VDF Assignment 2

For FSM:

Step 1 : List Process Variable

States : {000,001,010,011,100}

Inputs : {00,01,10,11}

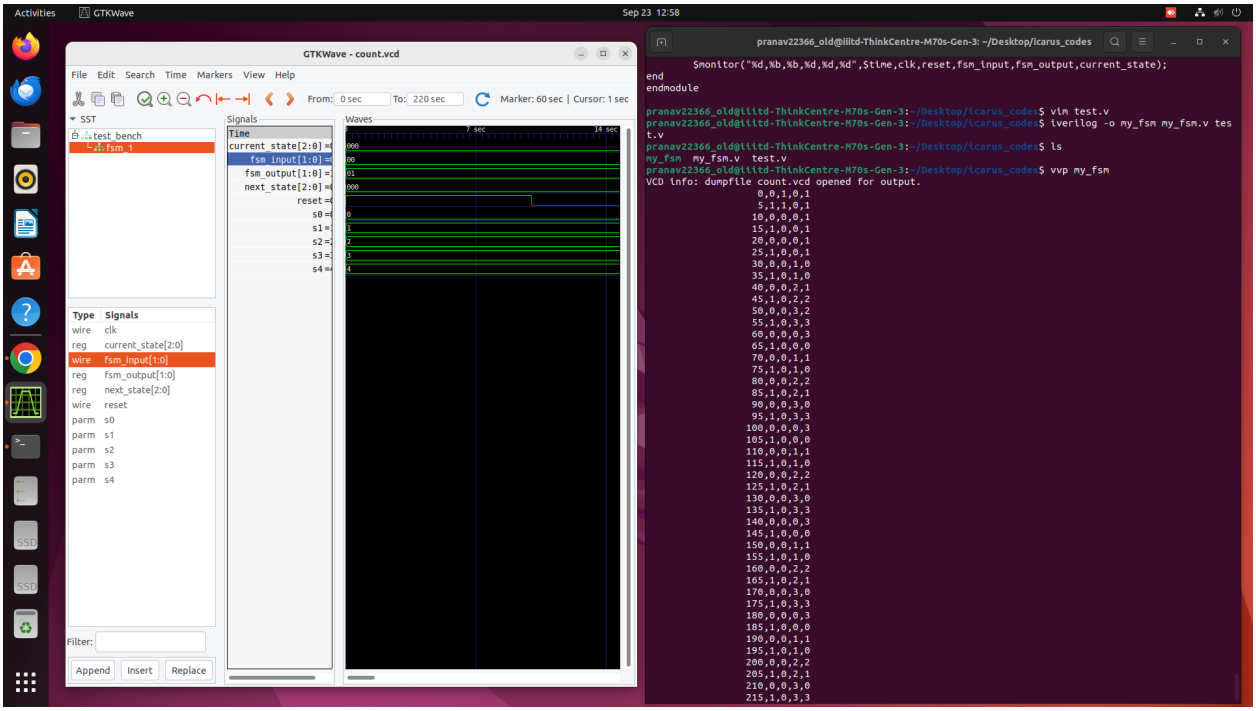
Outputs : {00,01,10,11}

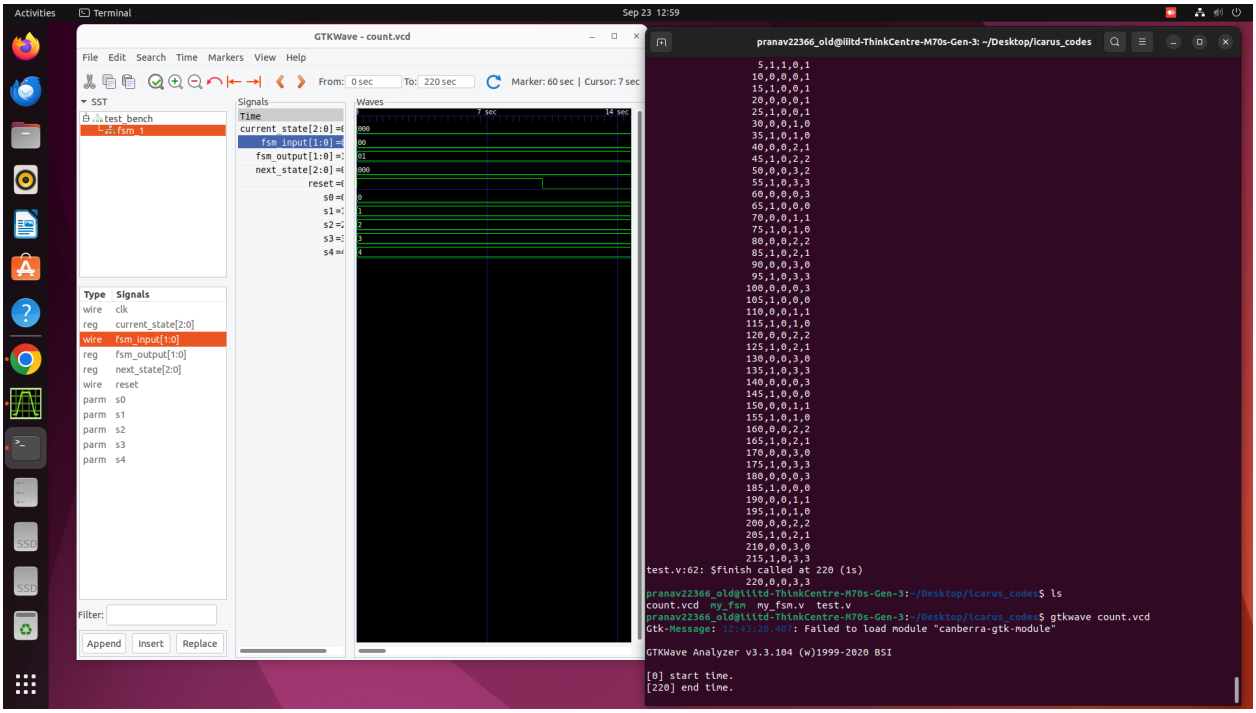
Step 2:State Transition Table

Current State	Input	Output	Next State
000	00	01	000
000	01	00	000
000	10	01	001
000	11	00	010
001	00	11	000
001	01	11	010
001	10	10	001
001	11	10	011
010	00	00	000
010	01	01	100
010	10	00	010
010	11	00	011
011	00	11	010
011	01	01	100
011	10	01	100
011	11	11	100
100	00	00	100
100	01	00	000

100	10	10	000
100	11	11	100

1.





Log : pranav22366_old in terminal

2.

(a)Verilog model:

```
// // we have states s1,s2,s3,s4,s5
// // States : {000,001,010,011,100}
module my_fsm(
    input wire clk,
    input wire reset,
    input wire [1:0] fsm_input,
    output reg [1:0] fsm_output
);

// State definitions
reg [2:0] current_state, next_state;
parameter s0 = 3'b000;
parameter s1 = 3'b001;
parameter s2 = 3'b010;
parameter s3 = 3'b011;
parameter s4 = 3'b100;

// Initialize current_state on reset
always @(posedge clk or posedge reset) begin
    if (reset) begin
        current_state <= s0; // Reset to initial state
    end else begin
        current_state <= next_state; // Update current state
    end
end

// Combinational block for state transition and output logic
always @(*) begin
    case (current_state)
        s0: begin
            if (fsm_input == 2'b00) begin
```

```
                next_state = s0;
                fsm_output = 2'b01;
            end else if (fsm_input == 2'b01) begin
                next_state = s0;
                fsm_output = 2'b00;
            end else if (fsm_input == 2'b10) begin
                next_state = s1;
                fsm_output = 2'b01;
            end else begin
                next_state = s3;
                fsm_output = 2'b00;
            end
        end
        s1: begin
            if (fsm_input == 2'b00) begin
                next_state = s0;
                fsm_output = 2'b11;
            end else if (fsm_input == 2'b10) begin
                next_state = s1;
                fsm_output = 2'b10;
            end else if (fsm_input == 2'b01) begin
                next_state = s2;
                fsm_output = 2'b11;
            end else begin
                next_state = s3;
                fsm_output = 2'b10;
            end
        end
        s2: begin
            if (fsm_input == 2'b00) begin
                next_state = s0;
                fsm_output = 2'b00;
            end else if (fsm_input == 2'b10) begin
                next_state = s2;
                fsm_output = 2'b00;
            end else if (fsm_input == 2'b01) begin
```

```
                next_state = s4;
                fsm_output = 2'b01;
            end else begin
                next_state = s3;
                fsm_output = 2'b00;
            end
        end
        s3: begin
            if (fsm_input == 2'b00) begin
                next_state = s2;
                fsm_output = 2'b11;
            end else if (fsm_input == 2'b10) begin
                next_state = s4;
                fsm_output = 2'b01;
            end else if (fsm_input == 2'b01) begin
                next_state = s4;
                fsm_output = 2'b01;
            end else begin
                next_state = s4;
                fsm_output = 2'b11;
            end
        end
        s4: begin
            if (fsm_input == 2'b00) begin
                next_state = s4;
                fsm_output = 2'b00;
            end else if (fsm_input == 2'b10) begin
                next_state = s0;
                fsm_output = 2'b10;
            end else if (fsm_input == 2'b01) begin
                next_state = s0;
                fsm_output = 2'b00;
            end else begin
                next_state = s4;
                fsm_output = 2'b11;
            end
```

```
        end
    endcase
end

endmodule
```

Binary encoding for the states:

State Transition Table

Current State	Input	Output	Next State
000	00	01	000
000	01	00	000
000	10	01	001
000	11	00	010
001	00	11	000
001	01	11	010
001	10	10	001
001	11	10	011
010	00	00	000
010	01	01	100
010	10	00	010
010	11	00	011
011	00	11	010
011	01	01	100
011	10	01	100
011	11	11	100
100	00	00	100
100	01	00	000
100	10	10	000
100	11	11	100

