Lab 6: Adventure Game

Lina Mi

ID:@01377283

Purpose:

1) Master Finite State Machine and be able to draw a state transition diagram by analyzing the situation.
2) Analyzing the Adventure Game; design a Finite State Machine to implement an adventure game, including draw state transition diagram for the two parts of game, namely Room FSM and Sword FSM.
3) Use VHDL to program to implement the Adventure Game.
4) Use ModelSim software to simulate the procedure of designed game, analyze and correct the programming problems, obtain the simulation results (simulation waves).

Content:

1) Adventure Game :

This game has seven rooms and one object(a sword). The game begin in the Cave of Cacophony. In order to win the game, the player must first proceed through the Twisty Tunnel and Rapid River. From there, player will find a Vorpal Sword in the Secret Sword Stash. The sword will allow you to pass through the Dragon Den safely into Victory Vault (at which point, the player won the game). If the player enter the Dragon Den without the Vorpal Sword, he will be devoured by a dangerous dragon and pass into Grievous Graveyard (where the game ends with player dead). The game can be factored into two communicating state machine: one state machine keeps track of which room the player are in, while the other keeps track of whether the player currently have the sword. According to previous description, the Room FSM is shown in Figure 1. In this state machine, each state corresponds to a different room. Upon reset, the start state is the Cave of Cacophony. The player can move among the different rooms using the direction input (n,s,e,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. 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 assert the win output.

2) State transition diagrams, state transition and output tables :

From previous description on the Adventure Game, the diagram of Room FSM and Sword FSM are drawn as Figure 1 and Figure 2

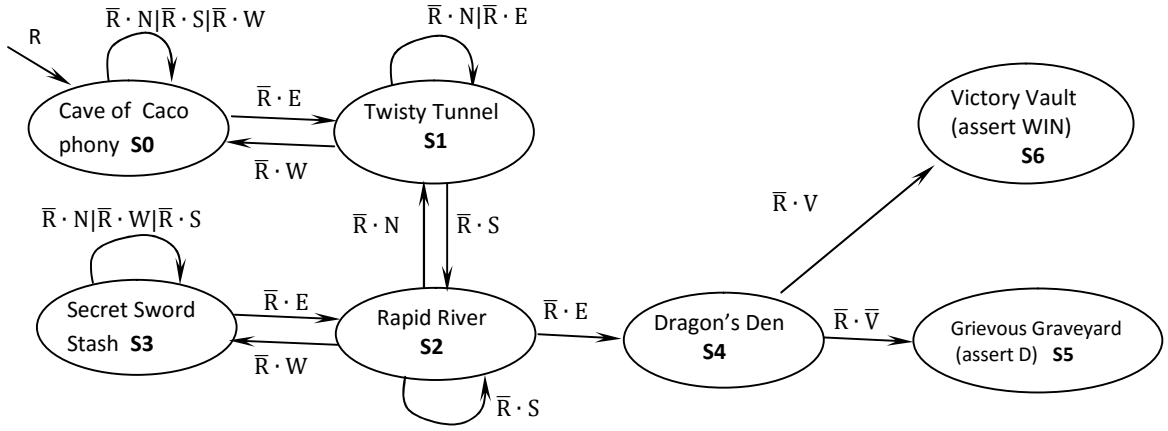


Figure 1 State Transition Diagram for Room FSM

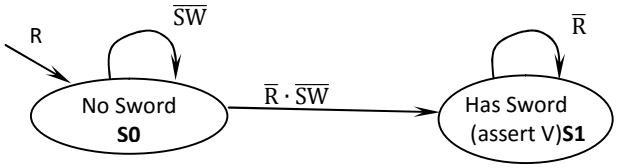


Figure 2 State Transition Diagram for Sword FSM

According the Figure1, the corresponding state transition table for Room FSM with direction input is shown as following

Current State	CLK	Reset	V	Direction Inputs				Next State
				N	S	E	W	
X	X	1	X	X	X	X	X	S0
S0	1	0	0	0	0	0	1	S0
S0	1	0	0	0	0	1	0	S1
S0	1	0	0	0	1	0	0	S0
S0	1	0	0	1	0	0	0	S0
S1	1	0	0	0	0	0	1	S0
S1	1	0	0	0	0	1	0	S1
S1	1	0	0	0	1	0	0	S2
S1	1	0	0	1	0	0	0	S1
S2	1	0	0	0	0	0	1	S3
S2	1	0	0	0	0	1	0	S4
S2	1	0	0	0	1	0	0	S2
S2	1	0	0	1	0	0	0	S1
S3	1	0	0	0	0	0	1	S3
S3	1	0	0	0	0	1	0	S2
S3	1	0	0	0	1	0	0	S3
S3	1	0	0	1	0	0	0	S3
S4	1	0	0	X	X	X	X	S5
S4	1	0	1	X	X	X	X	S6

The outputs table with current state for Room FSM :

Current State	State Outputs							SW	Win	d
	S0	S1	S2	S3	S4	S5	S6			
S0	1	0	0	0	0	0	0	0	0	0
S1	0	1	0	0	0	0	0	0	0	0
S2	0	0	1	0	0	0	0	1/0	0	0
S3	0	0	0	1	0	0	0	1	0	0
S4	0	0	0	0	1	0	0	1/0	0	0
S5	0	0	0	0	0	1	0	0	0	1
S6	0	0	0	0	0	0	1	1	1	0

According to Figure 2, the state transition table for Sword FSM is shown as following:

Current State	Inputs		Next State
	Reset	SW	
X	1	X	S0
S0	0	0	S0
S0	0	1	S1
S1	0	0	S1
S1	0	1	S1

The outputs table with current state for Sword FSM :

Current State	Output V
S0	0
S1	1

3) VHDL programming

In this lab, we program with VHDL(VHSIC Hardware Description Language) to implement the above state transition table for Room FSM in Altera Quartus software, and hence finish the design of Adventure Game.

