Eugene Ngo

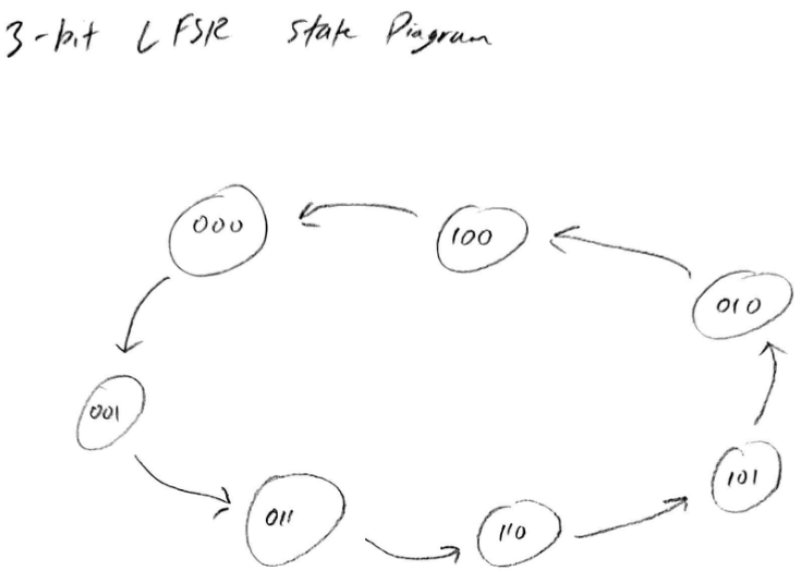
12/2/2022

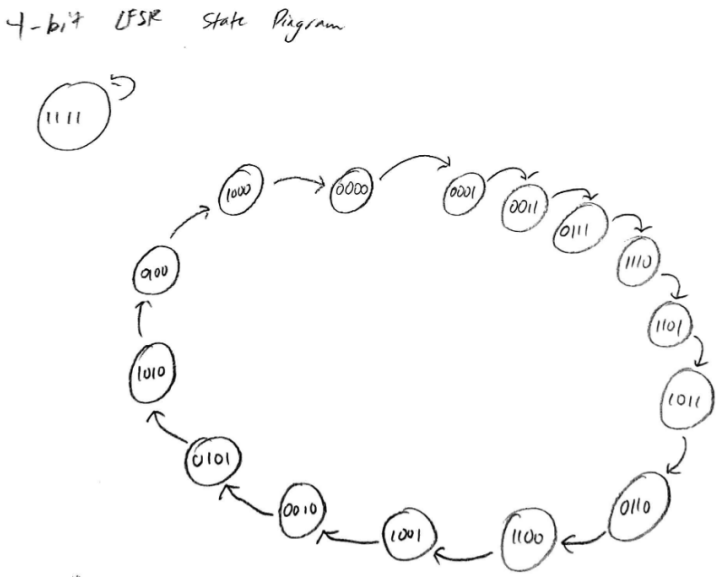
EE 271

Lab 5 Report

Procedure

The first step to implementing the computer input was to fully implement the 3-bit and 4-bit LFSRs into state diagrams.

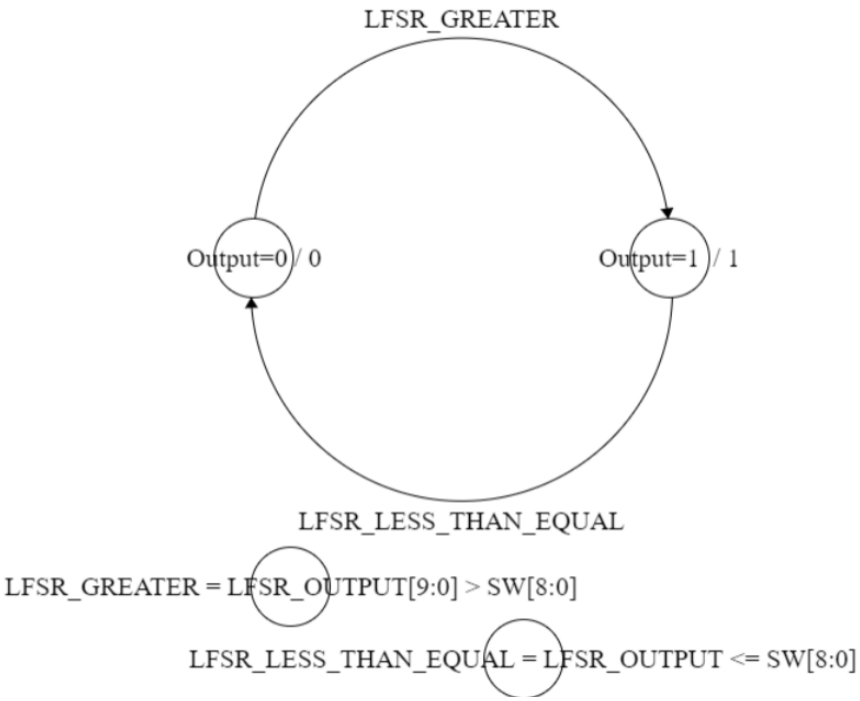




Then, the concept for the 4-bit LFSR was expanded to take in 10-bit input Q, and then XNOR everything from bit 1-9. Resulting in a state diagram with 1024 states, but the overall logic is that the output is a somewhat random 10 bit number. The code for implementing the LFSR is as provided:

