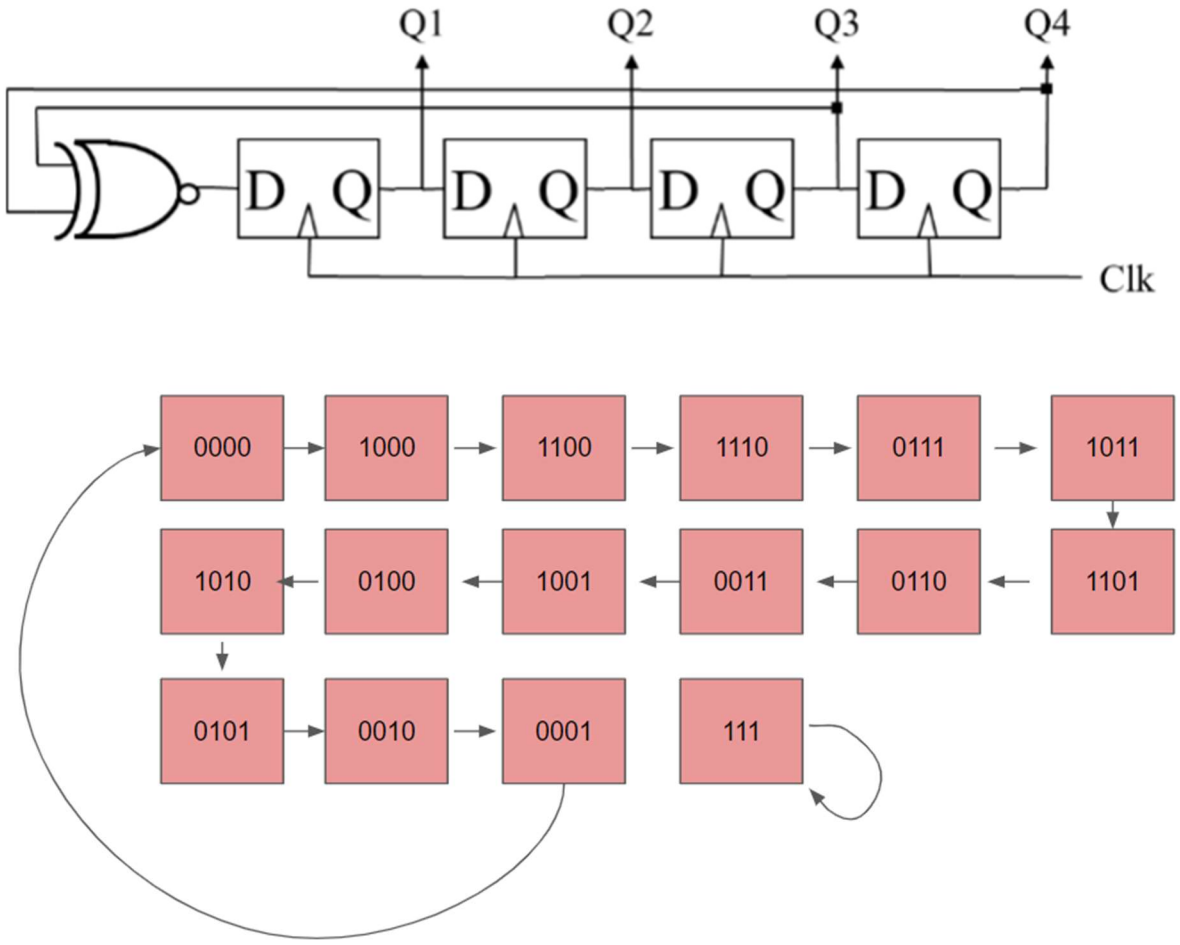

Fig 3. 4-bit LFSR circuit design and state diagram

Fig 4. Block diagram of compareTen, a 10-bit comparator that took the input of a randomly generated number, which is the 10-bit LFSR applying an expanded version of the 3-bit and 4-bit LFSRs above, and the input from the switches on the De1-SoC board.

Both Player Input

To control which LED is on, each LED is controlled by a module with inputs of L and R (whether the left or right button was pressed), and whether the next left and next right (NL and NR) lights are currently on. From this, the two states of whether a light is on or off can be determined.