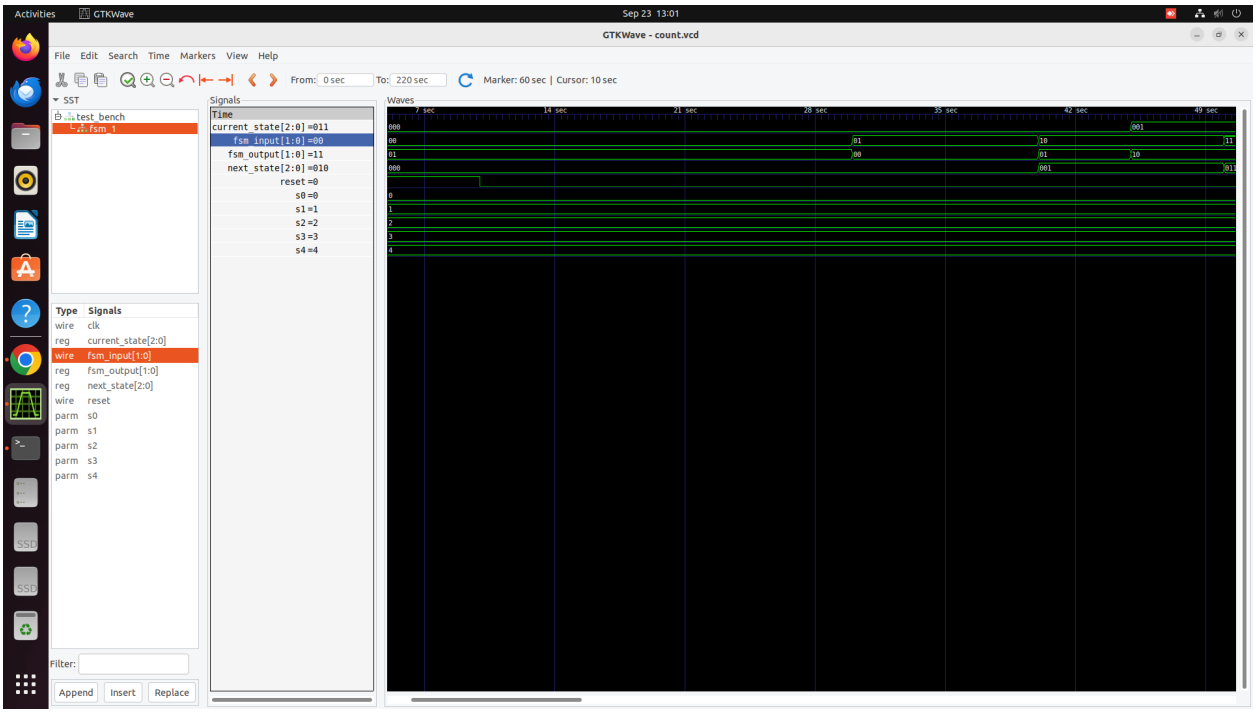
(b) Testbench:

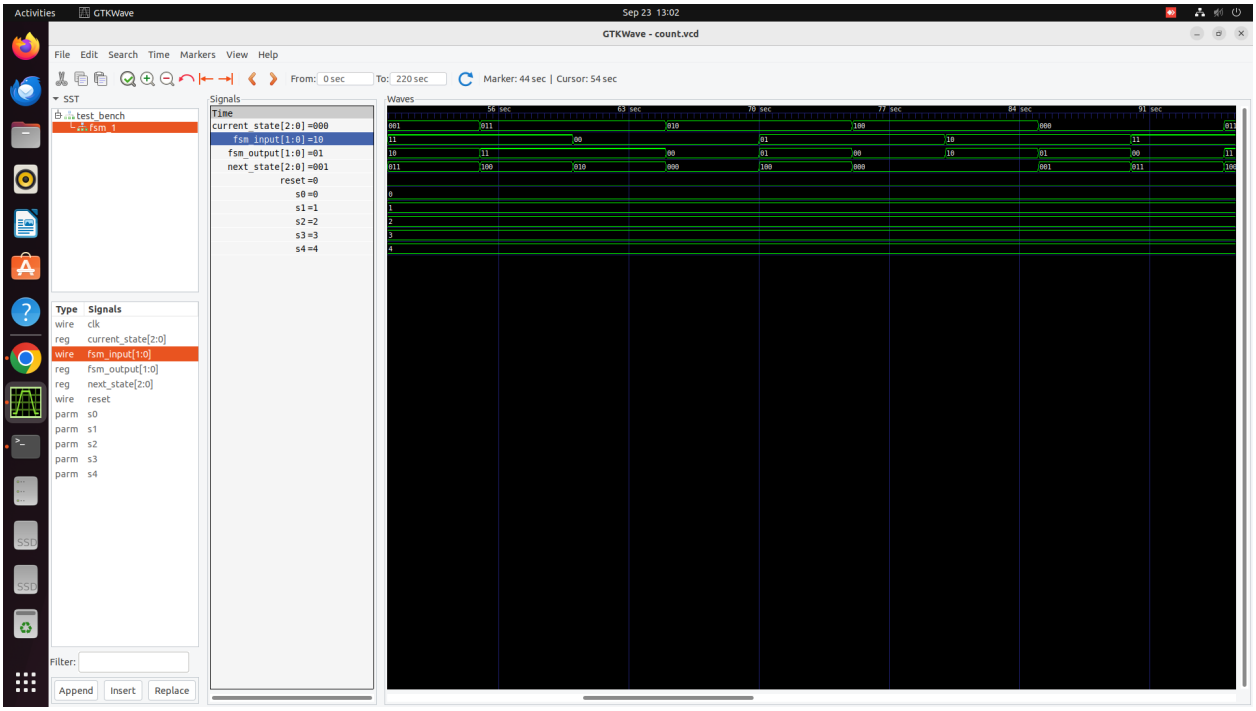
```
module test_bench;
    // Inputs and Outputs
    reg clk;
    reg reset;
    reg [1:0] fsm_input;
    wire [1:0] fsm_output;

    // Instantiate the FSM
    my_fsm fsm_1 (
        .clk(clk),
        .reset(reset),
        .fsm_input(fsm_input),
        .fsm_output(fsm_output)
    );

    // Generate the clock signal
    initial begin
        clk = 0;
        forever #5 clk = ~clk; // Toggle clock every 5 time unit
    end

    // Stimulus generation
    initial begin
        reset = 1; // Assert reset
        fsm_input = 2'b00; // Initial input

        #10; // Wait for some time
        reset = 0; // Deassert reset

        // Test various inputs
        #10 fsm_input = 2'b00; // Test input 00
        #10 fsm_input = 2'b01; // Test input 01
```

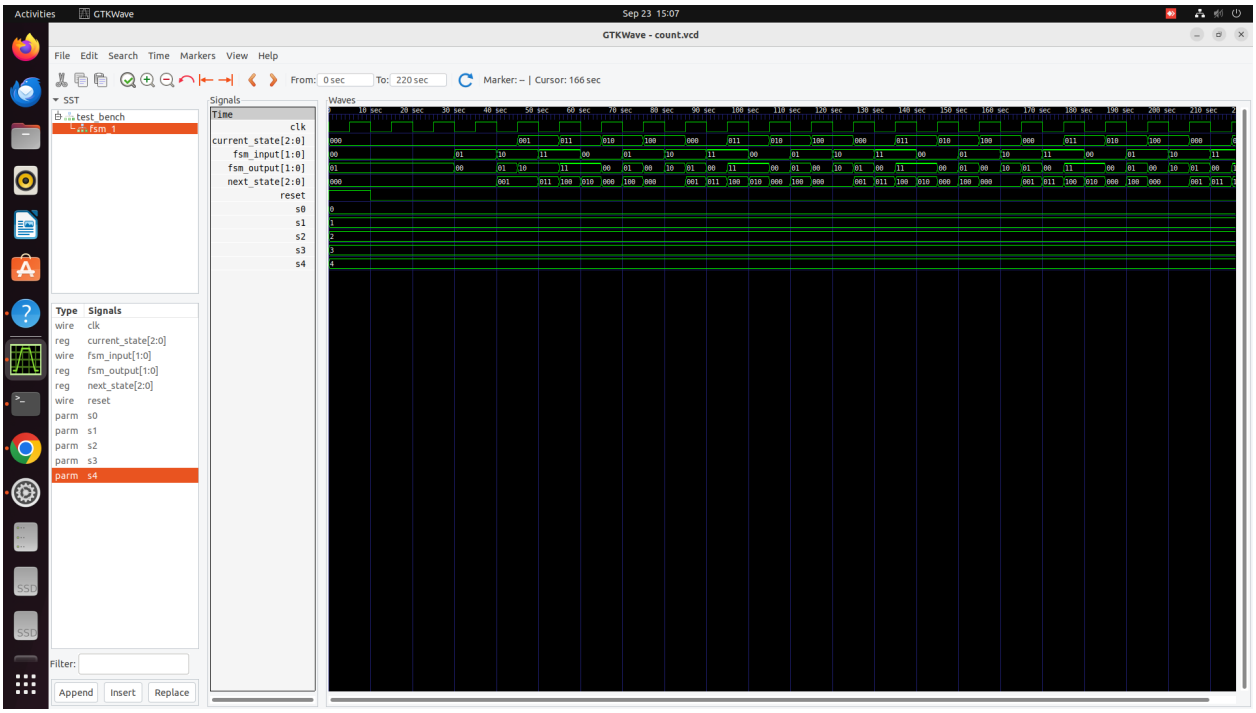
```
        #10 fsm_input = 2'b10; // Test input 10
        #10 fsm_input = 2'b11; // Test input 11
reset = 1;
#10
reset = 0;
        // Test state s1
        #10 fsm_input = 2'b00; // s1, input 00
        #10 fsm_input = 2'b01; // s1, input 01
        #10 fsm_input = 2'b10; // s1, input 10
        #10 fsm_input = 2'b11; // s1, input 11

        // Test state s2
        #10 fsm_input = 2'b00; // s2, input 00
        #10 fsm_input = 2'b01; // s2, input 01
        #10 fsm_input = 2'b10; // s2, input 10
        #10 fsm_input = 2'b11; // s2, input 11

        // Test state s3
        #10 fsm_input = 2'b00; // s3, input 00
        #10 fsm_input = 2'b01; // s3, input 01
        #10 fsm_input = 2'b10; // s3, input 10
        #10 fsm_input = 2'b11; // s3, input 11

        // Test state s4
        #10 fsm_input = 2'b00; // s4, input 00
        #10 fsm_input = 2'b01; // s4, input 01
        #10 fsm_input = 2'b10; // s4, input 10
        #10 fsm_input = 2'b11; // s4, input 11

        #10 fsm_input = 2'b00; // Test input 00
        #10 fsm_input = 2'b01; // Test input 01
        #10 fsm_input = 2'b10; // Test input 10
        #10 fsm_input = 2'b11; // Test input 11
        #10 fsm_input = 2'b00; // Test input 00
        #10 fsm_input = 2'b11; // Test input 00
        #10 fsm_input = 2'b10; // Test input 10
```