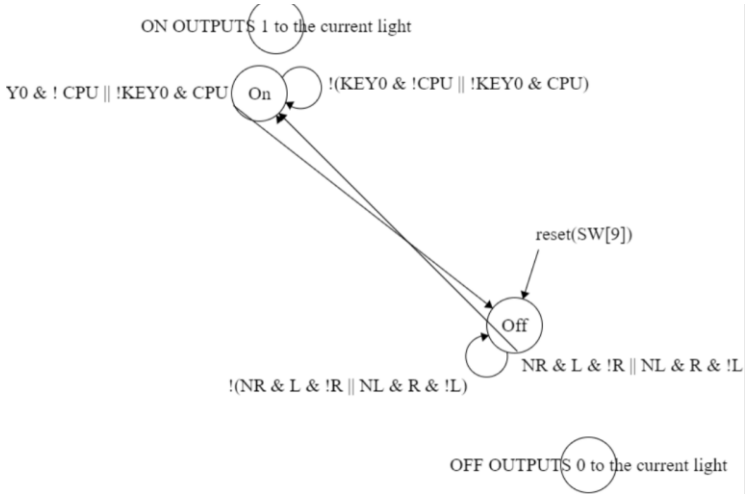
```
// LFSR module takes in 1-bit clk and 1-bit reset as inputs and returns 10-bit Q. This module is used to
// simulate computer button presses via a random number generator.

module LFSR(clk, reset, Q);
   output logic [9:0] Q;
   input logic clk, reset;

   logic xnor_out;

   assign xnor_out = (Q[0] ~^ Q[3]);

   always_ff @(posedge clk) begin
      if(reset) Q <= 10'b0000000000;

      else Q <= {xnor_out, Q[9:1]};
   end
endmodule
```

The last logic module to finalize the computer input is the comparing module. This is what the randomized LFSR value is compared to, to determine if the LFSR input is a 1 (button press) or 0 (button release). This is what ends up being the "CPU" input and is then processed using a DFF and a Userinput state machine to process the Metastability.

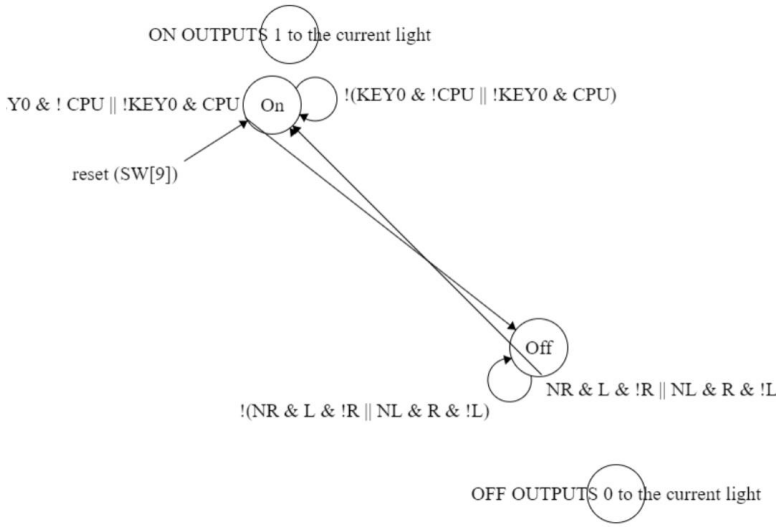


The rest of the state machines and logic used were derived from the last lab of Tug of War. There was a need to deal with the metastability of both CPU and Userinput, so both were run through a doubleFlipFlop and then those doubleflipped values were used to change states of the game. When approaching the Cyber war problem, the doubleflipped inputs from Key0 and the LFSR CPU input are then processed as inputs to the finite state machines, and an output is generated. Since each light LEDR is an output, there needs to be 9 separate finite state machines, all

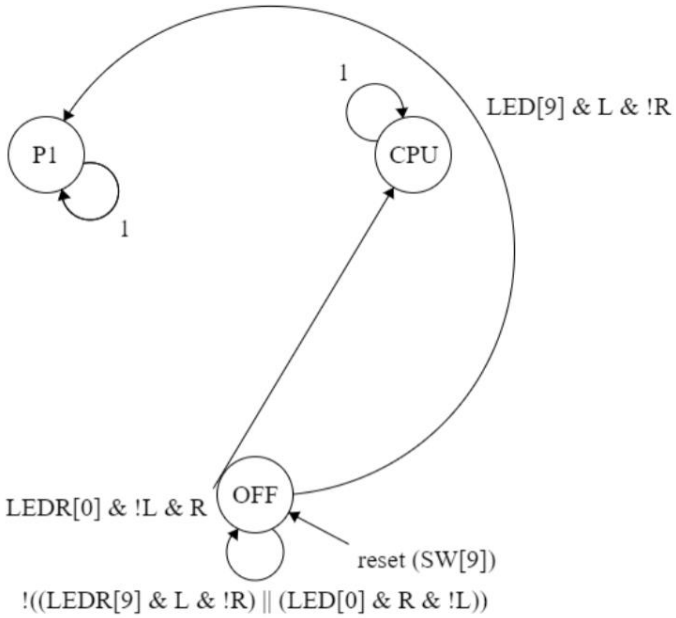
depending on each other. The output to LEDR is ON when the light next to it is ON, and the direction input is towards it. Or more specifically, if the LED on the left of the current LED is ON and the input direction is just to the right (only KEY 0 is pressed for that clock cycle), then the current LED is set to ON, and in the next clock cycle it will turn on. Taking this approach for all the LEDs enables the tug of war behavior. Another FSM for victory conditions is set, to run independent of the tug of war, where if either LEDR[9] or LEDR0 is ON and the inputs are left or right respectively, then HEX0 is set to 2 or 1 respectively, ending the game and displaying the winner on HEX0. To reset the game and play again, SW[9] is the reset switch and will be passed to all the different modules as a reset.



