Jyresa Mae M. Amboang
Frances Nicole S. Gigataras
Crishelle A. Agopitac
Erika Dianne Q. Canillo
Laboratory Submitted: 05/04/23
Laboratory Hours: 336 hours

Laboratory Activity No. 3
Adventure Game

Purpose / Introduction : In this laboratory activity the purpose is to create an adventure game in Quartus II. We have to build a finite state machine to do this. The state machine's state transition table needed to be built in the first stage. The player's current state included inputs for a reset and movement instructions as well as the room they were in. We were able to determine the logic functions for each of the outputs as well as the combination logic for the following states from the transition table. Lastly, we built the schematic in Quartus and performed the simulations in Modelsim.

Overview: Our approach was to first create the state transition table and logic equations for the room finite state machine. After that, we dealt with the state transition table and logic equations for the sword state machine. We used a one-hot encoding for the states, so the number of bits which equals the number of D flip-flops needed is given by the ceiling of the number of states.

Design:
STATE TABLES

ROOM FSM

STATE TRANSITION TABLE							
STATE	**N**	**S**	**E**	**W**	**R**	**V**	**NEXT STATE**
X	X	X	X	X	1	X	CC
CC	X	X	1	X	0	X	TT
TT	X	X	X	1	0	X	CC
TT	X	1	X	X	0	X	RR
RR	1	X	X	X	0	X	TT
RR	X	X	X	1	0	X	SS
RR	X	X	1	X	0	X	DD
SS	X	X	1	X	0	X	RR
DD	X	X	X	X	0	0	GG
DD	X	X	X	X	0	1	VV

OUTPUT TABLE			
Room Name	Binary Representation		
	SW	D	WIN
Cave of Cacophony	0	0	0
Twisty Tunnel	0	0	1
Rapid River	0	1	0
Secret Sword Stash	0	1	1
Dragon’s Den	1	0	0
Grievous Graveyard	1	0	1
Victory Vault	1	1	0

SWORD FSM

STATE TRANSITION TABLE			
CURRENT STATE	**SW**	**R**	**NEXT STATE**
X	X	1	NoSword
NoSword	0	0	NoSword
X	1	0	HasSword
HasSword	X	0	HasSword

OUTPUT TABLE	
CURRENT SATE	**V**
NoSword	0
HasSword	1

STATE ENCODINGS

ROOM	
Cave of Cacophony	**0000001**

Twisty Tunnel	**0000010**
Rapid River	**0000100**
Secret Sword Stash	**0001000**
Dragon’s Den	**0010000**
Grievous Graveyard	**0100000**
Victory Vault	**1000000**
SWORD	
NoSword	**0**
HasSword	**1**

REVISED ENCODED STATE TRANSITION TABLES AND OUTPUT TABLES

ROOM FSM

STATE TRANSITION TABLE							
STATE	**N**	**S**	**E**	**W**	**R**	**V**	**NEXT STATE**
X	X	X	X	X	1	X	0000010
0000001	X	X	1	X	0	X	0000010
0000010	X	X	X	1	0	X	0000001
0000010	X	1	X	X	0	X	0000100
0000100	1	X	X	X	0	X	0000010
0000100	X	X	X	1	0	X	0001000
0000100	X	X	1	X	0	X	0010000
0001000	X	X	1	X	0	X	0000100
0010000	X	X	X	X	0	0	0100000
0010000	X	X	X	X	0	1	1000000

OUTPUT TABLE			
CURRENT STATE	SW	D	WIN
0000001	0	0	0

0000010	0	0	0
0000100	0	0	0
0001000	1	0	0
0010000	0	0	0
0100000	0	1	0
1000000	0	0	1