The following is the written program:

```
-- 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 : instd_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;
```

4) Simulation using ModelSim

The designed Adventure Game using above VHDL program was simulated in ModelSim to verify its correctness

The simulation results is shown in Figure 3 and 4:

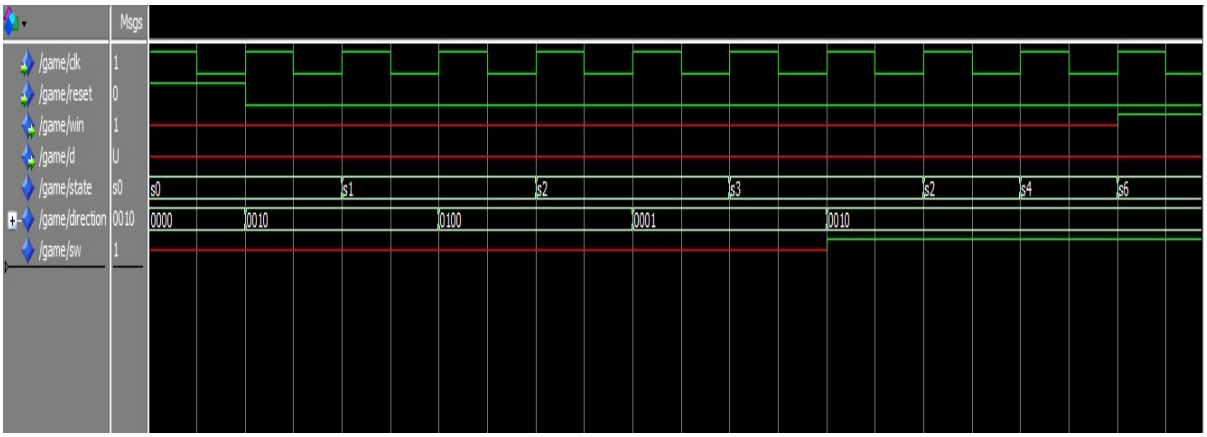
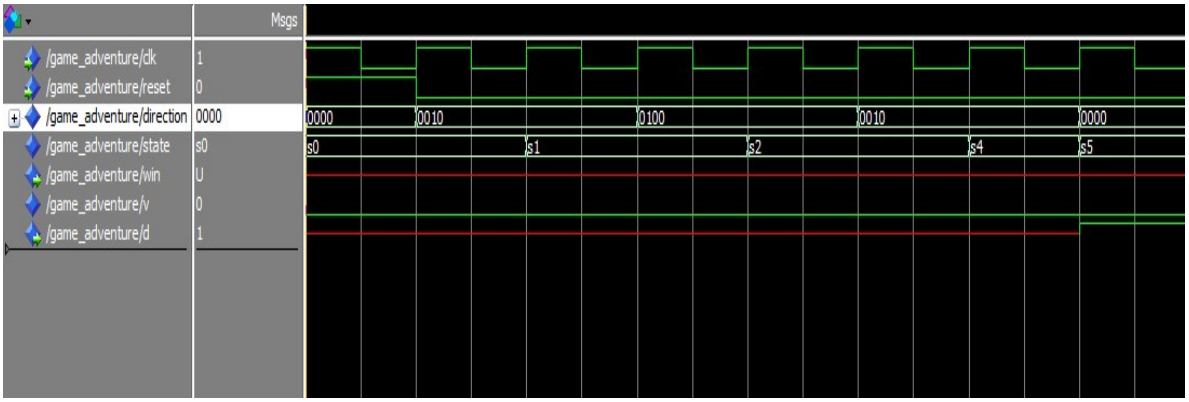


Figure 3 Simulation Results of Adventure Game in case of Win



.

Figure 4 Simulation Results of Adventure Game in case of Dead

From Figure 3 and 4, for both cases of Win and Dead, it can be seen that designed Adventure Game can successfully implement the state transition of Room FSM under each combination of current state and direction inputs and sword state input, and output correct signals for each output variable. From Figure 3, the simulation result showed the correct procedure (state sequence) to win, and so did the simulation result to dead in Figure 4.

5) Conclusions: (my lab partners and I used 3 hours to finish the Lab 6)

From the lab 5,

1) I can draw state transition diagram for the project based on the function description of Adventure Game, and list the state transition table under different current state and inputs and output table;
2) I can use the VHDL to implement the FSM;
3) I can use ModelSim software to simulate the operation of designed game and thus to test the correctness of designed project.

What I did in Lab 6

1) complete the state transition diagram for both Room FSM and Sword FSM ;
2) List the state transition table and output table for both FSMS;
3) programming with VHDL language to implement the state transition table of Room FSM;
4) Debug the program with lab partner;
5) make simulation on designed Game using ModelSim with my lab partner;

What my lab partner did:

1) Complete the state transition diagram for both Room FSM and Sword FSM;
2) List the state transition table and output table for both FSMS;
3) Programming with VHDL language together with me to implement designed Room FSM;
4) Debug the program and compile the program successfully;
5) Simulate the designed FSM using ModelSim and get simulation results;