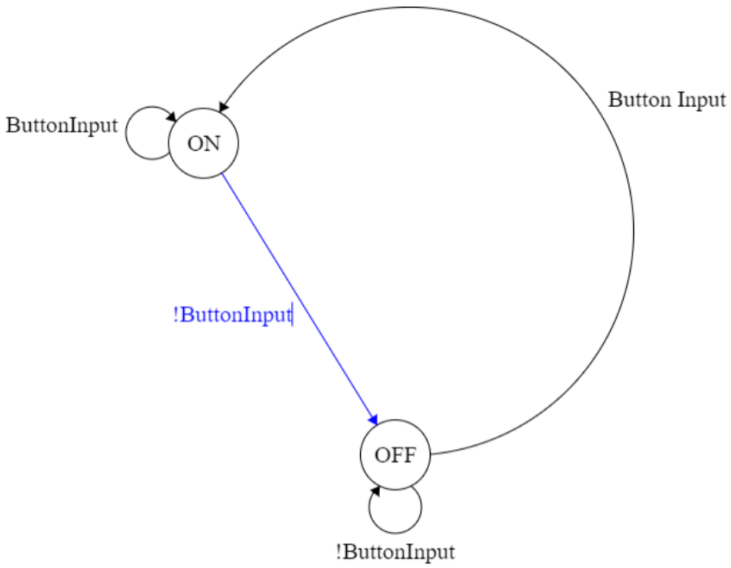
This FSM is repeated 8 times for LEDR0,1,2,3,4,6,7,8,9 where "current light is either one of those lights. NR and NL are the lights next to those LEDs, so in the case of LED 2 it would be NR is LED1 and NL is LED3. With the edge cases, the NR for LED0 is a 1 bit value of Zero and for LED9 the NL is a 1 bit value of Zero. LED 5 is not used for this FSM because it requires a special FSM which will be explained below. CPU and KEY0 are the main inputs and represent Computer input and human input respectively.



The only difference for LEDR[5]'s FSM is that it resets to ON (the tug of war starts at the middle)