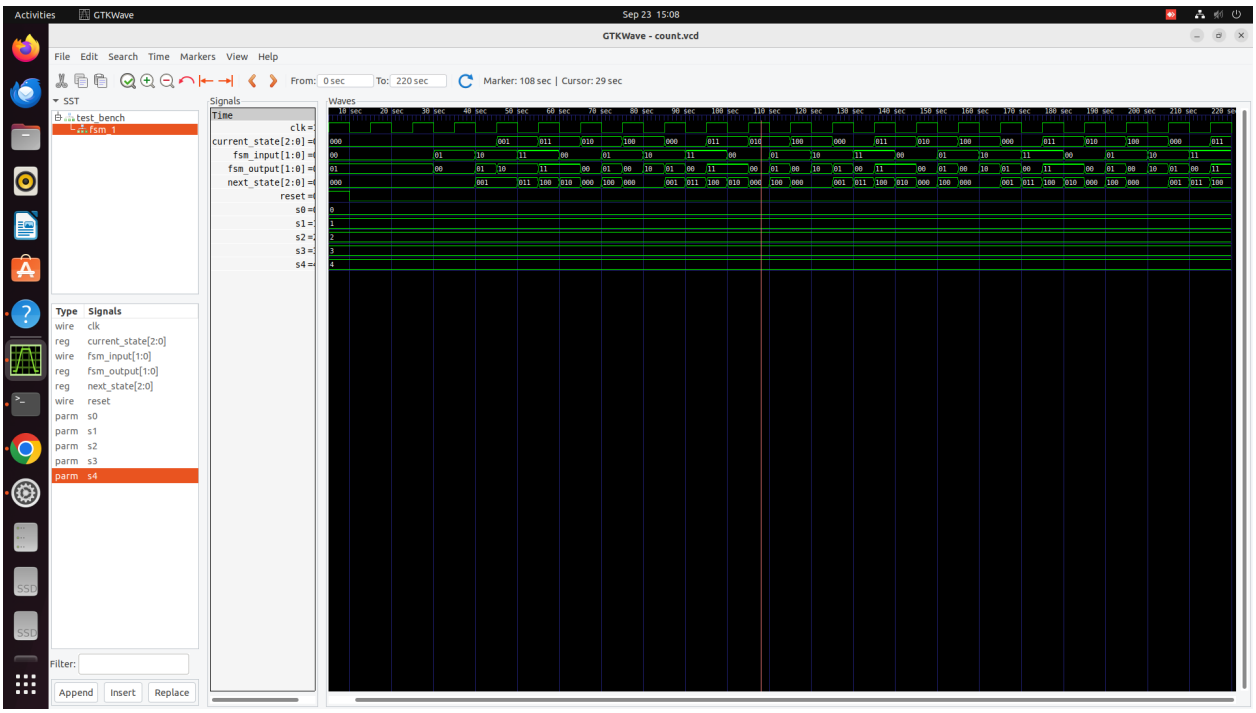
```
#10 fsm_input = 2'b10; // Test input 10
       #10 fsm_input = 2'b00; // Test input 00
        #10 fsm_input = 2'b01; // Test input 01
#10 fsm_input = 2'b01; // Test input 01
#10 fsm_input = 2'b10; // Test input 10
        #10 fsm_input = 2'b11; // Test input 00
#10 fsm_input = 2'b01; // Test input 01
#10 fsm_input = 2'b01; // Test input 01
        #10 fsm_input = 2'b10; // Test input 10
#10 fsm_input = 2'b11; // Test input 00
#10 fsm_input = 2'b10; // Test input 10
        // Finish simulation
        #10;
        $finish; // End simulation
    end

    // Monitor outputs
    initial begin
        $monitor("Time: %0t | Reset: %b | Input: %b | Output: %b
                 $time, reset, fsm_input, fsm_output);
    end

    // Generate VCD dump
    initial begin
        $dumpfile("count.vcd");
        $dumpvars(0);
    end

endmodule
```

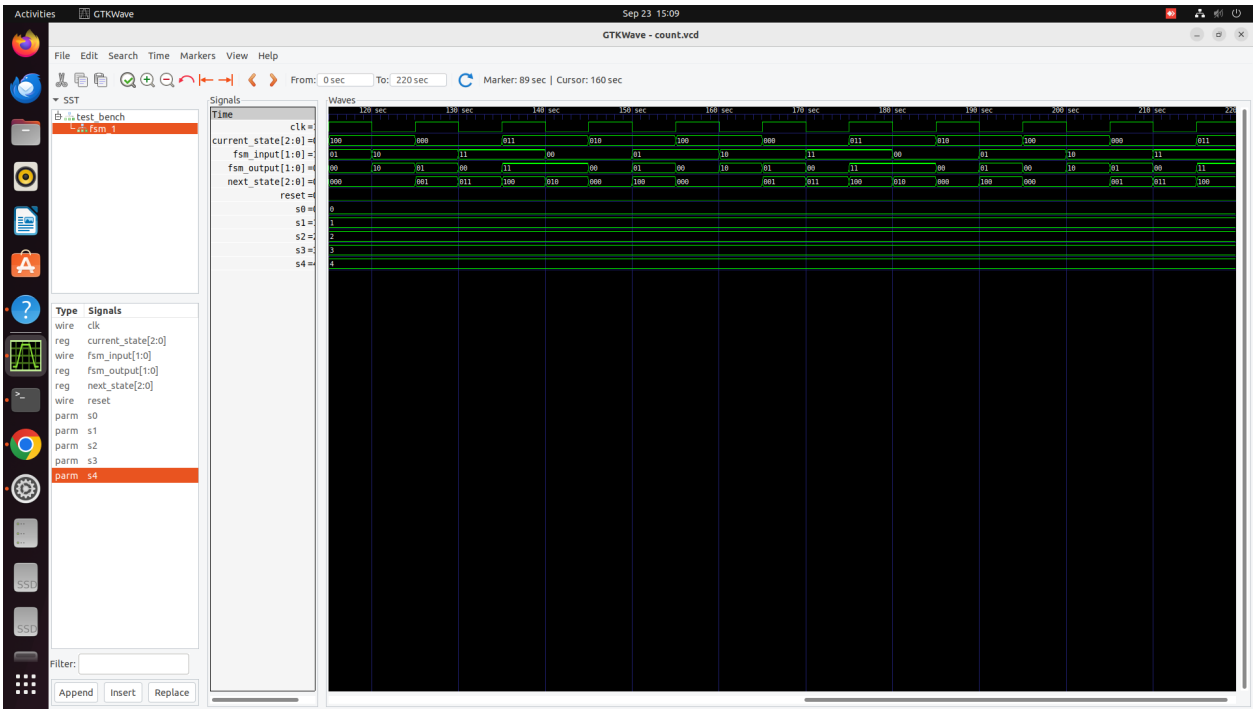
(c) simulation output waveform:

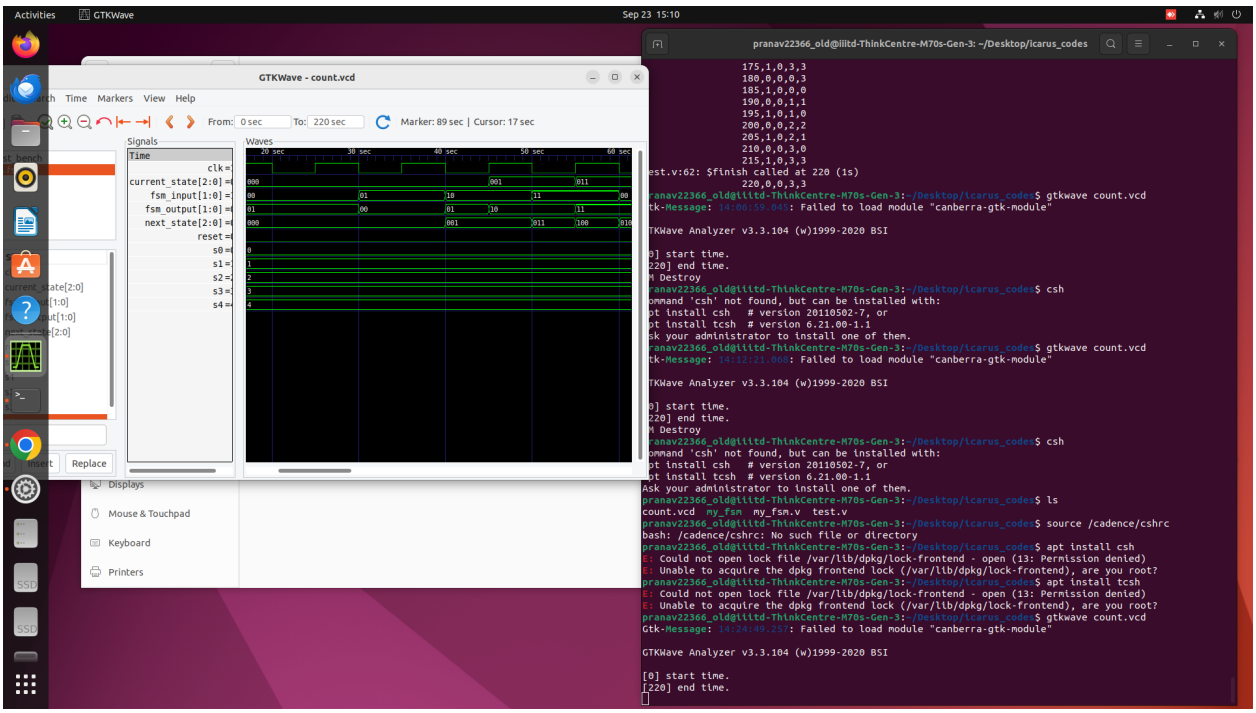

Activities
GTKWave
Sep 23 13:01
GTKWave - count.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 220 sec
Marker: 60 sec | Cursor: 10 sec
SST
test_bench
fsm_1
Signals
Time
current_state[2:0] =011
fsm_input[1:0] =00
fsm_output[1:0] =11
next_state[2:0] =010
reset =0
s0 =0
s1 =1
s2 =2
s3 =3
s4 =4
Waves
Type Signals
wire clk
reg current_state[2:0]
wire fsm_input[1:0]
reg fsm_output[1:0]
reg next_state[2:0]
wire reset
parm s0
parm s1
parm s2
parm s3
parm s4
Filter:
Append
Insert
Replace


