# Assignment-2

Name: Shivam Shukla

Roll No.: 2022478

Advisor: Sneh Saurabh

Date: 24/09/2024

Course: VDF



INDRAPRASTHA INSTITUTE of INFORMATION TECHNOLOGY **DELHI** 

#### Q1) Snippet of the log files to prove that I have indeed run the tools



```
shivam@Shivam-Shukla: /mn ×
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.153.1-microsoft-standard-WSL2 x86 64)
                   https://help.ubuntu.com
* Documentation:
                   https://landscape.canonical.com
 * Management:
 * Support:
                   https://ubuntu.com/pro
This message is shown once a day. To disable it please create the
/home/shivam/.hushlogin file.
shivam@Shivam-Shukla:~$ cd /mnt/d/Sem_5/VDF/Assignments/Assignment_2/icarus_codes
shivam@Shivam-Shukla:/mnt/d/Sem_5/VDF/Assignments/Assignment_2/icarus_codes$ iverilog -Wall -o test fsm.v fsm_tb.v
shivam@Shivam-Shukla:/mnt/d/Sem_5/VDF/Assignments/Assignment_2/icarus_codes$ vvp test
VCD info: dumpfile fsm.vcd opened for output.
shivam@Shivam-Shukla:/mnt/d/Sem_5/VDF/Assignments/Assignment_2/icarus_codes$ qtkwave fsm.vcd
GTKWave Analyzer v3.3.104 (w)1999-2020 BSI
(gtkwave:205294): dconf-WARNING **: 15:25:50.655: failed to commit changes to dconf: Could not connect: No such fi
le or directory
[0] start time.
[834] end time.
```

#### Q1) Snippet of the log files to prove that I have indeed run the tools



```
Shivam@Shivam-Shukla: /mn × + v
shivam@Shivam-Shukla:/mnt/d/Sem_5/VDF/Assignments/Assignment_2/icarus_codes$ covered score -t fsm_tb -v fsm_tb.v
-v fsm.v -vcd fsm.vcd -o fsm.cdd
Covered covered-20090802 -- Verilog Code Coverage Utility
Written by Trevor Williams (phase1geo@gmail.com)
Copyright 2006-2010
Freely distributable under the GPL license
Reading design...
Parsing file 'fsm_tb.v'
Parsing file 'fsm.v'
Checking for race conditions...
Scoring VCD dumpfile fsm.vcd...
*** Scoring completed successfully! ***
Dynamic memory allocated:
                           408677 bytes
shivam@Shivam-Shukla:/mnt/d/Sem_5/VDF/Assignments/Assignment_2/icarus_codes$ covered report -d v fsm.cdd
Covered covered-20090802 -- Verilog Code Coverage Utility
Written by Trevor Williams (phase1geo@gmail.com)
Copyright 2006-2010
Freely distributable under the GPL license
```

#### Q1) Snippet of the log files to prove that I have indeed run the tools



```
shivam@Shivam-Shukla: /mn × + v
shivam@Shivam-Shukla:/mnt/d/Sem_5/VDF/Assignments/Assignment_2/icarus_codes$ cd ../yosys_codes/
shivam@Shivam-Shukla:/mnt/d/Sem_5/VDF/Assignments/Assignment_2/vosys_codes$ vosys
    yosys -- Yosys Open SYnthesis Suite
    Copyright (C) 2012 - 2019 Clifford Wolf <clifford@clifford.at>
    Permission to use, copy, modify, and/or distribute this software for any
    purpose with or without fee is hereby granted, provided that the above
    copyright notice and this permission notice appear in all copies.
    THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
    WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
    MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
    ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
    WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
    ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
    OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 Yosys 0.9 (git shal 1979e0b)
vosys> script vosys_commands.tcl
-- Executing script file 'yosys_commands.tcl' --
1. Executing Verilog-2005 frontend: fsm.v
Parsing Verilog input from 'fsm.v' to AST representation.
Generating RTLIL representation for module '\fsm'.
```

#### Q2 (a) Verilog model using binary encoding



```
module fsm (
input clk,
                 //Clock signal
input reset,
                  //Reset signal
input [1:0] in, //2-bit input signal
output reg [1:0] out //2-bit output signal
);
//States
parameter S1 = 3'b000;
parameter S2 = 3'b001;
parameter S3 = 3'b010;
parameter S4 = 3'b011;
parameter S5 = 3'b100;
//State reg and next state reg
reg [2:0] state reg, next reg;
```

```
//State transition
always @(posedge clk or posedge reset)
begin
if (reset) begin
state reg <= S1; //Reset to initial state
S1
end else begin
state reg <= next reg; //Transition to
next state
end
end
always @(*) begin
//Default values
next reg=state reg;
out=2'b00;
```

```
//Switch cases for states and outputs
case (state reg)
S1: begin
case (in)
2'b00: begin
out=2'b01;
next reg=S1;
end
2'b01: begin
out=2'b00;
next reg=S1;
end
2'b10: begin
out=2'b01;
next reg=S2;
end
```