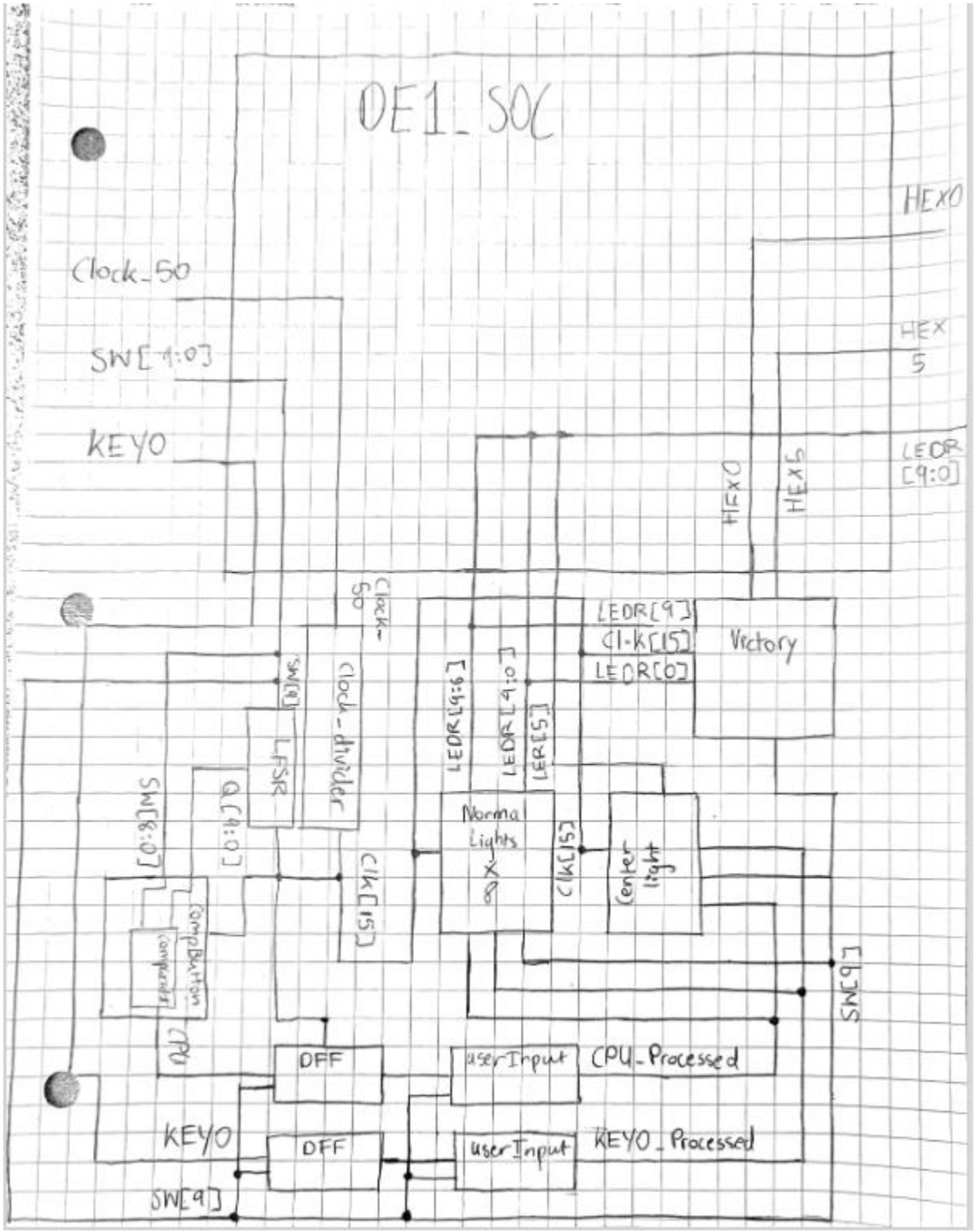


This FSM is for the winning condition. If the current light is on LEDR9 and the input is processed as Left, then the winner is P1. Which is then outputted to the HEX0. If the current light is on LEDR0 and the input is processed as Right, the winner is P2 which is then outputted to the HEX0

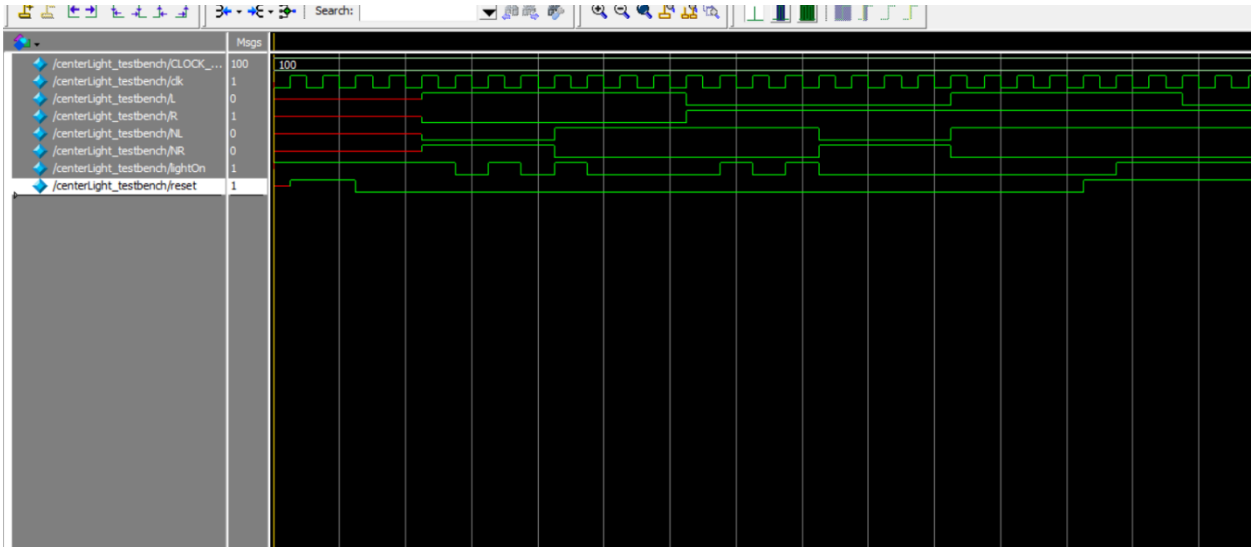


This last FSM exists to simplify the inputs from Key0 and CPU. The inputs are processed via a doubleflip before this and then processed via this finite state machine, making inputs take 2 clock cycles to be processed, dealing with metastability and simplifying inputs to be easier to process in other FSMs.

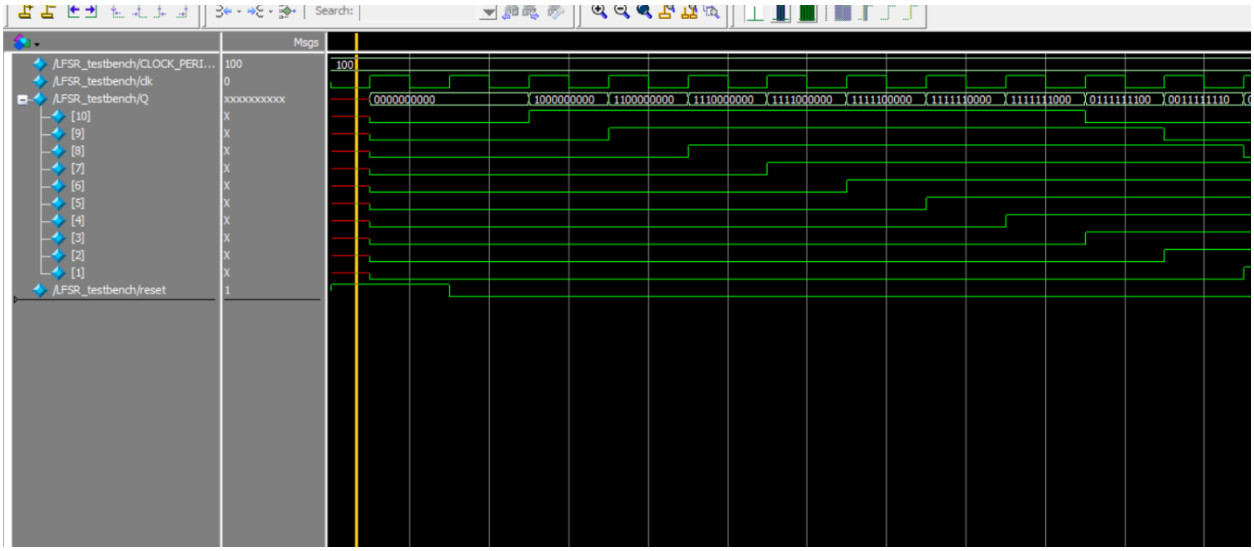
Top Level Block Diagram:



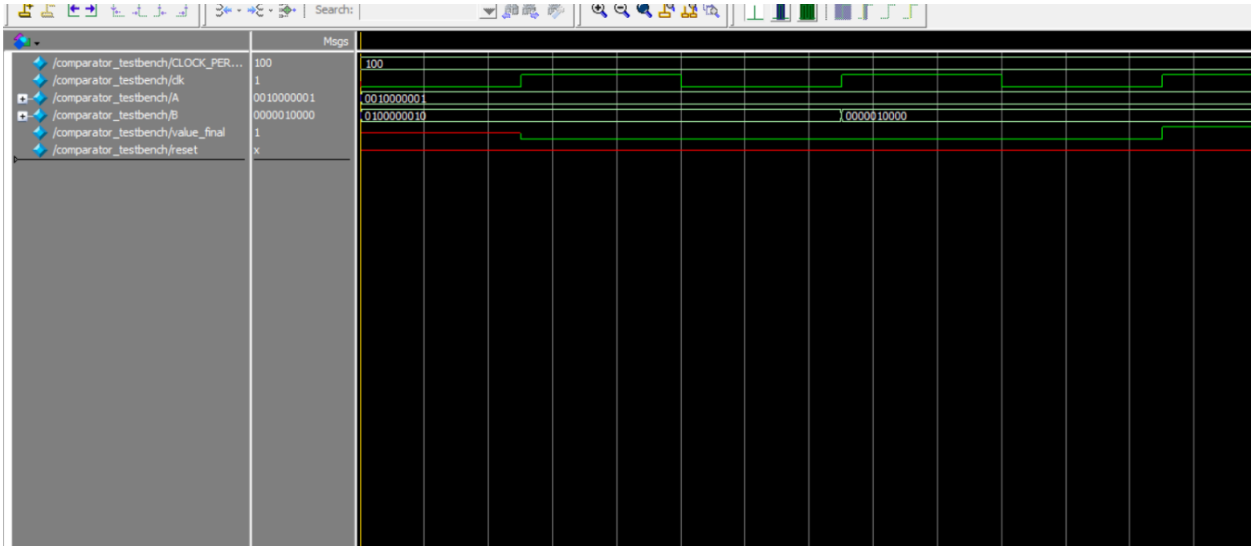
Results



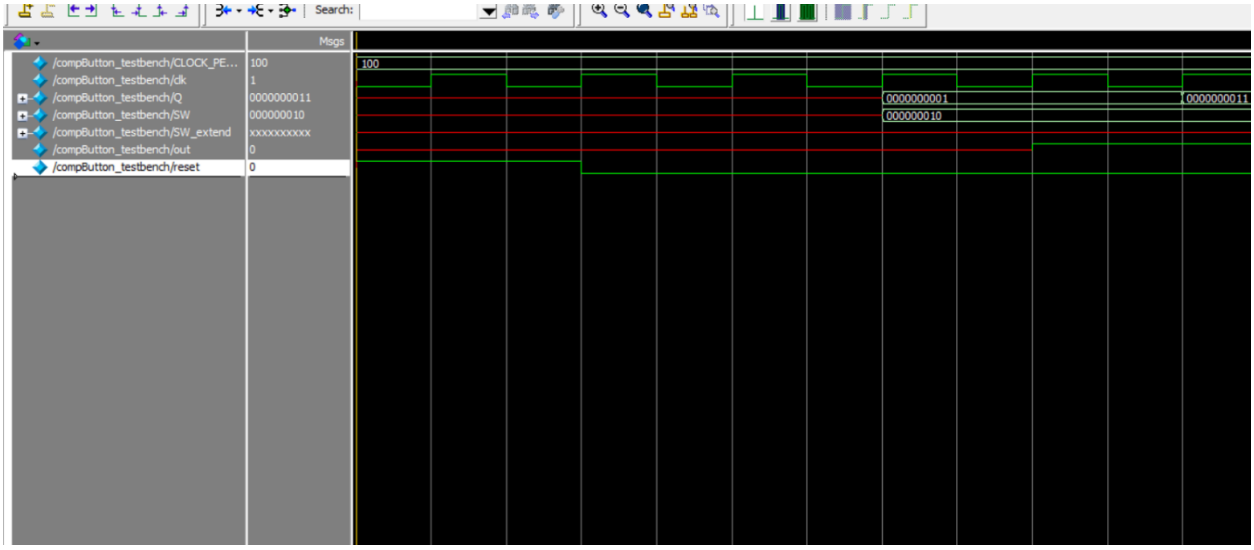
The centerLight simulation tests next state logic for the lights. This module is different from normalLight because on reset or restart, the center light will turn on which can be seen in the simulation.