SWORD FSM

STATE TRANSITION TABLE			
CURRENT STATE	**SW**	**R**	**NEXT STATE**
X	X	1	0
0	0	0	0
X	1	0	1
1	X	0	1

OUTPUT TABLE	
CURRENT SATE	**V**
0	0
1	1

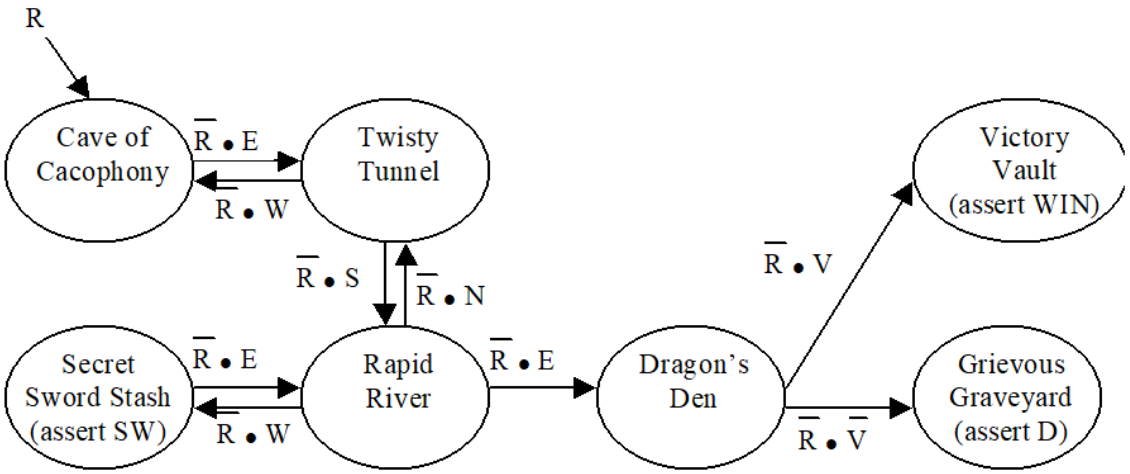
BOOLEAN EQUATIONS

ROOM FSM	
S1’	**R + S2• W**
S2’	**S1•E• !R + S3•N• !R**
S3’	**S2•S• !R + S4•E• !R**
S4’	**S3 + W• !R**
S5’	**S3 + E• !R**
S6’	**S5• !V• !R**
S7’	**S5• V• !R**