### Q2 (a) Verilog model using binary encoding



```
2'b11: begin
                                       2'b11: begin
                                                                                 2'b11: begin
out=2'b00;
                                       out=2'b10;
                                                                                 out=2'b00;
next reg=S3;
                                       next reg=S4;
                                                                                 next reg=S4;
end
                                       end
                                                                                 end
endcase
                                       endcase
                                                                                 endcase
end
                                       end
                                                                                 end
S2: begin
                                       S3: begin
                                                                                 S4: begin
case (in)
                                       case (in)
                                                                                 case (in)
2'b00: begin
                                       2'b00: begin
                                                                                 2'b00: begin
out=2'b11;
                                       out=2'b00;
                                                                                 out=2'b11;
next reg=S1;
                                       next reg=S1;
                                                                                 next reg=S3;
end
                                       end
                                                                                 end
2'b01: begin
                                      2'b01: begin
                                                                                 2'b01: begin
out=2'b11;
                                       out=2'b01;
                                                                                 out=2'b01;
next reg=S3;
                                       next reg=S5;
                                                                                 next reg=S5;
end
                                       end
                                                                                 end
2'b10: begin
                                       2'b10: begin
                                                                                 2'b10: begin
out=2'b10;
                                       out=2'b00;
                                                                                 out=2'b01;
next reg=S2;
                                       next reg=S3;
                                                                                 next reg=S5;
end
                                       end
                                                                                 end
```

#### Q2 (a) Verilog model using binary encoding



```
2'b11: begin
out=2'b11;
next reg=S4;
end
endcase
end
S5: begin
case (in)
2'b00: begin
out=2'b00;
next reg=S5;
end
2'b01: begin
out=2'b00;
next reg=S1;
end
```

```
2'b10: begin
out=2'b10;
next reg=S1;
end
2'b11: begin
out=2'b11;
next reg=S5;
end
endcase
end
endcase
end
endmodule
```

#### Q2 (b) The testbench to test the Verilog model.

end



```
module fsm tb;
                                                //Tests
                                                                                                 #20 in=2'b11; //S4
reg clk;
                                                initial
                                                                                                 #20 in=2'b11; //S4
reg reset;
                                                begin
                                                                                                 #20 in=2'b10; //S5
reg [1:0] in;
                                                                                                 #20 in=2'b00; //S5
wire [1:0] out;
                                                $dumpfile("fsm.vcd");
                                                                            //Dumpfile
                                                                                                 #20 in=2'b11; //S5
                                                $dumpvars(0,fsm tb);
                                                                                                 #20 in=2'b01; //S1
//Instantiate our verilog top level fsm
fsm dut(
                                                //Defining reset and in
                                                                                                 reset=1;
.clk(clk),
                                                reset = 1; in = 2'b00;
                                                                                                 \#0.5 \text{ reset=0};
.reset(reset),
                                                #20 \text{ reset} = 0;
.in(in),
                                                                                                 #20 in=2'b11; //S3
.out(out)
                                                //All possible paths are tested by input
                                                                                                 #20 in=2'b00; //S1
);
                                                combinations
                                                                                                 #20 in=2'b11; //S3
                                                #20 in=2'b00; //S1
                                                                                                 #20 in=2'b01; //S5
//Generating clk
                                                #20 in=2'b01; //S1
                                                                                                 #20 in=2'b10; //S1
initial
                                                #20 in=2'b10; //S2
begin
                                                #20 in=2'b10; //S2
                                                                                                 reset=1;
clk=0;
                                                #20 in=2'b01; //S3
                                                                                                 \#0.5 \text{ reset=0};
forever \#10 \text{ clk} = \sim \text{clk}; //To toggle clk
                                                #20 in=2'b10; //S3
```

#### Q2 (b) The testbench to test the Verilog model.



```
#20 in=2'b10; //S2
#20 in=2'b00; //S1
#20 in=2'b11; //S3
#20 in=2'b01; //S5
#20 in=2'b01; //S1
reset=1;
\#0.5 \text{ reset=0};
#20 in=2'b10; //S2
#20 in=2'b10; //S2
#20 in=2'b11; //S4
#20 in=2'b00; //S3
#20 in=2'b11; //S4
#20 in=2'b01; //S5
#20 in=2'b10; //S1
```

```
reset=1;
#0.5 reset=0:
#20 in=2'b10;
#20 in=2'b10;
#20 in=2'b01;
#20 in=2'b00;
#20 in=2'b01;
#20 in=2'b11;
#20 in=2'b11;
#20 in=2'b00:
#20 in=2'b01;
#20 in=2'b00;
#20 in=2'b01;
//End simulation
#10 $finish;
end
```

```
//signal verification
always @(posedge clk or posedge reset)
begin
if (reset) begin
end else begin
end
end
end
endmodule
```

PS: I've combined the code snippets in sequence, with each section following after the white rectangles. Can also find the code here

#### Q2 (c) The simulation output waveform





The waveform shows the FSM simulation with clock (clk), reset (reset), input (in), and output (out).