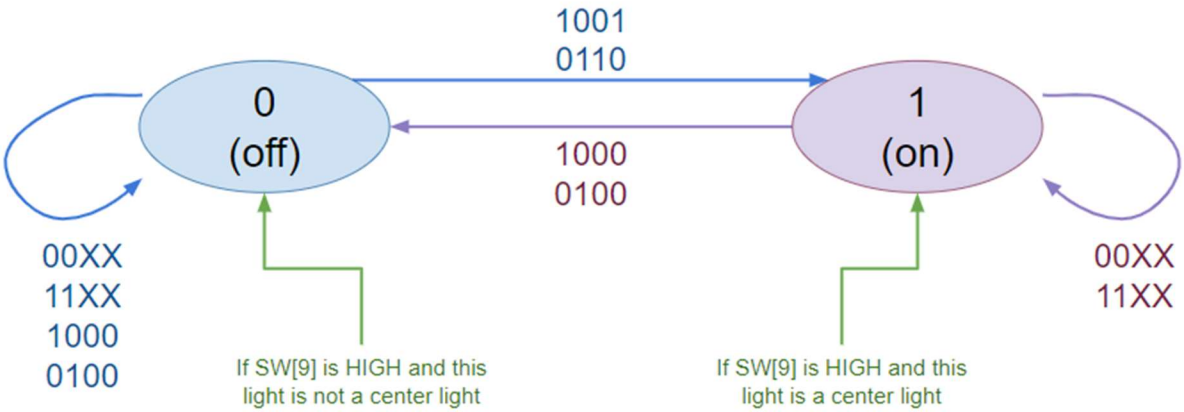

Fig 5. The state diagram for each playfield light, where it can be either on or off.

In Fig 5., a state diagram is used to convey the different states a playfield light can be in and what inputs may cause them to change. The order of the four-digit input is L, R, NL, and NR, where 0 represents a signal that is LOW and 1 represents a signal that is HIGH.

For an example case, if the light is currently off, but L is high (such that player 1 pressed their button) and NR is high, meaning that the next right light is on, then the light will turn on.

A center light has this functionality as well, but has separate logic that triggers on reset and not dependent on the button presses or adjacent lights (depending only on SW[9]), turning the light on. Therefore, if this is not the center light, then the light will turn off.

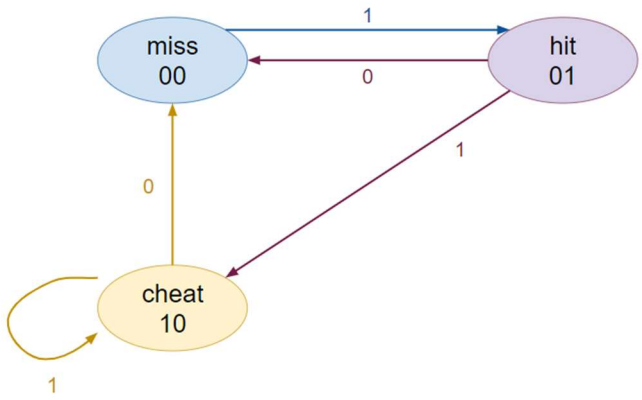

Fig 6. The state diagram for the inputTug module, which takes the user input and determines whether it is a miss, a valid hit, or a cheat (pressing button continuously). A 1 represents that the button is pressed, and 0 means unpressed.

In Fig 6., a state diagram is used to convey the different states for what an input can be designated as. This eventually controls whether the lights will be “tugged” or not.

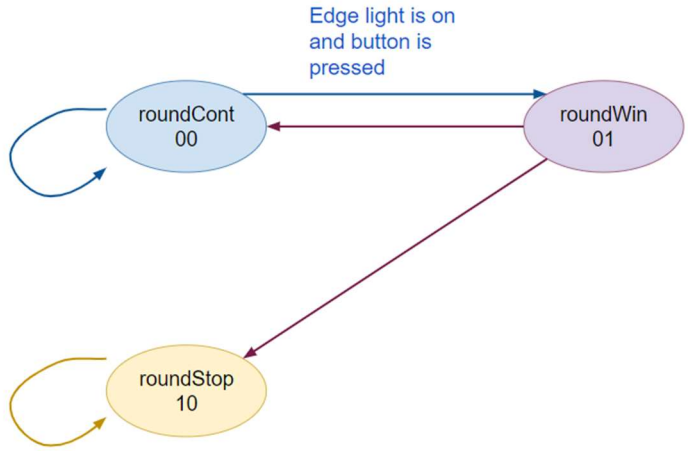

Fig 7. The state diagram for victoryLight, other than its states from normalLight. These states ensure that after a player has won the round that the round will end and no further changes can be made to the playfield. The only method to get out of roundStop is to reset the round, moving the game forward for another player to score.