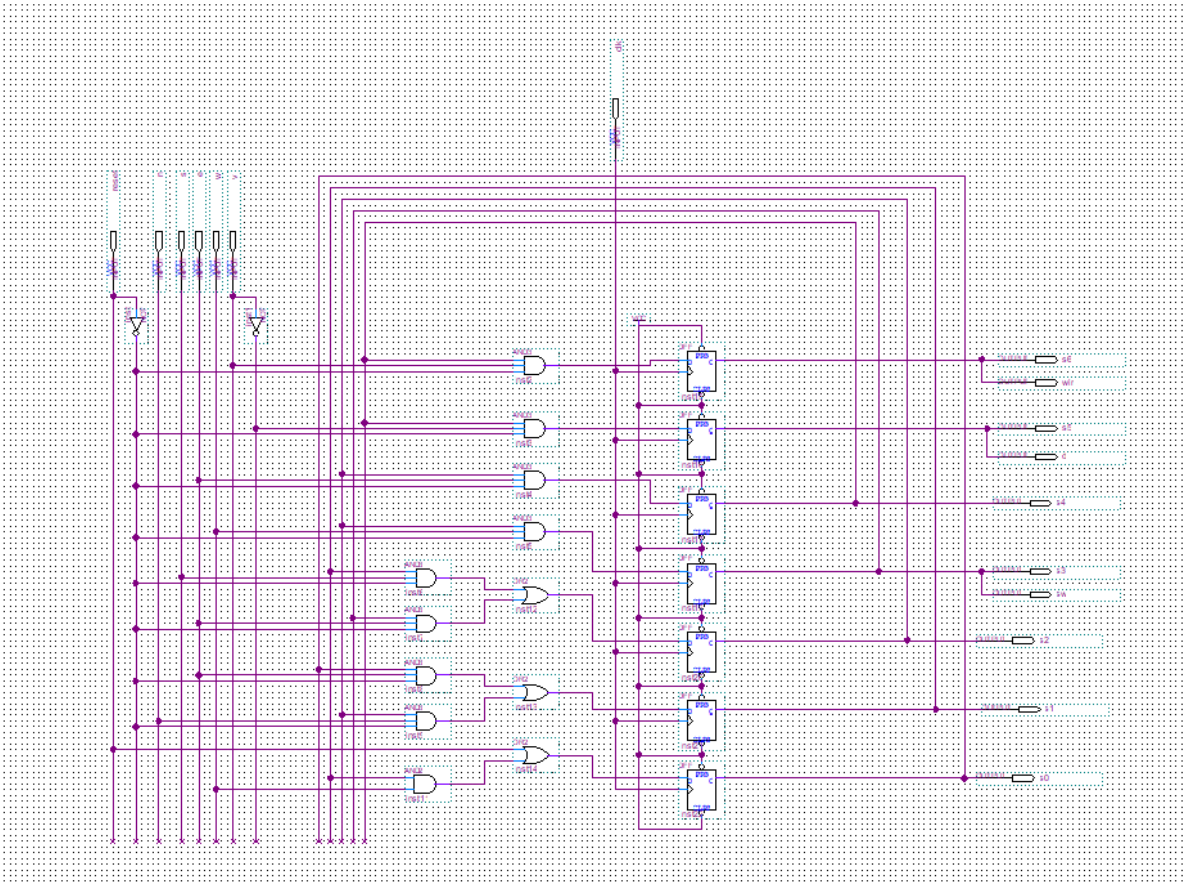
SW	S4
D	S6
WIN	S7

Results

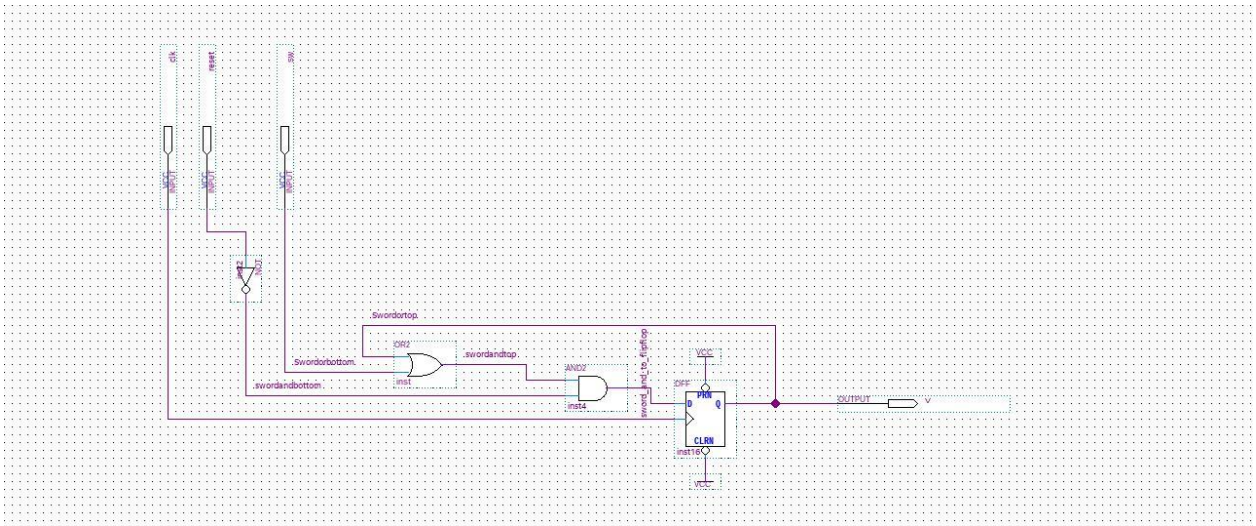
Complete State Transition Diagram for Room FSM