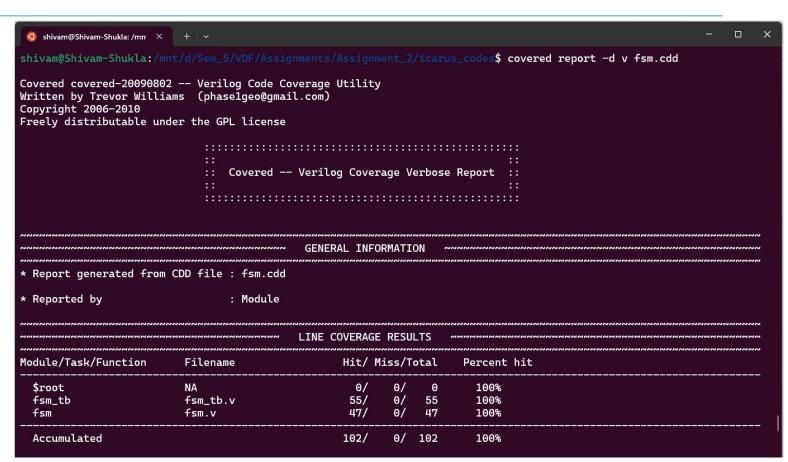
- Clock (clk): Toggles regularly, driving state transitions.
- Reset (reset): When high, the FSM resets to the initial state (S1), and the output goes to its default.
- Input (in): The 2-bit input (00, 01, 10, 11) changes, causing the FSM to move between states.
- Output (out): Changes as per FSM logic, producing the correct output for each state and input.

The simulation output waveform confirms correct transitions and outputs, matching the Verilog FSM design.

#### Q2 (d) Line Coverage report for the above testbench.



The testbench and FSM design achieved 100% line coverage, meaning every line of code was executed in the simulation. This confirms that the testbench fully tests all parts of the FSM design.



#### Q2 (e) Code synthesis using YOSYS



Synthesized the FSM code using YOSYS and NanGate\_45nm\_OCL\_v2010\_12 technology library.

The statistics shows the hardware structure, including:

- Wires: 36 wires and 42 wire bits connect the logic elements.
- Logic Cells: 35 cells, including inverters, NOR, NAND, AOI gates, and 3 flip-flops for storing state.
- Area: The total area is 49.21, showing moderate complexity.