Activities
GTKWave
Sep 23 13:02
GTKWave - count.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 220 sec
Marker: 44 sec | Cursor: 54 sec
SST
test_bench
fsm_1
Signals
Time
current_state[2:0] =000
fsm_input[1:0] =10
fsm_output[1:0] =01
next_state[2:0] =001
reset =0
s0 =0
s1 =1
s2 =2
s3 =3
s4 =4
Waves
Type Signals
wire clk
reg current_state[2:0]
wire fsm_input[1:0]
reg fsm_output[1:0]
reg next_state[2:0]
wire reset
parm s0
parm s1
parm s2
parm s3
parm s4
Filter:
Append
Insert
Replace


Activities
GTKWave
Sep 23 15:07
GTKWave - count.vcd
File Edit Search Time Markers View Help
From: 0 sec To: 220 sec
Marker: – | Cursor: 166 sec
SST
test_bench
fsm_1
Signals
Time
clk
current_state[2:0]
fsm_input[1:0]
fsm_output[1:0]
next_state[2:0]
reset
s0
s1
s2
s3
s4
Waves
Type Signals
wire clk
reg current_state[2:0]
wire fsm_input[1:0]
reg fsm_output[1:0]
reg next_state[2:0]
wire reset
parm s0
parm s1
parm s2
parm s3
parm s4
Filter:
Append
Insert
Replace


Activities
GTKWave
Sep 23 15:08
GTKWave - count.vcd
File Edit Search Time Markers View Help
From: 0 sec To: 220 sec
Marker: 108 sec | Cursor: 29 sec
SST
test_bench
fsm_1
Signals
Time
clk
current_state[2:0]
fsm_input[1:0]
fsm_output[1:0]
next_state[2:0]
reset
s0
s1
s2
s3
s4
Waves
Type Signals
wire clk
reg current_state[2:0]
wire fsm_input[1:0]
reg fsm_output[1:0]
reg next_state[2:0]
wire reset
parm s0
parm s1
parm s2
parm s3
parm s4
Filter:
Append
Insert
Replace





Explanation of correctness of model:

As per the verilog code , the state transitions happen at posedge clk since this is a sequential block. The output is evaluated on the basis of the current state and input since this is a combinational block. For 10s , the reset has been toggled ON ,

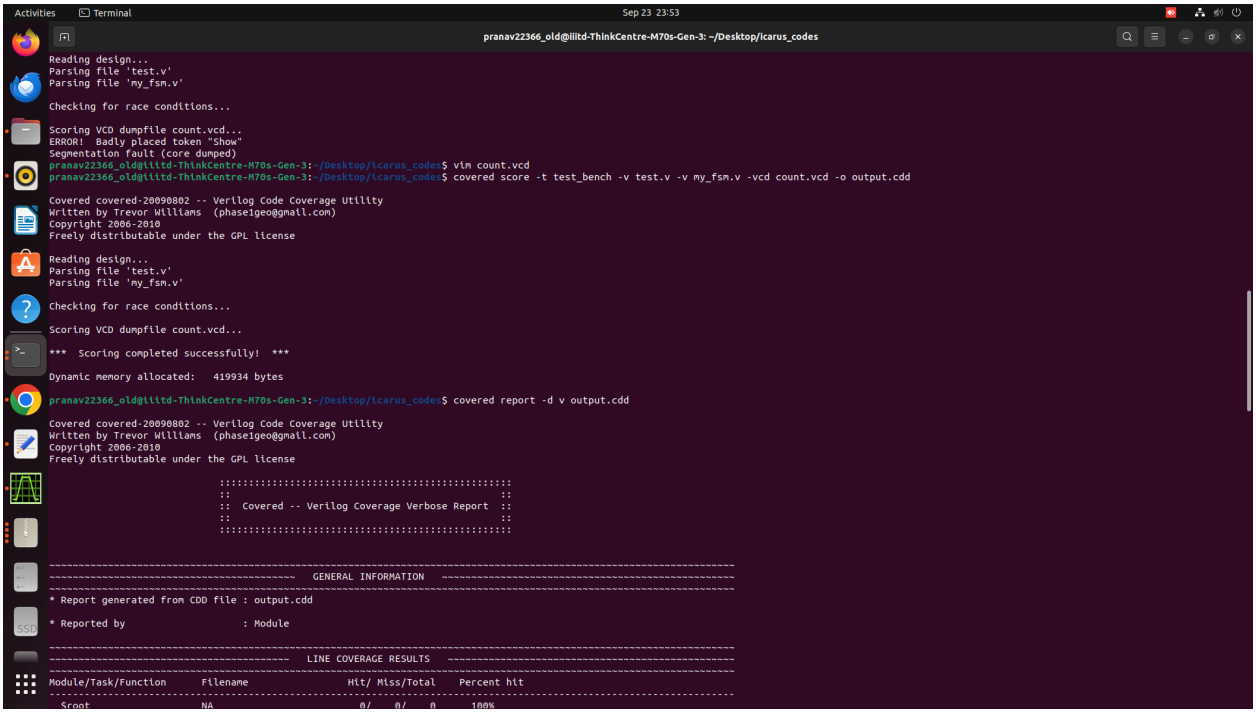
since the next state for input:00 and current state : s0 is the same as the current state , the output remains same and no state transition happens. At t=30 , the input changes to 01 and so output changes to 00 but next_state doesnt change so no state transition happens at posedge at t=35s.

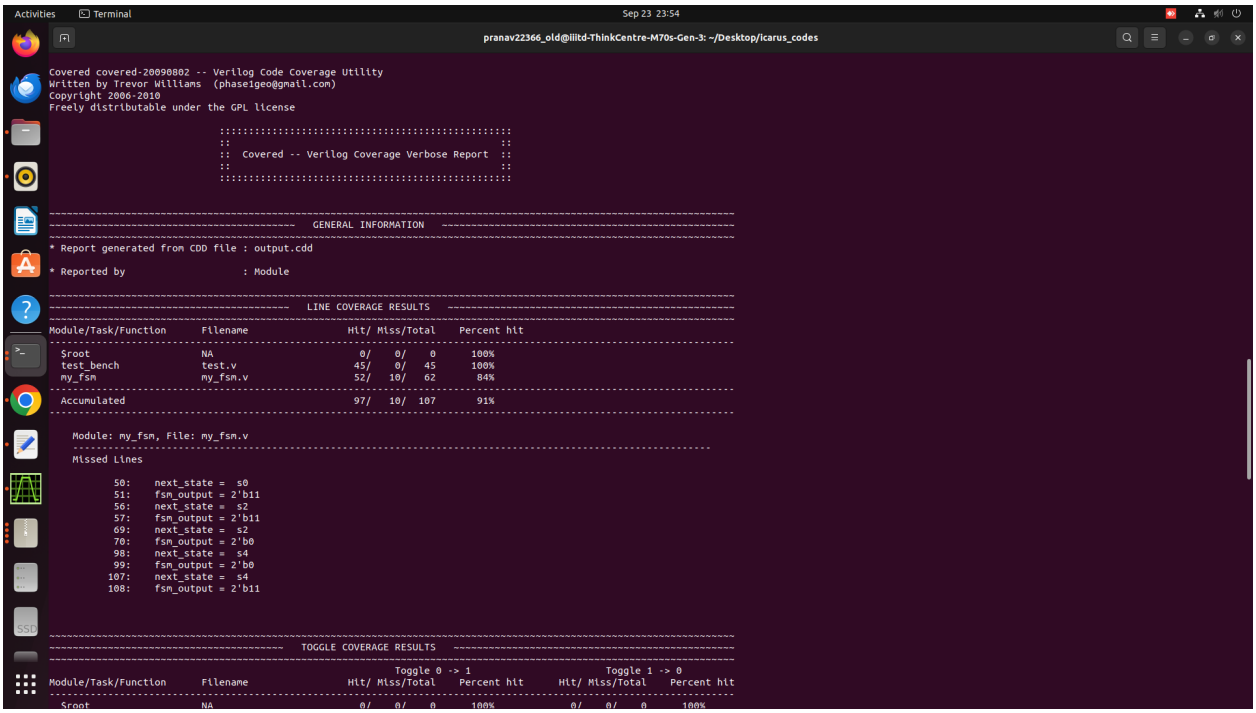
Now at t=40s , the input changes to 10 and next_state becomes s2 , so at t=45s when there is a rising edge the current_state becomes s2. With input 11 arriving at t=50s and current state being s2, the next state is evaluated to s4 and the state transition happens at the posedge at t=55s to s4. This continue uptil t=270s.