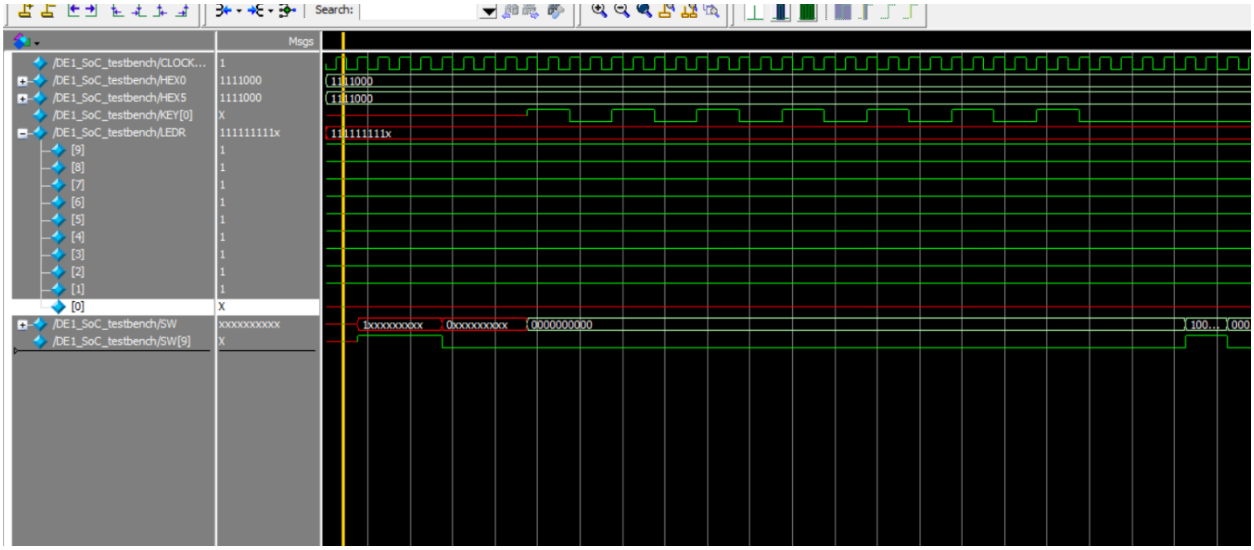
The LFSR simulation shows that at every clock cycle shown by clk a randomized value is generated.



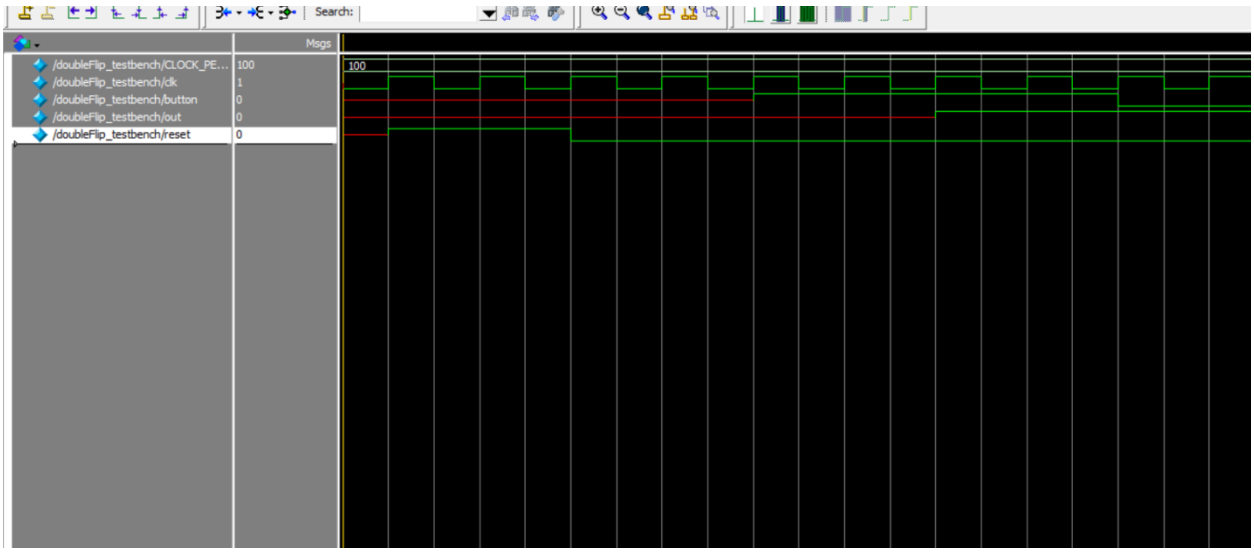
The comparator simulation shows that the output to the CPU input is dependent on the SW[8:0] and the LFSR value. If the LFSR value is greater than the SW[8:0] value then the CPU input is 1, otherwise it is 0.



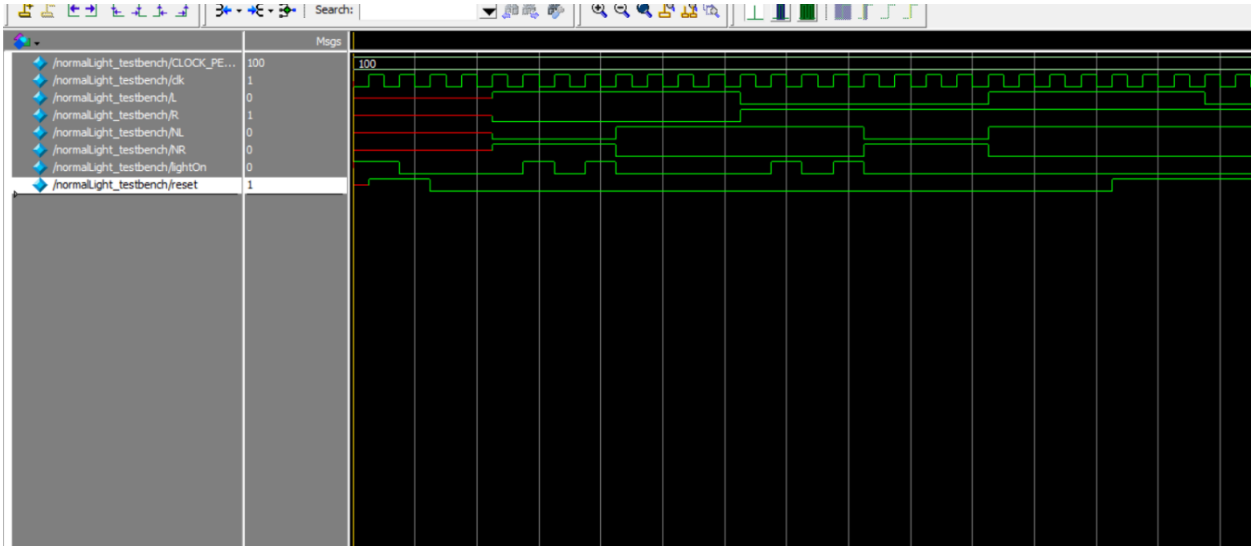
The compButton simulation shows that the output to the CPU input is dependent on the comparator module and follows the same logic as comparator.