The FSM uses basic gates and flip-flops efficiently, with 3 bits representing the state. The logic is optimized, balancing complexity and area.

```
shivam@Shivam-Shukla: /r ×
                                          8. Printing statistics.
=== fsm ===
   Number of wires:
                                      36
                                      42
   Number of wire bits:
   Number of public wires:
                                       6
   Number of public wire bits:
                                      12
   Number of memories:
                                       0
   Number of memory bits:
                                       0
   Number of processes:
                                      35
   Number of cells:
     A0I211 X1
     A0I21_X1
     A0I221 X1
     A0I22 X1
     DFFR_X1
     INV X1
     MUX2 X1
     NAND2_X1
     NAND3_X1
     NAND4_X1
     NOR2 X1
     NOR3 X1
     NOR4_X1
     OAI21 X1
     OR3_X1
     XNOR2_X1
   Chip area for module '\fsm': 49.210000
9. Executing Verilog backend.
Dumping module '\fsm'.
```



```
synth exam...
                                          \equiv
                                  Save
1 /* Generated by Yosys 0.9 (git shal 1979e0b) */
3 module fsm(clk, reset, in, out);
   wire 00;
   wire 01;
   wire 02;
   wire 03;
    wire 04;
   wire 05;
         06
   wire 07;
   wire 08:
    wire 09;
    wire 10 :
   wire 11 :
    wire 12 :
   wire 13;
   wire 14;
   wire 15 :
   wire 16 :
   wire 17 ;
   wire 18 :
   wire 19 :
   wire 20
   wire 21 :
   wire 22 :
   wire 23 :
   wire 24 :
   wire 25
   wire 26 :
   wire 27 :
   wire 28 :
   wire 29 :
   input clk:
   input [1:0] in;
   wire [2:0] next reg;
   output [1:0] out:
       Verilog ▼ Tab Width: 8 ▼
                               Ln 37, Col 20
```

```
synth exam...
             0
                                    Save
                                           ≡
  Open
     input reset:
     wire [2:0] state reg;
     INV X1 30 (
       .A(reset).
42
       .ZN( 02 )
43
    ):
    INV X1 31 (
       .A(state reg[1]),
       .ZN( 03 )
47
    );
     INV X1 32 (
       .A(state reg[0]),
50
       .ZN( 04 )
51
    );
52
    INV X1 33 (
53
       .A(in[0]).
54
       .ZN( 05 )
55
    ):
     INV X1 34 (
57
       .A(in[1]),
58
       .ZN( 06 )
59
    NOR3 X1 35 (
      .A1( 03 ),
      .A2(state reg[0]),
63
       .A3(state reg[2]),
64
       .ZN( 07 )
65
    ):
     NOR2 X1 36 (
       .Al(state reg[1]).
68
       .A2(state reg[0]),
69
       .ZN( 08 )
71
     NOR3 X1 37 (
72
       .A1( 03 ),
73
       .A2( 04 ),
       .A3(state reg[2]),
       7N 00
Bracket... Verilog ▼ Tab Width: 8 ▼
                                 Ln 75. Col 14
```

```
synth_exam...
         •
              \oplus
                                     Save
                                             \equiv
  Open
 75
        .ZN( 09 )
 76
     OR3 X1 38 (
       .A1( 03 ),
       .A2( 04 ),
       .A3(state reg[2]),
 81
        .ZN( 10 )
 82
 83
     XNOR2 X1 39 (
       .A(in[0]),
       .B(in[1]),
       .ZN( 11 )
 87
     );
     NOR3 X1 40 (
       .Al(state reg[1]),
       .A2( 04 ),
       .A3(state reg[2]),
 92
       .ZN( 12 )
 93
     A0I22 X1 41 (
       .A1(state reg[1]),
       .A2(state reg[2]),
       .B1( 09 ).
       .B2( 11 ),
 99
        .ZN( 13 )
100
101
     A0I211 X1 42 (
102
        .A(state reg[0]),
103
       .B(state reg[2]),
104
       .C1( 05 ),
105
       .C2( 03 ),
106
       .ZN( 14 )
107
108
     A0I22 X1 43 (
       .A1(in[0]),
109
110
        .A2( 12 ),
111
        .B1( 14 ),
         Verilog ▼ Tab Width: 8 ▼ Ln 111, Col 15
```



```
synth exam...
  Open
                                     Save
        .B2(in[1]),
112
113
        .ZN(15)
114
     );
     NAND2 X1 44 (
116
        .A1( 13 ),
117
       .A2( 15 ),
118
        .ZN(next reg[1])
119
     ):
120
     NOR4 X1 45 (
121
       .Al(state reg[1]),
122
       .A2(state reg[0]),
       .A3(state reg[2]),
124
       .A4(in[0]),
125
       .ZN( 16 )
126
127
     A0I211 X1 46 (
128
       .A(state reg[1]),
       .B(state reg[2]),
130
       .C1(in[0]),
       .C2( 04 ),
132
        .ZN( 17 )
133
     ):
     NOR4 X1 47 (
       .A1( 03 ),
136
       .A2(state reg[2]),
137
       .A3( 05 ),
138
        .A4( 06 ),
139
        .ZN( 18 )
140
141
     A0I221 X1 48 (
142
       .A( 18 ).
143
       .B1( 17 ).
144
       .B2(in[1]).
145
       .Cl(state reg[0]),
146
       .C2(state reg[2]),
147
       .ZN( 19 )
148 );
         Verilog ▼ Tab Width: 8 ▼
                                 Ln 111. Col 15
```

```
synth_exam...
              1
                                     Save
                                            \equiv
                   /mnt/d/Sem 5/V..
149
     INV X1 49 (
150
        .A( 19 ),
151
        .ZN(next reg[0])
152
     );
153
     NAND3 X1 50 (
       .A1(in[0]),
155
       .A2( 06 ),
156
       .A3( 07 ),
157
       .ZN( 20 )
158
159
     NAND2 X1 51 (
160
       .Al(state reg[2]),
161
       .A2( 08 ).
162
        .ZN(21)
163
164
     NAND4 X1 52 (
165
       .Al(state reg[2]),
166
        .A2(in[0]),
167
        .A3(in[1]),
168
       .A4( 08 ),
       .ZN(22)
169
170
     );
171
     A0I21 X1 53 (
172
       .A( 16 ).
173
       .B1( 12 ).
174
       .B2( 06 ).
175
        .ZN(23)
176
     NAND4 X1 54 (
178
       .A1( 10 ),
       .A2( 20 ),
179
180
       .A3(22),
181
       .A4(23),
182
        .ZN(out[0])
183
184
     MUX2 X1 55 (
185
        .A( 10 ),
         Verilog ▼ Tab Width: 8 ▼
                               Ln 111. Col 15
```

```
synth exam...
         •
              \oplus
                                             \equiv
                                     Save
                   /mnt/d/Sem 5/V...
185
        .A( 10 ).
186
        .B( 21 ),
187
       .S( 11 ),
188
       .Z(24)
189
190
     0AI21 X1 56
       .A(state reg[2]),
       .B1(state reg[0]),
       .B2(state_reg[1]),
194
       .ZN(25)
195
     );
196
     NAND3 X1 57 (
197
       .A1( 20 ),
198
       .A2(24).
199
       .A3(25),
       .ZN(next reg[2])
201
202
     A0I21 X1 58 (
203
       .A( 12 ).
204
       .B1( 11 ),
205
       .B2( 09 ),
206
        .ZN(26)
207
208
     OAI21 X1 59 (
       .A( 26 ),
210
       .B1(21).
211
       .B2( 06 ).
212
        .ZN(out[1])
213
     ):
214
     INV X1 60 (
215
       .A(reset).
216
       .ZN( 00 )
217
218
     INV X1 61 (
219
       .A(reset).
220
        .ZN( 01 )
221 );
         Verilog ▼ Tab Width: 8 ▼
                                 Ln 111, Col 15
```



```
synth exam...
                                   Save
207
     OAI21 X1 59
       .A( 26 ).
       .B1( 21 ).
       .B2( 06 ).
212
       .ZN(out[1])
213
214 INV X1 60 (
       .A(reset).
216
       .ZN( 00 )
217 ):
     INV X1 61 (
       .A(reset).
220
       .ZN( 01 )
221 ):
222
    DFFR X1 62 (
223
       .CK(clk).
      .D(next reg[0]).
225
       .Q(state reg[0]),
       .QN( 28 ),
227
       .RN( 01 )
228
229
    DFFR X1 63 (
       .CK(clk).
       .D(next reg[1]),
       .Q(state reg[1]),
233
       .QN( 27 ),
234
       .RN( 00 )
235
     ):
     DFFR X1 64 (
       .CK(clk).
238
       .D(next reg[2]),
       .Q(state reg[2]),
240
       .QN(29),
241
       .RN( 02 )
242 ):
243 endmodule
         Verilog ▼ Tab Width: 8 ▼ Ln 111, Col 15
```