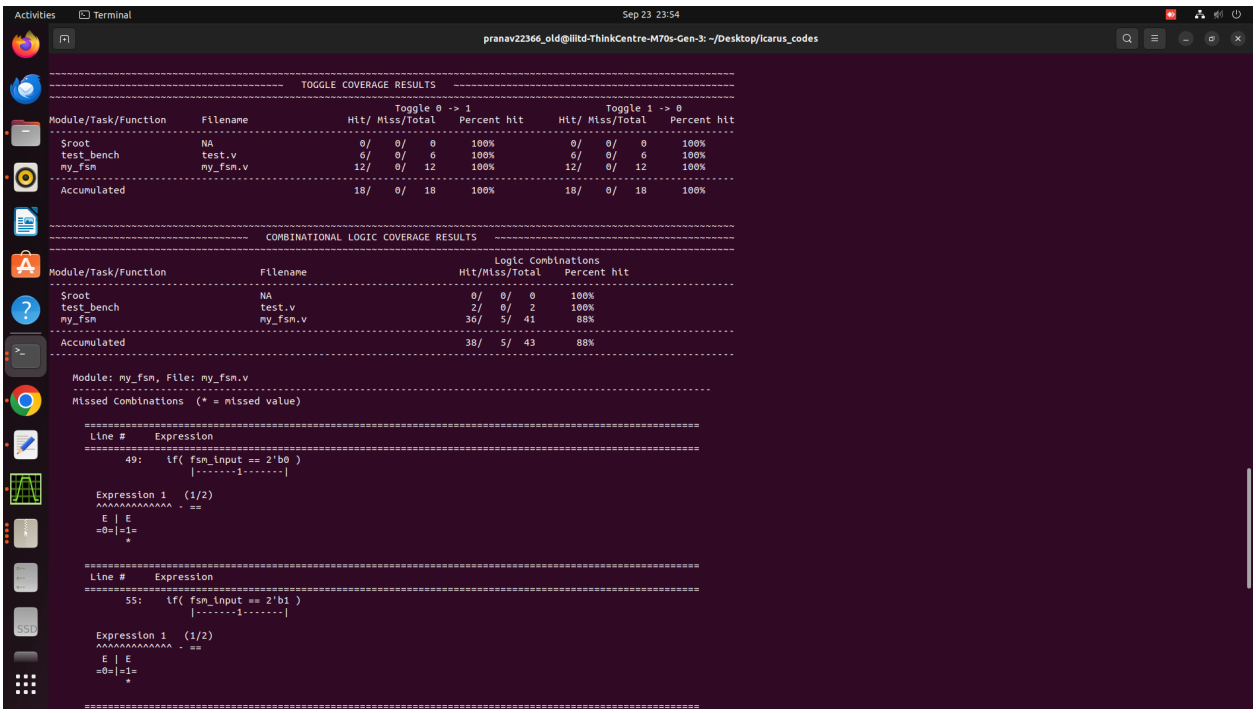
Now like this , the input changes every 10s and the next_state is evaluated , the current_state is given next_state at the posedge. This is is how state transition happens only at positive edge of clock (sequential block) while output is evaluted instantly as state/input changes (combinational block).

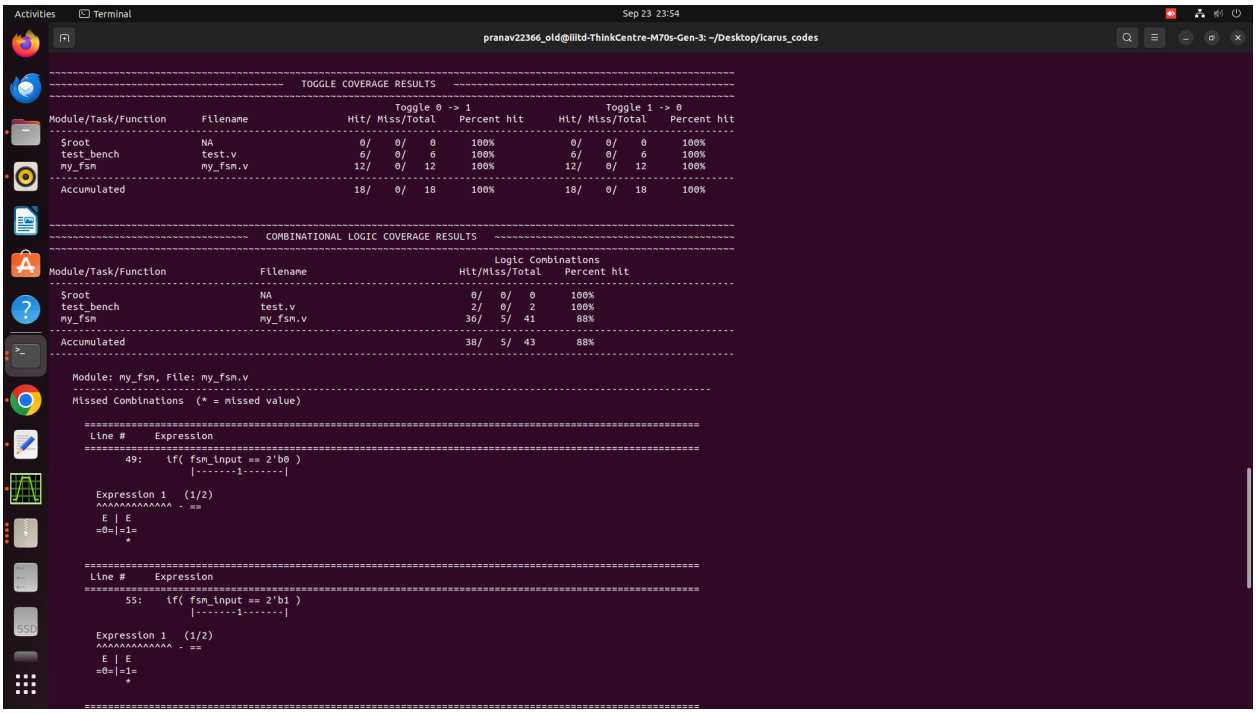
This show that the simulation waveform correctly exemplifies the state transition table.

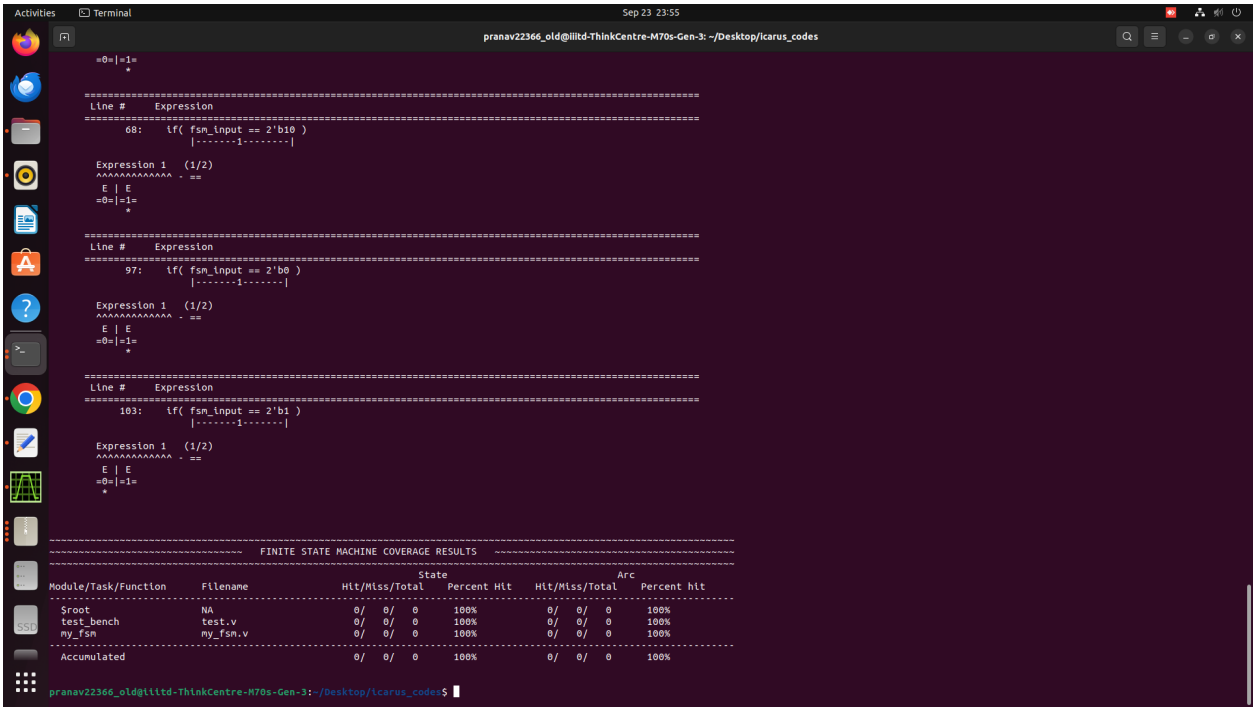
(d) line coverage report:




Activities
Terminal
Sep 23 23:54
pranav22366_old@lilitd-ThinkCentre-M70s-Gen-3: ~/Desktop/icarus_codes
Covered covered-20090802 -- Verilog Code Coverage Utility
Written by Trevor Williams (phase1geo@gmail.com)
Copyright 2006-2010
Freely distributable under the GPL license
:: Covered -- Verilog Coverage Verbose Report ::
GENERAL INFORMATION
* Report generated from CDD file : output.cdd
* Reported by : Module
LINE COVERAGE RESULTS
Module/Task/Function Filename Hit/ Miss/Total Percent hit
$root NA 0/ 0/ 0 100%
test_bench test.v 45/ 0/ 45 100%
my_fsm my_fsm.v 52/ 10/ 62 84%
Accumulated 97/ 10/ 107 91%
Module: my_fsm, File: my_fsm.v
Missed Lines
50: next_state = s0
51: fsm_output = 2'b11
56: next_state = s2
57: fsm_output = 2'b11
69: next_state = s2
70: fsm_output = 2'b0
98: next_state = s4
99: fsm_output = 2'b0
107: next_state = s4
108: fsm_output = 2'b11
TOGGLE COVERAGE RESULTS
Toggle 0 -> 1
Toggle 1 -> 0
Module/Task/Function Filename Hit/ Miss/Total Percent hit
Hit/ Miss/Total Percent hit
$root NA 0/ 0/ 0 100%
0/ 0/ 0 100%


Activities
Terminal
Sep 23 23:54
pranav22366_old@lilitd-ThinkCentre-M70s-Gen-3: ~/Desktop/icarus_codes
TOGGLE COVERAGE RESULTS
Toggle 0 -> 1
Toggle 1 -> 0
Module/Task/Function Filename Hit/ Miss/Total Percent hit
Hit/ Miss/Total Percent hit
$root NA 0/ 0/ 0 100% 0/ 0/ 0 100%
test_bench test.v 6/ 0/ 6 100% 6/ 0/ 6 100%
my_fsm my_fsm.v 12/ 0/ 12 100% 12/ 0/ 12 100%
Accumulated 18/ 0/ 18 100% 18/ 0/ 18 100%
COMBINATIONAL LOGIC COVERAGE RESULTS
Logic Combinations
Module/Task/Function Filename Hit/Miss/Total Percent hit
$root NA 0/ 0/ 0 100%
test_bench test.v 2/ 0/ 2 100%
my_fsm my_fsm.v 36/ 5/ 41 88%
Accumulated 38/ 5/ 43 88%
Module: my_fsm, File: my_fsm.v
Missed Combinations (* = missed value)
Line # Expression
49: if(fsm_input == 2'b0)
|-------1-------|
Expression 1 (1/2)
E | E
=0=|=1=
*
Line # Expression
55: if(fsm_input == 2'b1)
|-------1-------|
Expression 1 (1/2)
E | E
=0=|=1=
*





Explanation of report:

Line Coverage :

This report covers how many lines of the testbench and verilog source file have not been executed throughout the execution.The report highlights 10 lines as

missed , where every 2 lines constitute a state of the FSM. So in total 5 states of the FSM described in the source file have never been reached. Using the line numbers given , we can give the states as :

(s1,00) , (s1,01) , (s2,10) , (s4,00) , (s4,11) where x represents current state and y represents input . Here at Line number 50 of testbench , we are in the (s3,11) state for 10 s, after which I have changed input to hit the states (s2,11) , (s3,10) , (s3,01) which werent hitting earlier . As a result of the test bench not having cases corresponding to the 5 cases , the 10 lines havent been hit in the fsm. Then at line 35 , the reset is toggled again from 1 to 0 , thus covering all cases of reset toggling , this is shown in the toggle coverage report.