The DE1_SoC simulation tests when a user starts the game and the human player (player 1) presses the button KEY0 multiple times until victory, then the game is reset and the human player makes no inputs and the computer does until the computer achieves victory. The score is shown on the HEX displays.

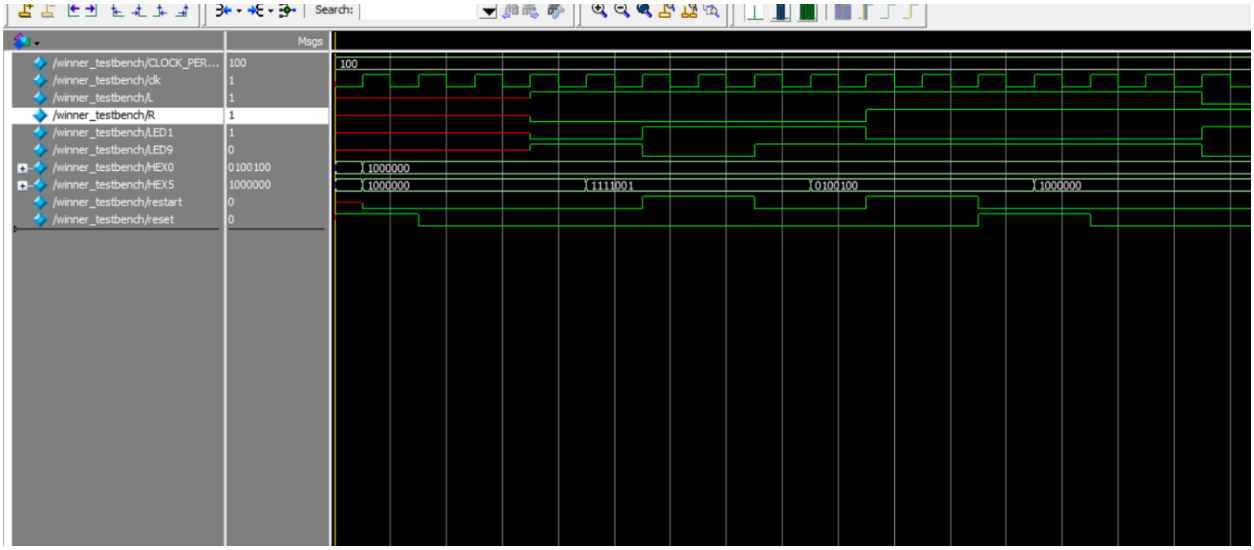


The doubleFlip simulation tests that when there is a button input, the output is updated to reflect the button input but two clock cycles later which is shown in the simulation. When button is positive, two clock cycles pass on clk and then the output, out, becomes true.



The normalLight simulation tests next state logic for the lights. This module is different from centerLight because on reset, the rest of the lights will be turned off as seen in the simulation.

The userInput simulation shows that the button press for cyber war must be a press and release. We can see in the simulation that the button is pressed and becomes one and then 3 clock cycles later once the button is released, then the output, out, returns true.



The winner simulation shows that when the player or computer scores a point, then the appropriate displayed score is incremented. In this case the computer wins multiple times so HEX5 updates to reflect that. The reset button also resets the score to 0 – 0 and each time a competitor scores, then the game is "restarted" and they compete to see who is first to 7 points.

