LAB REPORT – 6

Title: Designing a finite state machine(FSM) that implements an adventure game.

Purpose:

- The purpose of thus lab is to design FSM, simulate and implement an adventure game.
- Simulate the game using Modelsim.

Requirements:

- Altera Quartus II 9.1p2 software

Description:

Altera Quartus II 9.1p2 – It's a wed edition software which allow the user to analyze and synthesize the HDL designs, which enables the user to design the schematic model and simulate the design. Quartus implement the VHDL and Verilog for hardware description, vector waveform simulation and visual editing of logic circuits.

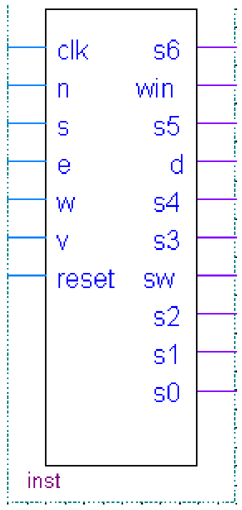
Adventure Game Description

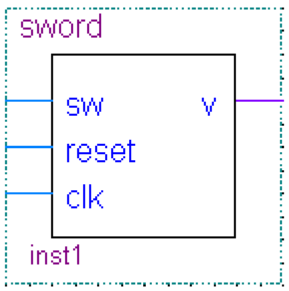
Step 1: The adventure game we will be designing has seven rooms and one object (a sword). The game begins in the Cave of Cacophony. To win the game, we must first proceed through the Twisty Tunnel and the Rapid River. From there, we will need to find a Vorpal Sword in the Secret Sword Stash. The sword will allow us to pass through the Dragon Den safely into Victory Vault (at which point you have won the game). If we enter the Dragon Den without the Vorpal Sword, we will be devoured by a dangerous dragon and pass into the Grievous Graveyard (where the game ends with you dead).

Step 2: This game can be factored into two communicating state machines, one state machine keeps track of which room you are in, while the other keeps track of whether you currently have the sword. The Room FSM is shown in Figure 1. In this state machine, each state corresponds to a different room. Upon reset, the machine's state goes to the Cave of Cacophony. The player can move among the different rooms using the inputs n, s, e, or w. When in the Secret Sword Stash, the SW output from the Room FSM indicates to the Sword FSM that the player is finding the sword.

Step 3: When in the Dragon Den, signal v, asserted by the Sword FSM when the player has the Vorpal Sword, determines whether the next state will be Victory Vault or Grievous Graveyard; the player must not provide any directional inputs. When in Grievous Graveyard, the machine generates the d (dead) output, and on Victory Vault the machine asserts the win output.

Step 4: In the Sword FSM (Figure 2), the states are "No Sword" and "Has Sword." Upon reset, the machine enters the "No Sword" state. Entering the Secret Sword Room causes the player to pick up a sword, so the transition to the "Has Sword" state is made when the SW input (an output of the Room FSM that indicates the player is in the Secret Sword Stash) is asserted. Once the "Has Sword" state is reached, the v (vorpal sword) output is asserted and the machine stays in that state until reset.





Symbol for Room FSN, showing its input and output

Symbol for sword FSM

Schematic Design

After working for more than 3 hours in lab we developed the adventure game design using Aletra Quartus II and simulated it using Modelsim.

Step 1: Creating a new schematic, click **File/New/VHDL,** after the windows opens click on the Templates icon and choose **VHDL/Full Designs/moore machine.** Then copy and paste the code it shows into the edit window. Then, edited the entity declaration to "**Game_Adventure**" eliminate one of the inputs, and turn the other one into a four-bit input array of STD_LOGIC named SW.