Toggle Coverage:

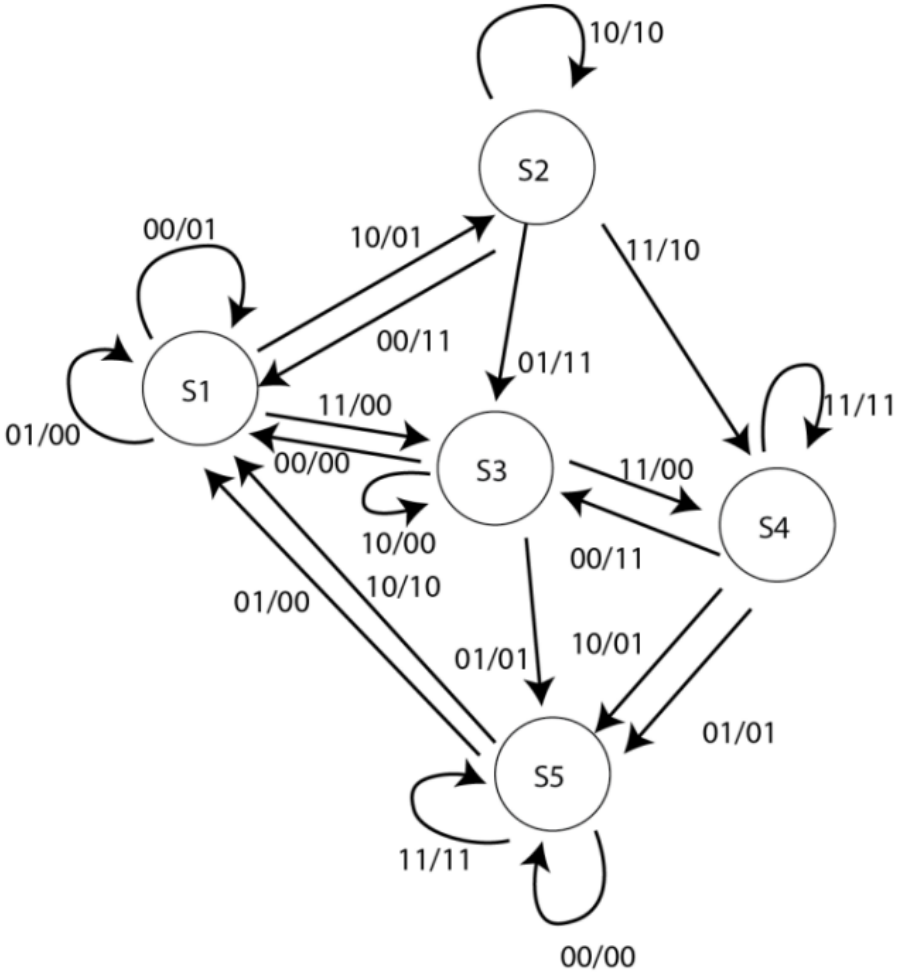
The toggle coverage covers the cases of inputs being driven from 0-1 and 1-0 which is the reset signal in out case. In the first 10s , I am toggling reset from 1 to 0 to show the FSM resetting to S0 when toggled ON . Thus, this covers all cases acheving a 100% toggle coverage.

Combinational Logic Coverage:

As can be seen in the line coverage report, the 5 states that havent been hit are the same 5 combinational logic cases for which output and next state hasnt been evaluated.The states are the same: (s1,00) , (s1,01) , (s2,10) , (s4,00) , (s4,11). These are the states for which nxt state and output hasnt been evaluated , hence the combinational block hasnt been executed which shows up in the combinational report .

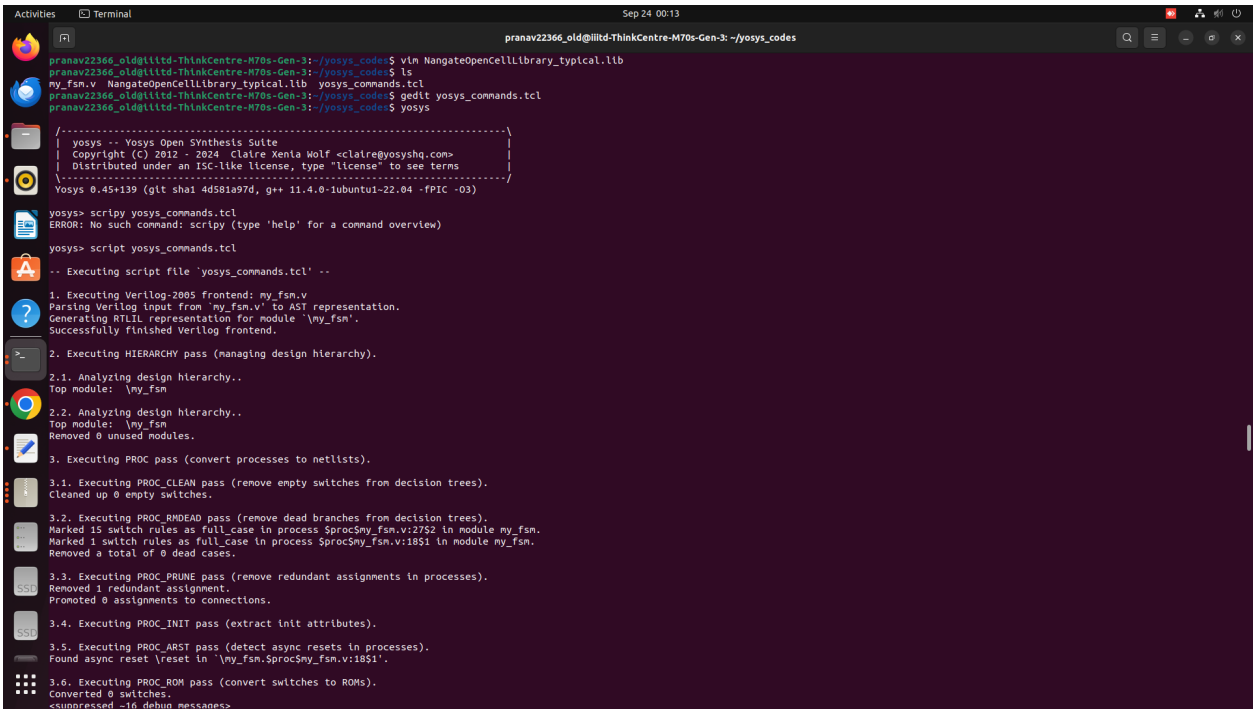
For combinational and line coverage :

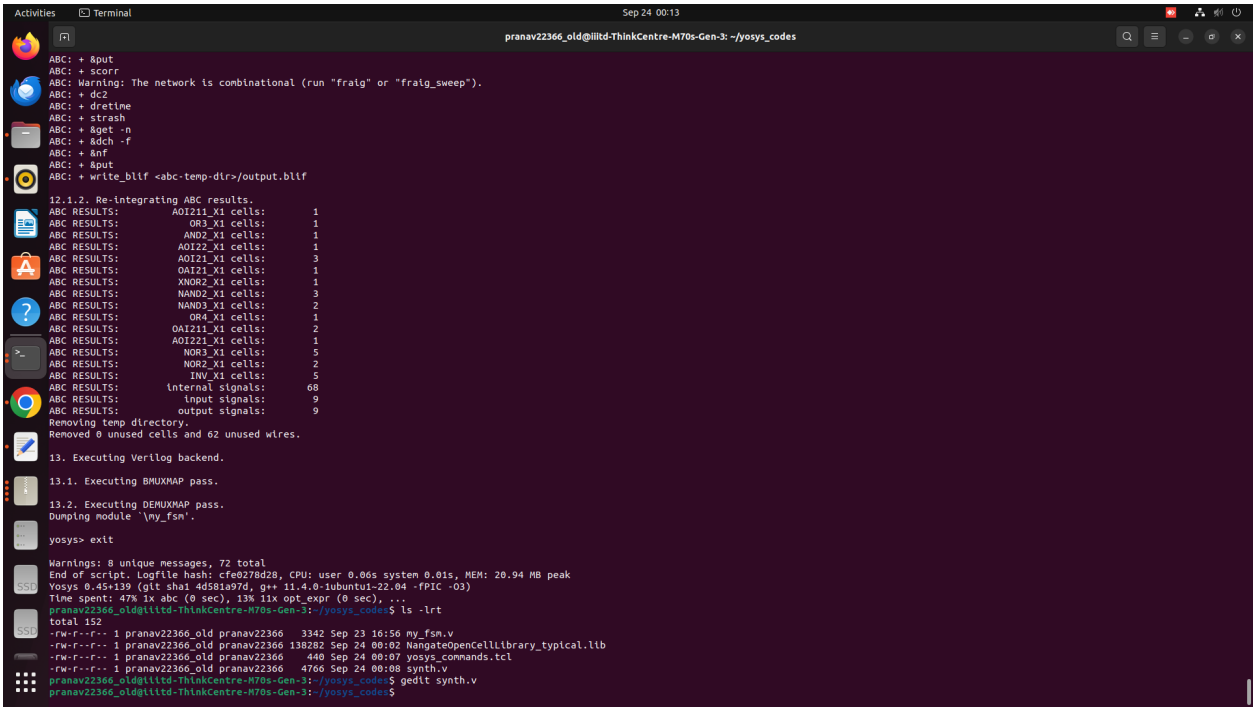
the state transitions explained



Here we can the see the 2 s1 states : (s1,00) and (s1,01) , 2 s4 states : (s4,00), (s4,11) and 1 s2 state : (s2,10) state which hasnt been hit up.