The netlist is an optimized version of the FSM that translates FSM into basic gates and flip-flops, making it ready for hardware implementation.

- Key Components: Inputs are clk, reset, and a 2-bit input (in), while the output is a 2-bit signal (out). The FSM transitions between states based on the state register and inputs.
- Flip-Flops (DFFR\_X1): Three D flip-flops store the FSM state, updating on the clock's rising edge and resetting when required.
- Logic Gates: Combinational gates like inverters, NORs, AOI, and multiplexers handle state transitions and output generation.

The result confirms that the design is efficient and optimized for further use in hardware.

#### Q2 (f) Re-write the Verilog code using one-hot encoding



```
module fsm opt (
input clk,
                 //Clock signal
input reset,
                 //Reset signal
input [1:0] in, //2-bit input signal
output reg [1:0] out //2-bit output signal
//States bits increased
parameter S1 = 5'b00001;
parameter S2 = 5'b00010;
parameter S3 = 5'b00100;
parameter S4 = 5'b01000;
parameter S5 = 5'b10000;
//State reg and next state reg
reg [4:0] state reg, next reg;
//bits increased to 5
```

```
//State transition
always @(posedge clk or posedge reset)
begin
if (reset) begin
state reg <= S1; //Reset to initial state
S1
end else begin
state reg <= next reg; //Transition to
next state
end
end
always @(*) begin
//Default values
next reg=state reg;
out=2'b00;
```

```
//Switch cases for states and outputs
case (state reg)
S1: begin
case (in)
2'b00: begin
out=2'b01;
next reg=S1;
end
2'b01: begin
out=2'b00;
next reg=S1;
end
2'b10: begin
out=2'b01;
next reg=S2;
end
```

#### Q2 (f) Re-write the Verilog code using one-hot encoding



```
2'b11: begin
                                       2'b11: begin
                                                                                 2'b11: begin
out=2'b00;
                                       out=2'b10;
                                                                                 out=2'b00;
next reg=S3;
                                       next reg=S4;
                                                                                 next reg=S4;
end
                                       end
                                                                                 end
endcase
                                                                                 endcase
                                       endcase
end
                                       end
                                                                                 end
S2: begin
                                       S3: begin
                                                                                 S4: begin
case (in)
                                       case (in)
                                                                                 case (in)
2'b00: begin
                                       2'b00: begin
                                                                                 2'b00: begin
out=2'b11;
                                       out=2'b00;
                                                                                 out=2'b11;
next reg=S1;
                                       next reg=S1;
                                                                                 next reg=S3;
end
                                      end
                                                                                 end
2'b01: begin
                                      2'b01: begin
                                                                                 2'b01: begin
out=2'b11;
                                       out=2'b01;
                                                                                 out=2'b01;
next reg=S3;
                                       next reg=S5;
                                                                                 next reg=S5;
end
                                       end
                                                                                 end
2'b10: begin
                                       2'b10: begin
                                                                                 2'b10: begin
out=2'b10;
                                       out=2'b00;
                                                                                 out=2'b01;
next reg=S2;
                                       next reg=S3;
                                                                                 next reg=S5;
end
                                       end
                                                                                 end
```

#### Q2 (f) Re-write the Verilog code using one-hot encoding



```
2'b11: begin
out=2'b11;
next reg=S4;
end
endcase
end
S5: begin
case (in)
2'b00: begin
out=2'b00;
next reg=S5;
end
2'b01: begin
out=2'b00;
next reg=S1;
end
```

```
2'b10: begin
out=2'b10;
next reg=S1;
end
2'b11: begin
out=2'b11;
next reg=S5;
end
endcase
end
endcase
end
endmodule
```

#### Q2 (f) One-hot encoding code synthesis using YOSYS



Synthesized the FSM using YOSYS with one-hot encoding and NanGate\_45nm\_OCL\_v2010\_12 technology library.