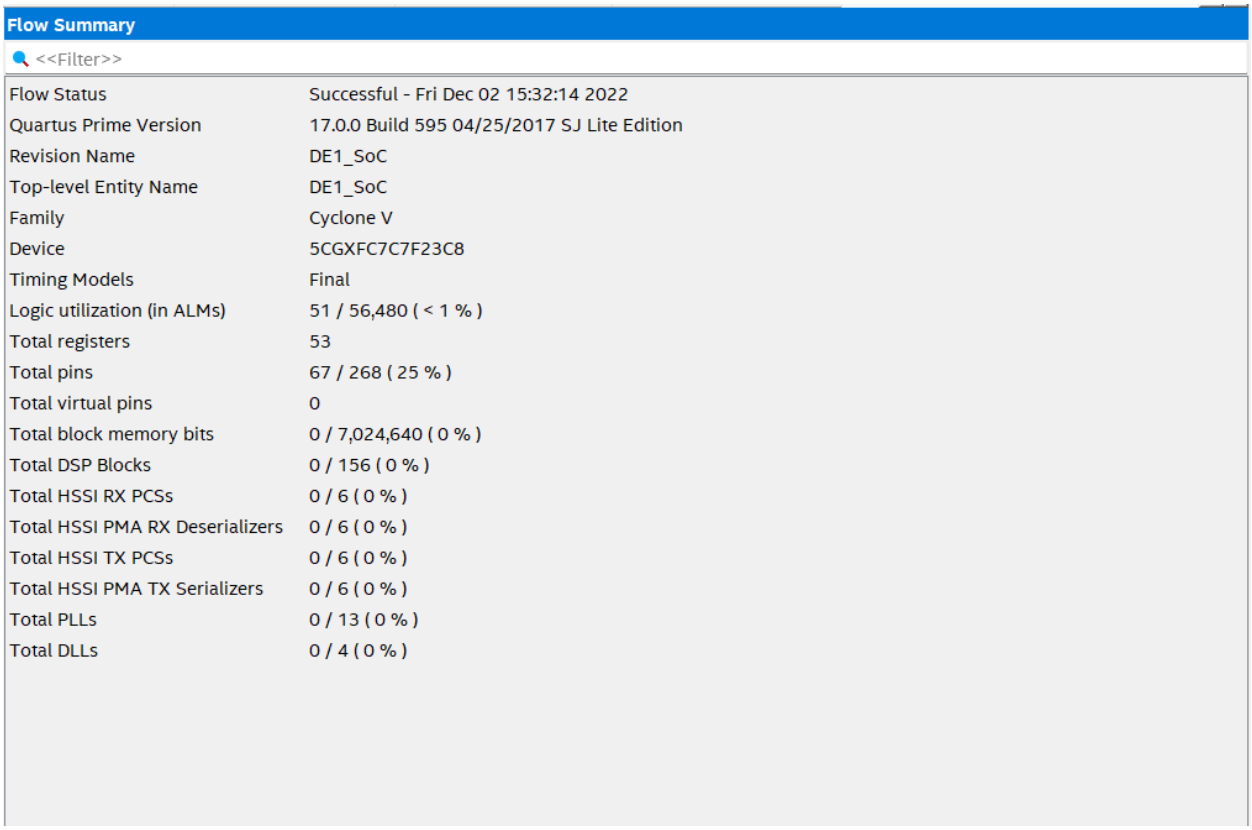
Resource Utilization by Entity:

Analysis & Synthesis 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	Library Name
1	|DE1_SoC	82 (0)	51 (0)	0	0	67	0	|DE1_SoC	DE1_SoC	work
1	|LFSR:random|	1 (1)	10 (10)	0	0	0	0	|DE1_SoC|LFSR:random	LFSR	work
2	|centerLight:L5|	2 (2)	1 (1)	0	0	0	0	|DE1_SoC|centerLight:L5	centerLight	work
3	|clock_divider:cdiv|	16 (16)	16 (16)	0	0	0	0	|DE1_SoC|clock_divider:cdiv	clock_divider	work
4	|compButton:comp|	7 (0)	1 (0)	0	0	0	0	|DE1_SoC|compButton:comp	compButton	work
1	|comparat...computer|	7 (7)	1 (1)	0	0	0	0	|DE1_SoC|compB...rator:computer	comparator	work
5	|doubleFlipFlop:flipFlop1|	1 (1)	2 (2)	0	0	0	0	|DE1_SoC|doubleFlipFlop:flipFlop1	doubleFlipFlop	work
6	|doubleFlipFlop:flipFlop2|	0 (0)	2 (2)	0	0	0	0	|DE1_SoC|doubleFlipFlop:flipFlop2	doubleFlipFlop	work
7	|normalLight:L1|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:L1	normalLight	work
8	|normalLight:L2|	3 (3)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:L2	normalLight	work
9	|normalLight:L3|	3 (3)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:L3	normalLight	work
10	|normalLight:L4|	3 (3)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:L4	normalLight	work
11	|normalLight:L6|	3 (3)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:L6	normalLight	work
12	|normalLight:L7|	3 (3)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:L7	normalLight	work
13	|normalLight:L8|	3 (3)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:L8	normalLight	work
14	|normalLight:L9|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:L9	normalLight	work
15	|userInput:computer|	2 (2)	1 (1)	0	0	0	0	|DE1_SoC|userInput:computer	userInput	work
16	|userInput:human|	2 (2)	1 (1)	0	0	0	0	|DE1_SoC|userInput:human	userInput	work
17	|winner:winnerFound|	31 (31)	9 (9)	0	0	0	0	|DE1_SoC|winner:winnerFound	winner	work

Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of resources of the given type used by the specific entity alone. The numbers listed outside of parentheses indicate the total resources of the given type used by the specific entity and all of its sub-entities in the hierarchy.

Flow Summary:



Flow Summary

<<Filter>>

Flow Status	Successful - Fri Dec 02 15:32:14 2022
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	DE1_SoC
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	51 / 56,480 (< 1 %)
Total registers	53
Total pins	67 / 268 (25 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	0 / 156 (0 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

Brief Overview:

Overall the resulting project fits the specs. When uploaded to the FPGA, a game of cyberwar is started and the human player is assigned to KEY0 while the computer opponent is assigned to KEY0. Then the two participants press their respective buttons as fast as they can until the LED which represents the rope, reaches one of either side. Once a winner is determined then a point is assigned to the appropriate winner: if the human wins, then the score displayed on HEX0 increments and if the computer wins then the score displayed on HEX5 increments. Once either competitor reaches 7 points they are the winner, the score is reset, the game is restarted, and a new game automatically begins. If at any point the game needs to be reset manually then switch 9 can be used to reset the game and continue to play. Switches 0 to 8 are used to toggle the computer opponent and determine the difficulty of the CPU.