(e)Synthesis:





Synth.v (Synthesis File):

/* Generated by Yosys 0.45+139 (git sha1 4d581a97d, g++ 11.4.0-1ubuntu1~22.04 -fPIC -O3) */

(* top = 1 *)

```
(* src = "my_fsm.v:3.1-112.10" *)
module my_fsm(clk, reset, fsm_input, fsm_output);
wire _00_;
wire _01_;
wire _02_;
wire _03_;
wire _04_;
wire _05_;
wire _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_;
(* force_downto = 32'd1 *)
(* src = "my_fsm.v:0.0-0.0|my_fsm.v:28.
5-109.12|/usr/local/bin/../
share/yosys/techmap.v:575.21-
575.22" *)
wire [2:0] _28_;
(* force_downto = 32'd1 *)
(* src = "my_fsm.v:0.0-0.0|my_fsm.v:28.
5-109.12|/usr/local/bin/../
share/yosys/techmap.v:575.21-
575.22" *)
wire [1:0] _29_;
(* src = "my_fsm.v:4.16-4.19" *)
input clk;
wire clk;
(* src = "my_fsm.v:11.11-11.24" *)
wire [2:0] current_state;
(* src = "my_fsm.v:6.22-6.31" *)
input [1:0] fsm_input;
wire [1:0] fsm_input;
(* src = "my_fsm.v:7.22-7.32" *)
output [1:0] fsm_output;
reg [1:0] fsm_output;
(* src = "my_fsm.v:11.26-11.36" *)
```

```
reg [2:0] next_state;
(* src = "my_fsm.v:5.16-5.21" *)
input reset;
wire reset;
INV_X1 _30_ (
.A(reset),
.ZN(_04_)
);
INV_X1 _31_ (
.A(fsm_input[1]),
.ZN(_07_)
);
NOR3_X1 _32_ (
.A1(current_state[0]),
.A2(current_state[2]),
.A3(_01_),
.ZN(_08_)
);
NOR3_X1 _33_ (
.A1(current_state[0]),
.A2(current_state[1]),
.A3(_00_),
.ZN(_09_)
);
NOR3_X1 _34_ (
.A1(current_state[1]),
.A2(current_state[2]),
```

```
.A3(_02_),
.ZN(_10_)
);
NOR3_X1 _35_ (
.A1(current_state[2]),
.A2(_02_),
.A3(_01_),
.ZN(_11_)
);
AOI211_X1 _36_ (
.A(current_state[2]),
.B(_01_),
.C1(_02_),
.C2(current_state[0]),
.ZN(_12_)
);
OR3_X1 _37_ (
.A1(_09_),
.A2(_10_),
.A3(_12_),
.ZN(_13_)
);
AOI21_X1 _38_ (
.A(_10_),
.B1(_08_),
.B2(fsm_input[0]),
.ZN(_14_)
```

```
);
AOI21_X1 _39_ (
.A(_07_),
.B1(_13_),
.B2(_14_),
.ZN(_28_[0])
);
AND2_X1 _40_ (
.A1(fsm_input[0]),
.A2(fsm_input[1]),
.ZN(_15_)
);
NAND2_X1 _41_ (
.A1(fsm_input[0]),
.A2(fsm_input[1]),
.ZN(_16_)
);
NOR2_X1 _42_ (
.A1(fsm_input[0]),
.A2(fsm_input[1]),
.ZN(_17_)
);
NAND2_X1 _43_ (
.A1(fsm_input[1]),
.A2(_08_),
.ZN(_18_)
);
```

```
AOI22_X1 _44_ (
.A1(fsm_input[0]),
.A2(_10_),
.B1(_11_),
.B2(_17_),
.ZN(_19_)
);
OAI211_X1 _45_ (
.A(_18_),
.B(_19_),
.C1(_13_),
.C2(_16_),
.ZN(_28_[1])
);
NAND3_X1 _46_ (
.A1(fsm_input[0]),
.A2(_07_),
.A3(_08_),
.ZN(_20_)
);
AOI21_X1 _47_ (
.A(_11_),
.B1(_15_),
.B2(_09_),
.ZN(_21_)
);
OAI21_X1 _48_ (
```

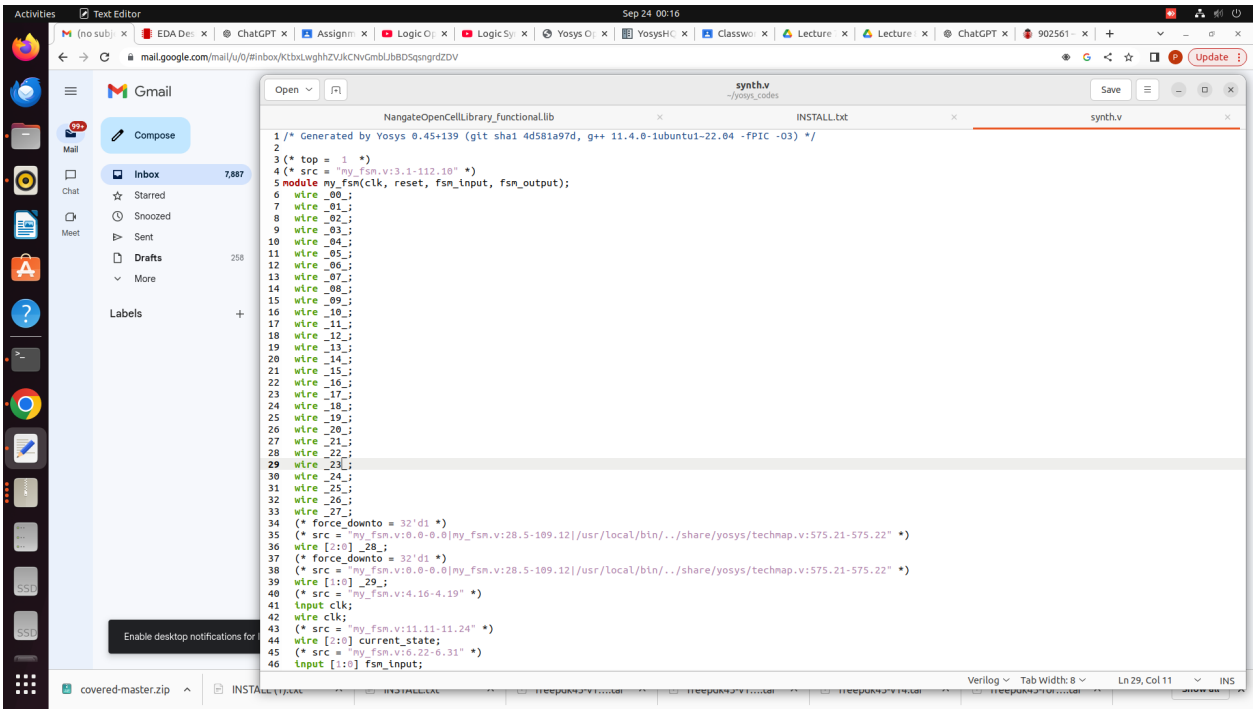
```
.A(_11_),
.B1(fsm_input[1]),
.B2(fsm_input[0]),
.ZN(_22_)
);
XNOR2_X1 _49_ (
.A(fsm_input[0]),
.B(fsm_input[1]),
.ZN(_23_)
);
NAND2_X1 _50_ (
.A1(_09_),
.A2(_23_),
.ZN(_24_)
);
NAND3_X1 _51_ (
.A1(_20_),
.A2(_22_),
.A3(_24_),
.ZN(_28_[2])
);
OR4_X1 _52_ (
.A1(fsm_input[0]),
.A2(_09_),
.A3(_10_),
.A4(_12_),
.ZN(_25_)
```