The synthesized netlist reveals the following hardware structure:

- Wires: 51 wires and 61 wire bits connect various logic elements.
- Logic Cells: 52 cells, including inverters, NOR, NAND, AOI gates, and 5 flip-flops to store the state.
- Area: The total area is 76.874, indicating an increase in complexity compared to binary encoding.

The FSM design efficiently utilizes basic gates and flip-flops with one-hot encoding. Each state is represented with a unique bit, optimizing state transitions while slightly increasing area usage.

```
🧿 shivam@Shivam-Shukla: /mn 🗀 🗙
=== fsm_opt ===
   Number of wires:
                                      51
   Number of wire bits:
                                      61
   Number of public wires:
   Number of public wire bits:
   Number of memories:
   Number of memory bits:
   Number of processes:
   Number of cells:
                                      52
     A0I22 X1
     DFFR X1
     DFFS X1
     INV X1
                                      12
     MUX2_X1
     NAND3_X1
     NAND4_X1
     NOR2_X1
     NOR3_X1
     NOR4 X1
     OAI211 X1
     OAI21 X1
     OAI221 X1
     OR2 X1
     OR3 X1
     OR4_X1
     XOR2_X1
   Chip area for module '\fsm_opt': 76.874000
9. Executing Verilog backend.
Dumping module `\fsm_opt'.
```



```
synth_examp...
                                 Save
 1 /* Generated by Yosys 0.9 (git shal 1979e0b) */
3 module fsm opt(clk, reset, in, out);
    wire 00;
   wire 01;
        02;
    wire
         03;
    wire
        04;
        05 :
    wire 06:
    wire 07 :
    wire 08;
    wire 10;
    wire 11 :
    wire 12 :
    wire 13 :
    wire 14 :
    wire 15 ;
    wire 16;
    wire 17 ;
    wire 18;
    wire 19;
    wire 20;
    wire 21 :
    wire 22 :
    wire 23;
    wire 24;
    wire 25;
    wire 26;
    wire
        27 :
        28 :
    wire 29 :
    wire 30 :
    wire 31;
    wire 32;
   wire 33 ;
38 wire 34:
        Verilog ▼ Tab Width: 8 ▼
                               Ln 1, Col 1
```

```
synth_examp...
                                    Save
 Open
                  /mnt/d/Sem 5/VD.
    wire 35 :
          36 :
    wire
    wire 37 :
    wire 38;
    wire 39 ;
    wire
    wire 41 :
    wire 42 ;
    wire 43;
    wire 44;
    input clk;
    input [1:0] in:
    wire [4:0] next reg;
    output [1:0] out;
    input reset;
    wire [4:0] state reg;
    INV X1 45 (
      .A(reset),
57
      .ZN( 04 )
58
   ):
    INV X1 46 (
      .A(state reg[3]),
61
      .ZN( 05 )
62
   );
    INV X1 47 (
      .A(state reg[2]),
65
      .ZN( 06 )
66
    INV X1 48 (
      .A(state reg[1]),
      .ZN( 07 )
70
   ):
    INV X1 49 (
72
      .A(state reg[0]),
73
      .ZN( 08 )
74
   ):
75
   INV X1 50 (
       Aletate real(11)
        Verilog ▼ Tab Width: 8 ▼
                                  Ln 1, Col 1
                                                 INS
```

```
synth examp...
                                             =
                                      Save
     INV X1 50 (
       .A(state reg[4]),
 77
        .ZN( 09 )
 78
     ):
     INV X1 51 (
       .A(in[0]),
       .ZN( 10 )
     ):
     INV X1 52 (
       .A(in[1]).
 85
       .ZN( 11 )
     );
 87
     OR2 X1 53 (
       .A1(state reg[1]),
       .A2(state reg[0]),
       .ZN( 12 )
 91
     );
     NOR4 X1 54 (
       .A1( 05 ),
       .A2(state reg[2]),
       .A3(state reg[4]),
       .A4( 12 ),
       .ZN( 13 )
 98
     OR4 X1 55 (
       .A1( 05 ),
       .A2(state reg[2]),
       .A3(state reg[4]),
102
103
       .A4( 12 ),
104
       .ZN( 14 )
105
106
     NOR4 X1 56 (
107
       .Al(state reg[3]),
108
       .A2( 06 ),
109
       .A3(state reg[4]),
110
        .A4( 12 ),
111
        .ZN( 15 )
112
          Verilog ▼ Tab Width: 8 ▼
                                    Ln 1, Col 1
```



```
synth examp...
                                              ≡
  Open
                                      Save
                    /mnt/d/Sem 5/VD..
112 ):
113
     OR4 X1 57 (
114
       .Al(state reg[3]),
115
       .A2( 06 ),
116
       .A3(state reg[4]),
117
       .A4( 12 ),
118
       .ZN( 16 )
119
     ):
     OR2 X1 58 (
       .Al(state reg[3]),
121
122
       .A2(state reg[2]),
123
       .ZN(17)
124
125
     OR3 X1 59 (
126
       .Al(state reg[3]),
127
       .A2(state reg[2]),
128
       .A3(state reg[0]).
129
       .ZN( 18 )
130
     NOR3 X1 60 (
132
       .A1( 07 ),
       .A2(state reg[4]),
134
       .A3( 18 ).
135
       .ZN( 19 )
136
     OR3 X1 61 (
138
       .A1( 07 ).
139
       .A2(state reg[4]),
140
       .A3( 18 ),
141
       .ZN( 20 )
142
143
     NAND3 X1 62 (
144
       .A1( 14 ).
145
       .A2( 16 ).
146
       .A3( 20 ).
147
       .ZN( 21 )
148 );
    MOD2 V1 63 /
          Verilog ▼ Tab Width: 8 ▼
                                    Ln 1, Col 1
                                                   INS
```

```
synth examp...
                                              \equiv
              \oplus
                                      Save
                   /mnt/d/Sem 5/VD...
     NOR2 X1 63
150
        .A1( 10 ),
151
        .A2( 11 ),
152
        .ZN( 22 )
153
      );
154
      NOR3 X1 64
        .A1( 09 ).
156
        .A2( 12 ).
       .A3( 17 ),
157
158
        .ZN(23)
159
      );
      OR3 X1 65 (
        .A1( 09 ),
161
162
        .A2( 12 ),
163
        .A3( 17 ),
164
        .ZN(24)
165
166
      NAND4 X1 66 (