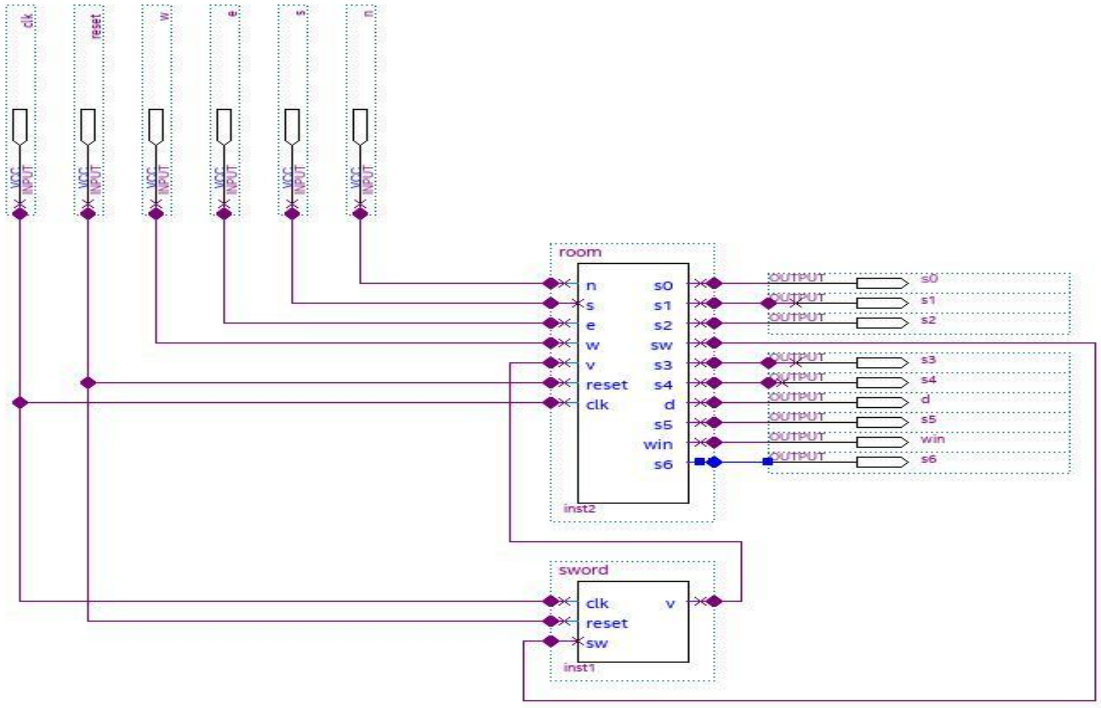
Room FSM schematic



Sword FSM Schematic



Full Schematic

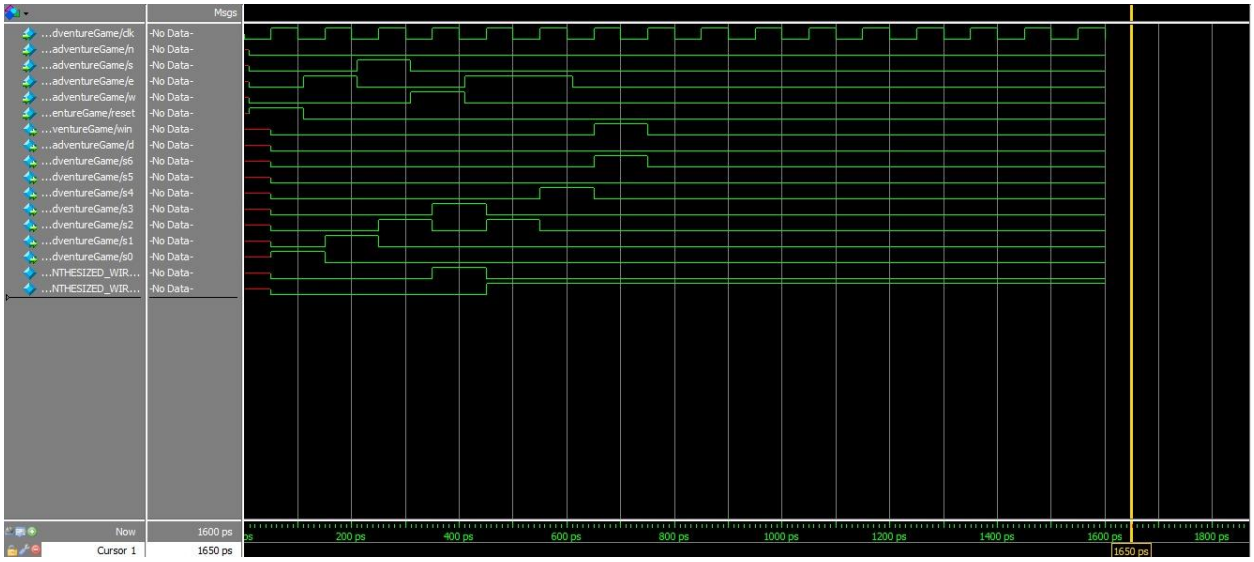


SIMULATION in MODELSIM

WINNING SEQUENCE

Testbench Code

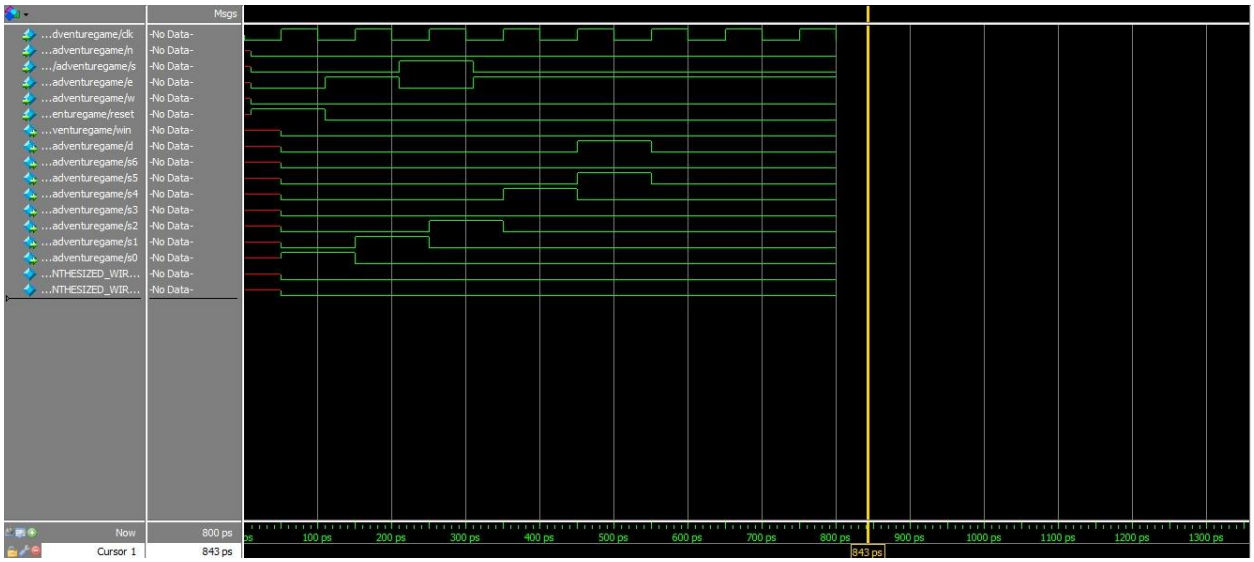
```
module testbench1();
	logic clk, n, s, e, w, reset;
	logic win, d, s6, s5, s4, s3, s2, s1, s0;

	// Rename project03_xx to the name of your top level module.
	// If you use different signal names, it will not work.
	project03_al adventuregame(.clk, .reset, .n, .s, .e, .w,
	                           .win, .d, .s6, .s5, .s4, .s3, .s2, .s1, .s0);

	// generate clock with 100 ns period
	initial
		forever begin
			clk = 0; #50; clk = 1; #50;
		end
	// apply inputs
	initial begin
		#10; // wait a bit so transitions don't occur on the clock edge

		// cycle 0: reset to Cave of Cacophony
		reset = 1;
		n = 0; s = 0; e = 0; w = 0;
		#100;

		// cycle 1: east to Twisty Tunnel
		reset = 0;
		e = 1;
		#100;

		// cycle 2: south to Rapid River
		e = 0;
		s = 1;
		#100;

// cycle 3: west to Secret Sword Stash
		s = 0;
		w = 1;
		#100;

		// cycle 4: east to Rapid River
		w = 0;
		e = 1;
		#100;

		// cycle 5: east to Dragon's Den
		#100;

		// cycle 6: won the game
		e = 0;
		#100;
	end

endmodule
```



LOSING SEQUENCE

Testbench Code

```
module testbench();

        logic clk, n, s, e, w, reset;
        logic win, d, s6, s5, s4, s3, s2, s1, s0;

        // Rename project03_xx to the name of your top level module.
        // If you use different signal names, it will not work.
        project03_al adventuregame(.clk, .reset, .n, .s, .e, .w,

                .win, .d, .s6, .s5, .s4, .s3, .s2, .s1, .s0);

        // generate clock with 100 ns period
        initial
                forever begin
                        clk = 0; #50; clk = 1; #50;