Step 2: Below is the code for adventure game.

```
-- Quartus II VHDL Template
-- Four-State Moore State Machine

-- A Moore machine's outputs are dependent only on the current state.
-- The output is written only when the state changes.  (State
-- transitions are synchronous.)

library ieee;
use ieee.std_logic_1164.all;

entity Game_Adventure is

	port(
	   clk: in     std_logic;
		N: in  std_logic;
		S: in  std_logic;
		E: in  std_logic;
		W: in  std_logic;
		--v    : in std_logic;
		reset: in     std_logic;
		win: out      std_logic;
		d: out std_logic;
		-- SW: out std_logic;
		F: out std_logic_vector(6 downto 0)

	);

end entity;

architecture rtl of Game_Adventure is

	-- Build an enumerated type for the state machine
	type state_type is (s0, s1, s2, s3, s4, s5, s6);

	-- Register to hold the current state
	signal state: state_type;
```

```
        signal direction: std_logic_vector(3 downto 0);
        signal v: std_logic;


begin

        -- Logic to advance to the next state
        process (clk, reset)
        begin
                if reset = '1' then
                        state <= s0;
                        d    <= '0';
                        win <= '0';
                elsif (rising_edge(clk)) then
                        case state is
                                when s0=>
                                        if direction = "0010" then
                                                state <= s1;
                                        else
                                                state <= s0;

                                        end if;
                                when s1=>
                                        if direction = "0001" then
                                                state <= s0;
                                        elsif direction="0100" then
                                                state <= s2;
                                        else
                                                state <= s1;
                                        end if;
                                when s2=>
                                        if direction = "1000" then
                                                state <= s1;
                                        elsif direction="0001" then
                                                state <= s3;
                                        elsif direction="0010" then
                                                state <= s4;
                                        else
                                                state <= s2;
                                        end if;
                                when s3 =>
                                        v <= '1';
                                        if direction = "0010" then
                                                state <= s2;
                                        else
                                                state <= s3;
                                        end if;
                                when s4 =>
                                        if v='1' then
                                                state <= s6;
                                                win <='1';
                                        else
                                                state <= s5;
                                                d <='1';
                                                end if;
                                        when others =>
                                        state <= s0;

                                end case;
                        end if;
                end process;


end rtl;
```

Step 4: Compile the given code and then simulate it using ModelSim.