In Fig 7., a state diagram is used to convey the different states for what the victoryLight can be designated as. This overall controls the round ending functionality.

These state diagrams were later transferred into Verilog to be simulated and then input into the De1-SoC board. Furthermore, the design was minimized for number of gates, which is illustrated below in the Resource Utilization by Entity report.

Resource Utilization by Entity Report:

The resources utilized to implement the Tug O'War design are shown in Figure 3. The most amount of gates in the overall design are in cdiv, necessary to divide the clock into a smaller frequency from 50MHz. Otherwise, the majority of gates are used in victoryLights, which not only has normal playfield light functionality, but also determines the round ending, increments any player wins, and controls the hex displays. Each individual light required the fewest, as they had very few states and very basic logic for whether the light was on or off. To minimize the design’s usage of resources, minimal flip flops were added, as well as gate-level implementations of some of the system’s elements, to ensure minimal logic gates being used for certain functions.

Resource Utilization by Entity

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	Full Hierarchy Name	Entity Name
1	⌄ |DE1_SoC	71 (1)	56 (0)	0	0	39	0	|DE1_SoC	DE1_SoC
1	|clock_divider:cdiv|	17 (17)	17 (17)	0	0	0	0	|DE1_SoC|clock_divider:cdiv	clock_divider
2	⌄ |computer:comp|	10 (0)	16 (1)	0	0	0	0	|DE1_SoC|computer:comp	computer
1	|LFSRTen:randGen|	1 (1)	10 (10)	0	0	0	0	|DE1_SoC|com...RTen:randGen	LFSRTen
2	|compareTen:compPressCheck|	7 (7)	1 (1)	0	0	0	0	|DE1_SoC|co...pPressCheck	compareTen
3	⌄ |inputTug:comp|	2 (2)	4 (2)	0	0	0	0	|DE1_SoC|co...putTug:comp	inputTug
1	|inputDejammer:metaPlayer|	0 (0)	2 (2)	0	0	0	0	|DE1_SoC|com...r:metaPlayer	inputDejammer
3	⌄ |inputTug:player1|	2 (2)	4 (2)	0	0	0	0	|DE1_SoC|inputTug:player1	inputTug
1	|inputDejammer:metaPlayer|	0 (0)	2 (2)	0	0	0	0	|DE1_SoC|inpu...er:metaPlayer	inputDejammer
4	|normalLight:led2|	2 (2)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:led2	normalLight
5	|normalLight:led3|	2 (2)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:led3	normalLight
6	|normalLight:led4|	2 (2)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:led4	normalLight
7	|normalLight:led5|	3 (3)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:led5	normalLight
8	|normalLight:led6|	2 (2)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:led6	normalLight
9	|normalLight:led7|	2 (2)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:led7	normalLight
10	|normalLight:led8|	2 (2)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:led8	normalLight
11	⌄ |victoryLight:led1|	13 (2)	6 (2)	0	0	0	0	|DE1_SoC|victoryLight:led1	victoryLight
1	|counter:player|	3 (3)	3 (3)	0	0	0	0	|DE1_SoC|vict...ounter:player	counter
2	|normalLight:victory|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|victo...lLight:victory	normalLight
3	|seg7:vicHex|	7 (7)	0 (0)	0	0	0	0	|DE1_SoC|vict...1|seg7:vicHex	seg7
12	⌄ |victoryLight:led9|	13 (2)	6 (2)	0	0	0	0	|DE1_SoC|victoryLight:led9	victoryLight
1	|counter:player|	3 (3)	3 (3)	0	0	0	0	|DE1_SoC|vict...ounter:player	counter
2	|normalLight:victory|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|victo...lLight:victory	normalLight
3	|seg7:vicHex|	7 (7)	0 (0)	0	0	0	0	|DE1_SoC|vict...9|seg7:vicHex	seg7

Fig 8. Resource Utilization by Entity

Displays resources used by each module involved in the design, under the top-level module DE1_SoC