                end

        // apply inputs
        initial begin
                #10; // wait a bit so transitions don't occur on the clock edge
                // cycle 0: reset to Cave of Cacophony
                        reset = 1;
                        n = 0; s = 0; e = 0; w = 0;
                        #100;

                        // cycle 1: east to Twisty Tunnel
                        reset = 0;
                        e = 1;
                        #100;

                        // cycle 2: south to Rapid River
                        e = 0;
                        s = 1;
                        #100;

                        // cycle 3: east to Dragon's Den
                        s = 0;
                        e = 1;
                        #100;

                        // cycle 4: eaten by dragon
                        s = 0;
                        #100;
                end

endmodule
```



Conclusion / Summary

Using sequential logic circuit components, we practiced implementing finite state machines in this lab. In Particular, we learned how to use D flip flops to implement them. We also learned how to construct state transition tables involving multiple variables and how to derive the required logic functions from them. Using on-hot encoding made the implementation of the room FSM an extremely time-consuming task. With each encoding scheme, there are compromises to be made. Using one-hot encoding, which has more bits but fewer logic gates, is a common practice. The number of D flip flops needed for one-hot encoding is seven. While the lab was fairly tedious, we came away from it with a better understanding of how state machines are implemented in the circuit itself using Quartus II and simulating it in the Modelsim.