State Transition Diagram

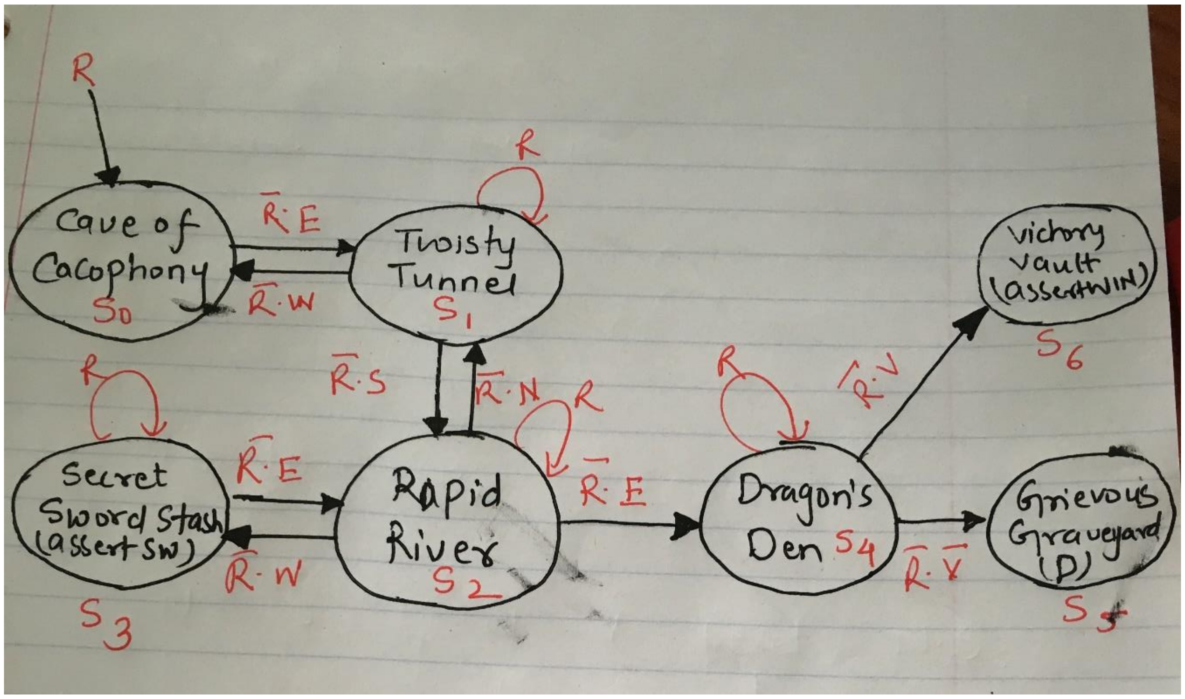


Fig. 1 State Transition diagram for Room FSM

Truth Table for FSM

Current State	N	S	E	W	V	Reset	Next Sate
S_0	0	0	0	0	0	1	S_0
S_0	0	0	1	0	0	0	S_1
S_1	0	0	0	1	0	0	S_0
S_1	0	1	0	0	0	0	S_2
S_2	1	0	0	0	0	0	S_1
S_2	0	0	0	1	0	0	S_3
S_3	0	0	1	0	0	0	S_2
S_2	0	0	1	0	0	0	S_4
S_4	0	0	0	0	1	0	S_6(win)
S_4	0	0	0	0	0	0	S_5(Dead)

Simulation

After designing the schematic model of seven segment we simulate the design in HDL such as Verilog HDL using ModelSim simulator. Following are the steps followed to simulate the design:

Step 1: We simulate the design using ModelSim. ModelSim expects a description of a circuit in a hardware description language (HDL) such as Verilog. To convert our schematic to Verilog, we opened the schematic and chose File - Create / Update, Create HDL Design File for Current File. Chose Verilog HDL.

Step 2: Now fire up ModelSim SE 10.1b from the Tools, RTL Run Simulation Tool pulldown.

Step 3: Chose **Compile / Compile All** to compile the Verilog code into a form that ModelSim can simulate. Then chose **Simulate / Start Simulation**.

Step 4: When the simulator starts, ModelSim will open more panes including sim and Objects that help us select signals for the waveform viewer. In the objects window, we'll see all the inputs, outputs, and internal wires. Shift-click to select them all. Then right-click and choose Add to Wave, Selected Signals. A Wave pane will pop up with the signals.

Step 5: To add the wave position, right-click on the names in the object pane and choose Add to wave. Finally, we apply the inputs. In the transcript pane at the bottom, type