167
        .A1( 14 ),
        .A2(16),
169
        .A3( 20 ),
170
        .A4(24),
171
        .ZN(25)
172
173
      MUX2 X1 67 (
174
        .A(state reg[3]).
175
        .B( 22 ).
176
        .S( 21 ).
177
        .Z(next reg[3])
178
      NOR2 X1 68
180
        .A1(in[0]),
181
        .A2(in[1]),
182
        .ZN( 26 )
183
      ):
184
      NOR2 X1 69
        .A1(in[0]),
186
        .A2( 11 ),
          Verilog ▼ Tab Width: 8 ▼
                                    Ln 1, Col 1
```

```
synth_examp...
              \oplus
                                              \equiv
         •
                                      Save
  Open
                   /mnt/d/Sem 5/VD..
187
        .ZN(27)
188
189
     A0I22 X1 70 (
        .A1( 13 ).
191
       .A2( 26 ),
192
       .B1(27),
193
        .B2( 15 ),
194
       .ZN( 28 )
195
     NOR4 X1 71 (
197
       .Al(state reg[1]).
        .A2( 08 ),
199
       .A3(state reg[4]),
        .A4( 17 ),
201
        .ZN(29)
202
     NOR4 X1 72 (
203
204
        .A1( 07 ),
       .A2(state reg[4]).
        .A3(in[1]),
207
        .A4( 18 ),
        .ZN(30)
209
210
     A0I22 X1 73 (
211
       .A1( 22 ).
212
        .A2(29).
213
       .B1( 30 ).
214
        .B2(in[0]),
215
        .ZN(31)
216
217
     OAI211 X1 74 (
218
       .A( 28 ),
       .B( 31 ),
       .C1( 06 ).
221
       .C2(25),
222
        .ZN(next reg[2])
223
     );
     OAT21 X1 75 (
          Verilog ▼ Tab Width: 8 ▼
                                    Ln 1, Col 1
                                                   INS
```



```
synth_examp...
              \oplus
                                     Save
                   /mnt/d/Sem 5/VD...
     OAI21 X1 75 (
       .A( 27 ),
226
       .B1(29),
       .B2(19),
228
       .ZN(32)
229
230
     OAI21 X1 76 (
       .A( 32 ),
232
       .B1(25).
       .B2( 07 ),
234
       .ZN(next reg[1])
235
     );
     OAI21 X1 77 (
       .A( 26 ).
238
       .B1( 19 ).
239
       .B2( 15 ),
240
       .ZN(33)
241
     XOR2 X1 78 (
       .A(in[0]),
243
244
       .B(in[1]),
245
       .Z(34)
246
     A0I22 X1 79 (
248
       .A1( 11 ).
249
       .A2( 29 ).
250
       .B1( 34 ),
251
       .B2(23),
252
       .ZN(35)
253
     OAI211 X1 80 (
255
       .A( 33 ),
       .B( 35 ),
257
       .C1( 08 ),
       .C2(29),
259
       .ZN(next reg[0])
    );
261 NOR2 X1 81 (
          Verilog ▼ Tab Width: 8 ▼
                                   Ln 1, Col 1
```

```
synth_examp...
                                           \equiv
                                    Save
                  /mnt/d/Sem 5/VD..
     NOR2 X1 81 (
261
262
       .A1( 13 ),
       .A2(30),
263
       .ZN(36)
264
265
     );
266
     NAND3 X1 82 (
267
       .A1(in[0]),
268
       .A2( 11 ).
269
       .A3( 15 ),
270
       .ZN(37)
271
272
     A0I22 X1 83 (
       .A1( 22 ),
273
274
       .A2(23),
275
       .B1(29),
276
       .B2( 10 ),
277
       .ZN(38)
278
     ):
279
     NAND3 X1 84 (
280
       .A1( 36 ),
281
       .A2(37),
282
       .A3( 38 ).
283
       .ZN(out[0])
284
285
     MUX2 X1 85 (
       .A( 24 ),
286
287
       .B( 14 ),
288
       .5(34).
289
       .Z(39)
290
291
     OAI211 X1 86
292
       .A( 37 ),
293
       .B(39).
294
       .C1( 09 ).
295
       .C2( 25 ),
296
       .ZN(next reg[4])
297
     Verilog ▼ Tab Width: 8 ▼
                                  Ln 1, Col 1
                                                INS
```

```
synth examp...
              \oplus
                                              \equiv
                                      Save
                                                    ×
                   /mnt/d/Sem 5/VD...
     OAI221 X1 87 (
299
        .A( 20 ),
300
        .B1(24),
301
        .B2( 11 ),
302
        .C1(34),
303
        .C2( 14 ),
        .ZN(out[1])
305
     INV X1 88 (
307
       .A(reset),
       .ZN( 00 )
309
     ):
310
     INV X1 89 (
311
        .A(reset),
312
        .ZN( 01 )
313
314
     INV X1 90 (
315
       .A(reset).
316
        .ZN( 02 )
317
     );
318
     INV X1 91 (
319
       .A(reset),
320
        .ZN( 03 )
321
     ):
     DFFR X1 92 (
       .CK(clk).
324
       .D(next reg[3]),
325
       .Q(state reg[3]),
326
        .QN( 40 ),
327
        .RN( 00 )
328
     DFFR X1 93 (
329
330
        .CK(clk).
331
       .D(next reg[4]),
332
        .Q(state reg[4]),
333
        .QN( 44 ),
334
        .RN( 04 )
335
          Verilog ▼ Tab Width: 8 ▼
                                     Ln 1, Col 1
```



```
synth_examp...
                                              \equiv
                                      Save
        .ZN( 03 )
322
     DFFR X1 92
323
       .CK(clk),
       .D(next reg[3]),
325
       .Q(state reg[3]),
       .QN( 40 ),
327
       .RN( 00 )
     DFFR X1 93 (
       .CK(clk).
331
       .D(next reg[4]),
       .Q(state reg[4]),
333
       .ON( 44 ).
334
       .RN( 04 )
     DFFS X1 94 (
337
       .CK(clk).
338
       .D(next reg[0]),
       .Q(state reg[0]),
340
       .QN( 43 ),
341
       .SN( 03 )
     DFFR X1 95 (
344
       .CK(clk),
345
       .D(next reg[1]),
       .Q(state reg[1]),
        .QN( 42 ),
348
       .RN( 02 )
     DFFR X1 96 (
       .CK(clk).
352
        .D(next rea[2]).
353
        .Q(state reg[2]),
        .QN(41),
        .RN( 01 )
357 endmodule
          Verilog ▼ Tab Width: 8 ▼
                                    Ln 1, Col 1
```