Force sim: game_adventure/ Reset

Force sim: game_adventure/Direction

Force sim: game_adventure/v

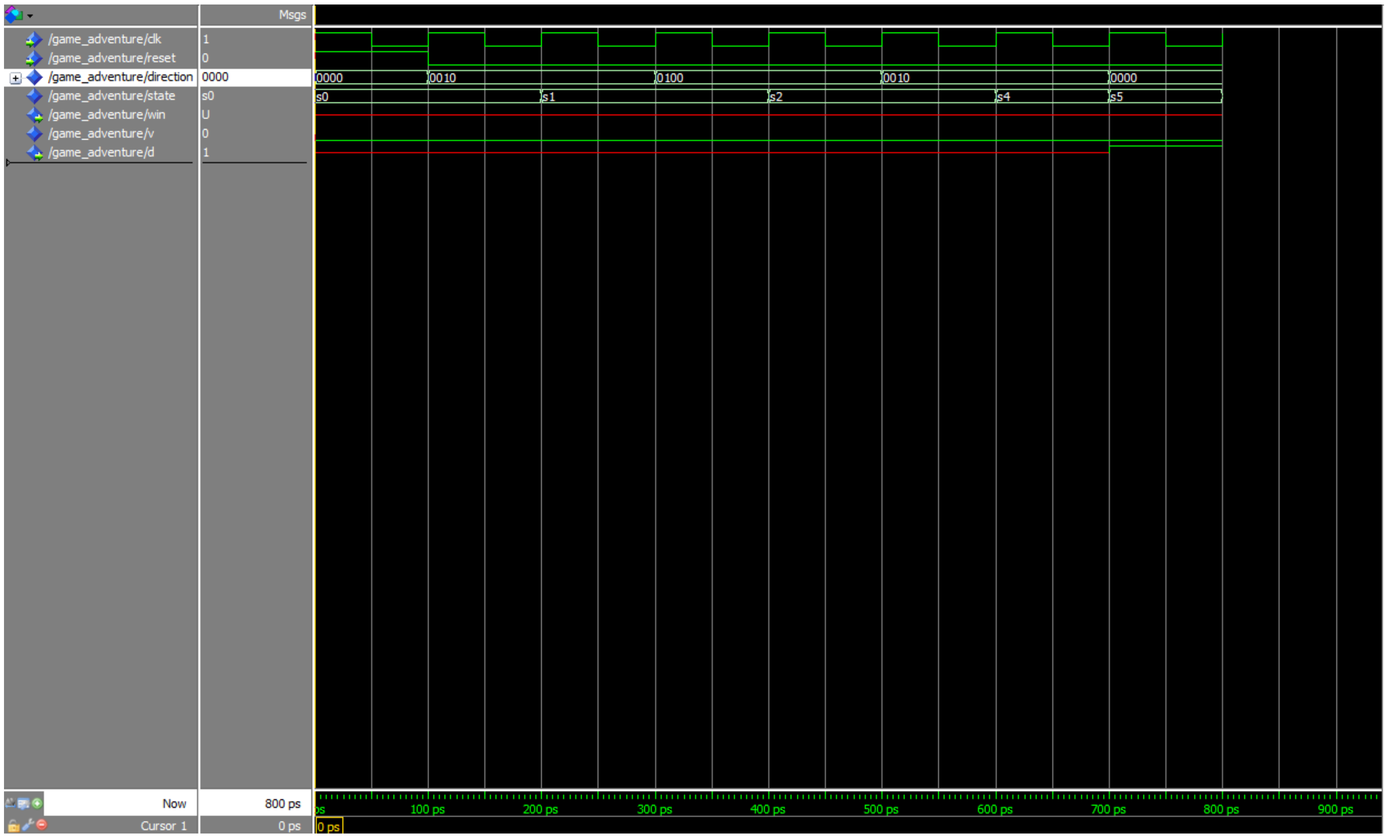


Fig.2 Simulation waveform of Dead state (Without sword)

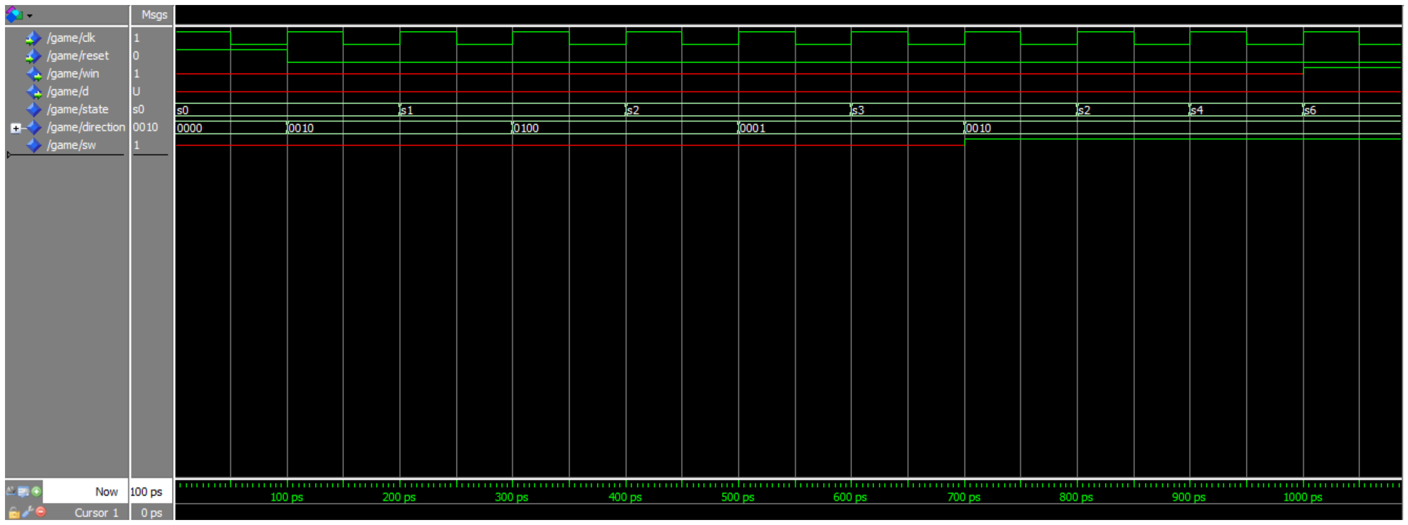


Fig.3 Simulation waveform of Win state (With sword)

Arnika Vishwakarma (@01367603)