The netlist is optimized for one-hot encoding, with five flip-flops representing the FSM states.

- Inputs: Clock (clk), reset (reset), and a 2-bit input (in).
- Outputs: A 2-bit output (out) driven by state transitions.
- Flip-Flops: Five D flip-flops store the FSM state, with each state represented by one bit in the state register.
- Logic Gates: Inverters, NOR, NAND, AOI gates, and multiplexers manage state transitions and output logic.

The design efficiently translates the FSM to hardware using simple combinational logic and flip-flops, balancing hardware complexity and performance.

#### Q2 (f) Comparison of One-Hot vs Binary Encoded FSM:



- Wires & Wire Bits: One-hot encoding uses 51 wires and 61 wire bits, whereas binary encoding uses 36 wires and 42 wire bits. One-hot requires more connections due to the increased number of states represented by individual bits.
- Logic Cells: One-hot uses 52 cells, while binary encoding uses 35. This indicates a higher complexity in one-hot encoding as each state has its own flip-flop.
- Flip-Flops: One-hot uses 5 flip-flops, representing each state individually, compared to 3 flip-flops in binary encoding, which requires fewer flip-flops due to compact state representation.
- Area: One-hot encoding consumes more area (76.874 vs 49.21) due to the increased hardware complexity of using more flip-flops and gates for each state.

#### Key Inference:

One-hot encoding simplifies state transition logic but increases hardware complexity and area, while binary encoding is more area-efficient, using fewer flip-flops at the cost of slightly more complex transition logic.

## Thank You



INDRAPRASTHA INSTITUTE of INFORMATION TECHNOLOGY **DELHI**