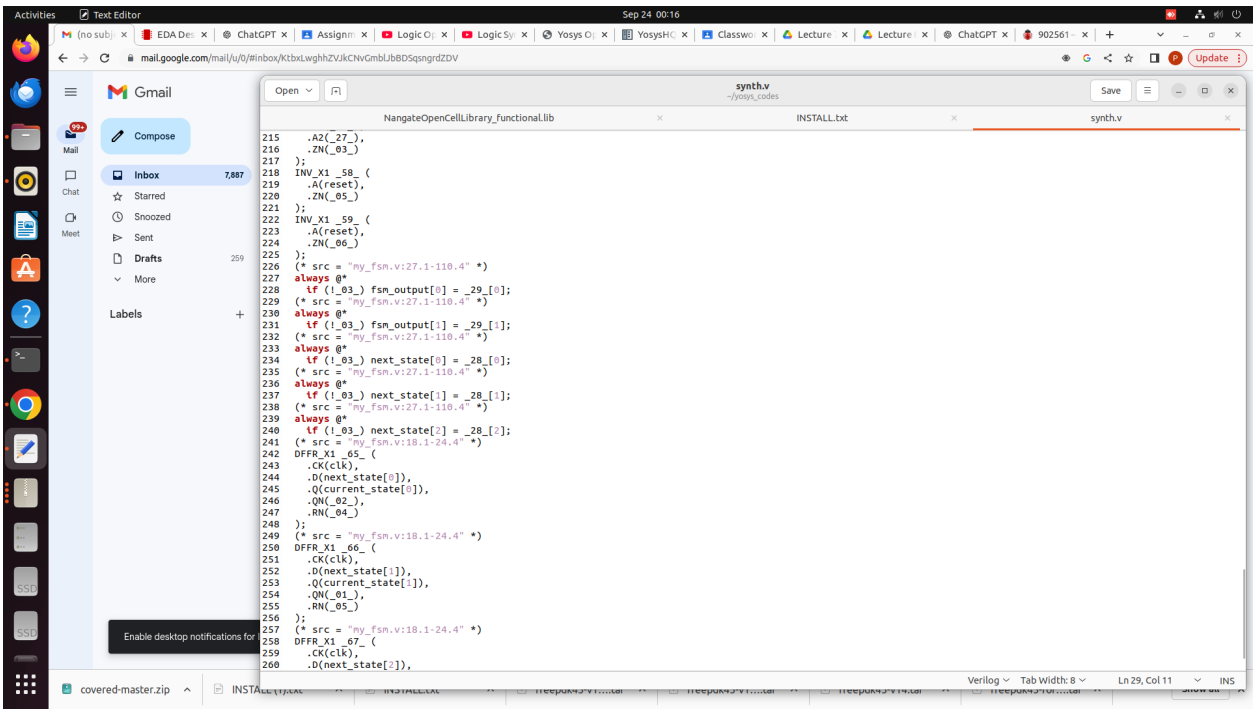
```
);
OAI211_X1 _53_ (
.A(_21_),
.B(_25_),
.C1(fsm_input[1]),
.C2(_14_),
.ZN(_29_[0])
);
AOI221_X1 _54_ (
.A(_10_),
.B1(_11_),
.B2(_23_),
.C1(_09_),
.C2(fsm_input[1]),
.ZN(_26_)
);
INV_X1 _55_ (
.A(_26_),
.ZN(_29_[1])
);
NOR3_X1 _56_ (
.A1(current_state[0]),
.A2(current_state[1]),
.A3(current_state[2]),
.ZN(_27_)
);
NOR2_X1 _57_ (
```

```
.A1(_13_),
.A2(_27_),
.ZN(_03_)
);
INV_X1 _58_ (
.A(reset),
.ZN(_05_)
);
INV_X1 _59_ (
.A(reset),
.ZN(_06_)
);
(* src = "my_fsm.v:27.1-110.4" *)
always @*
if (!_03_) fsm_output[0] = _29_[0];
(* src = "my_fsm.v:27.1-110.4" *)
always @*
if (!_03_) fsm_output[1] = _29_[1];
(* src = "my_fsm.v:27.1-110.4" *)
always @*
if (!_03_) next_state[0] = _28_[0];
(* src = "my_fsm.v:27.1-110.4" *)
always @*
if (!_03_) next_state[1] = _28_[1];
(* src = "my_fsm.v:27.1-110.4" *)
always @*
if (!_03_) next_state[2] = _28_[2];
```

```
(* src = "my_fsm.v:18.1-24.4" *)
DFFR_X1 _65_ (
.CK(clk),
.D(next_state[0]),
.Q(current_state[0]),
.QN(_02_),
.RN(_04_)
);
(* src = "my_fsm.v:18.1-24.4" *)
DFFR_X1 _66_ (
.CK(clk),
.D(next_state[1]),
.Q(current_state[1]),
.QN(_01_),
.RN(_05_)
);
(* src = "my_fsm.v:18.1-24.4" *)
DFFR_X1 _67_ (
.CK(clk),
.D(next_state[2]),
.Q(current_state[2]),
.QN(_00_),
.RN(_06_)
);
endmodule
```

Photos:





The synthesis file is a netlist of logic gates that can be mapped onto physical hardware, YOSYS synthesis tool translates the RTL code of the fsm to gate netlist.

The netlist is a collection of wires , signal declarations like clk,reset, combinational logic like AND,OR,NOR ,AOR gates , sequential circuit elements like D Flip Flops .

It shows how the FSM transitions between states using combinational logic, flip-flops, and generates outputs based on inputs and current state. The various logic gates (AND, OR, NOR, AOI, etc.) implement the FSM's logic, while the D flip-flops store the current state, triggered by the clock and reset signals. The Yosys tool is fed the .lib file and the verilog source file in a .tcl which is parsed and the cells of the 45nm technology are used for the instantiation of logical and sequential logical elements. The synthesis file can be seen using ls -lrt command and opened .

(f)

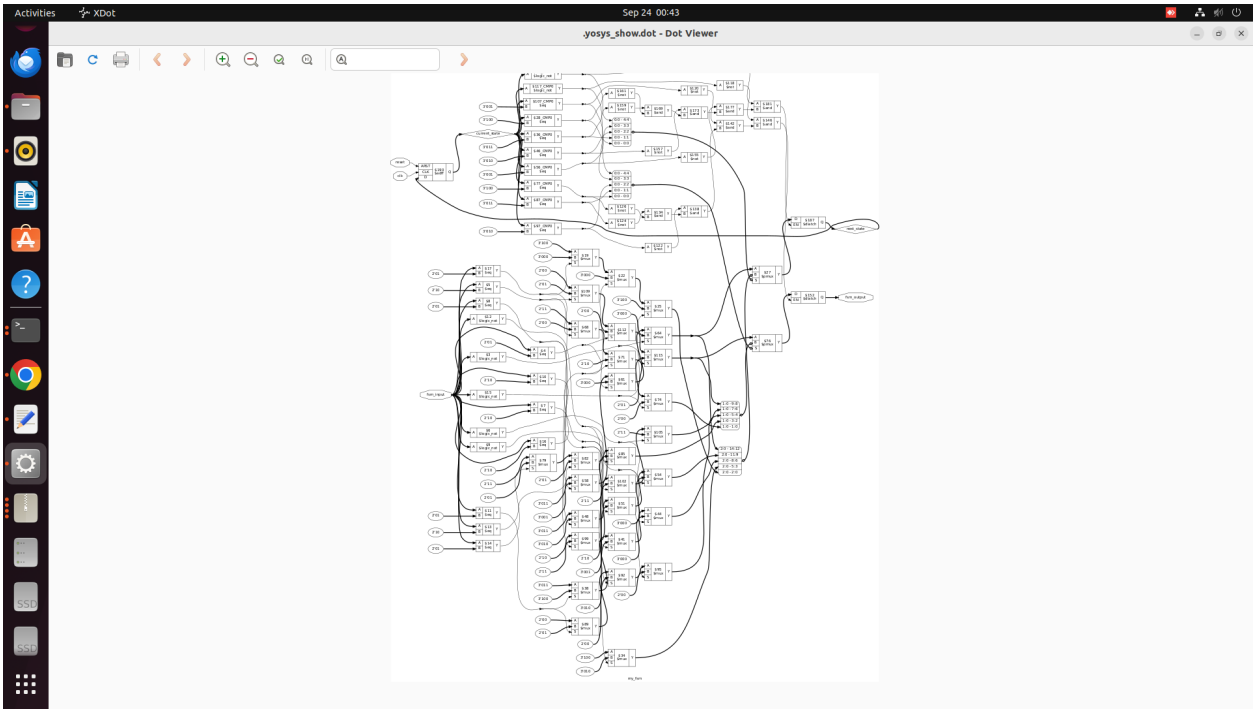
Using one-hot encoding:

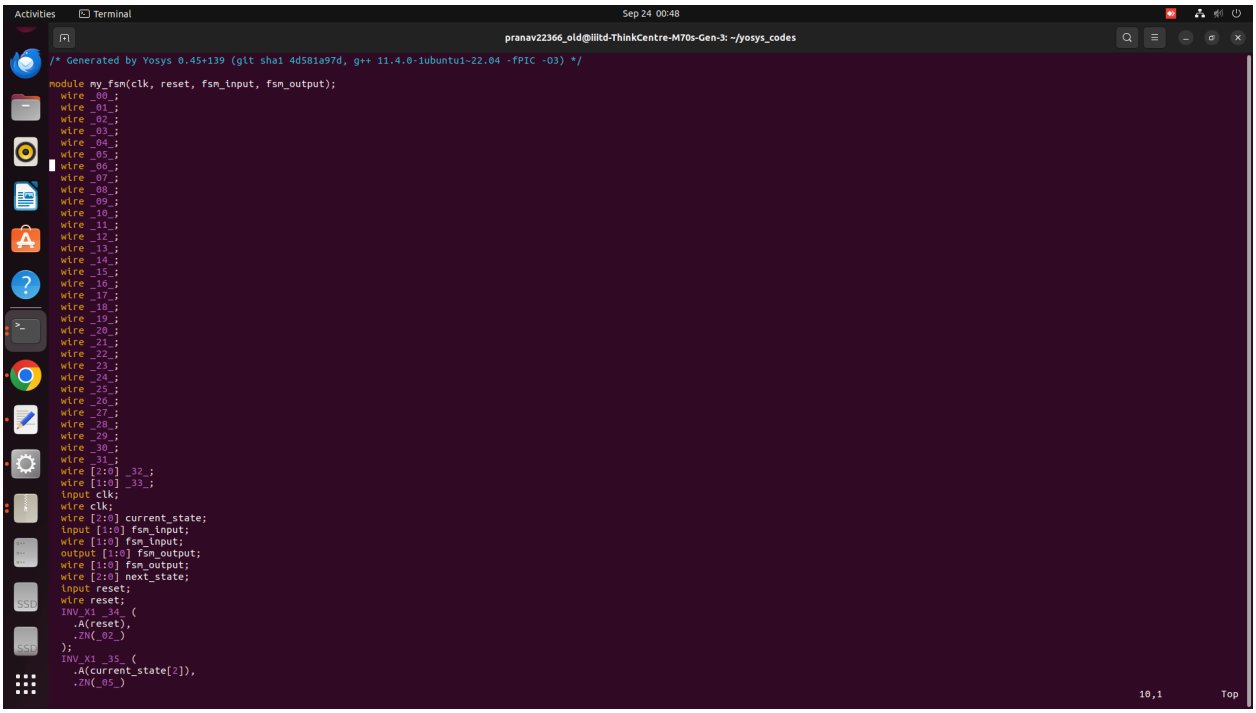
```
// now we will do one hot encoding
// every state is represented by singke bit set to 1 , with al e
// so new states are : 00001,00010,00100,01000,10000
// // we have states s1,s2,s3,s4,s5
// // States : {00001,00010,00100,01000,10000}
module my_fsm(
    input wire clk,
    input wire reset,
    input wire [1:0] fsm_input,
    output reg [1:0] fsm_output
);

// State definitions
reg [5:0] current_state, next_state;
parameter s0 = 5'b00001;
parameter s1 = 5'b00010;
parameter s2 = 5'b00100;
parameter s3 = 5'b01000;
parameter s4 = 5'b10000;

// Initialize current_state on reset
always @(posedge clk or posedge reset) begin
    if (reset) begin
        current_state <= s0; // Reset to initial state
    end else begin
```

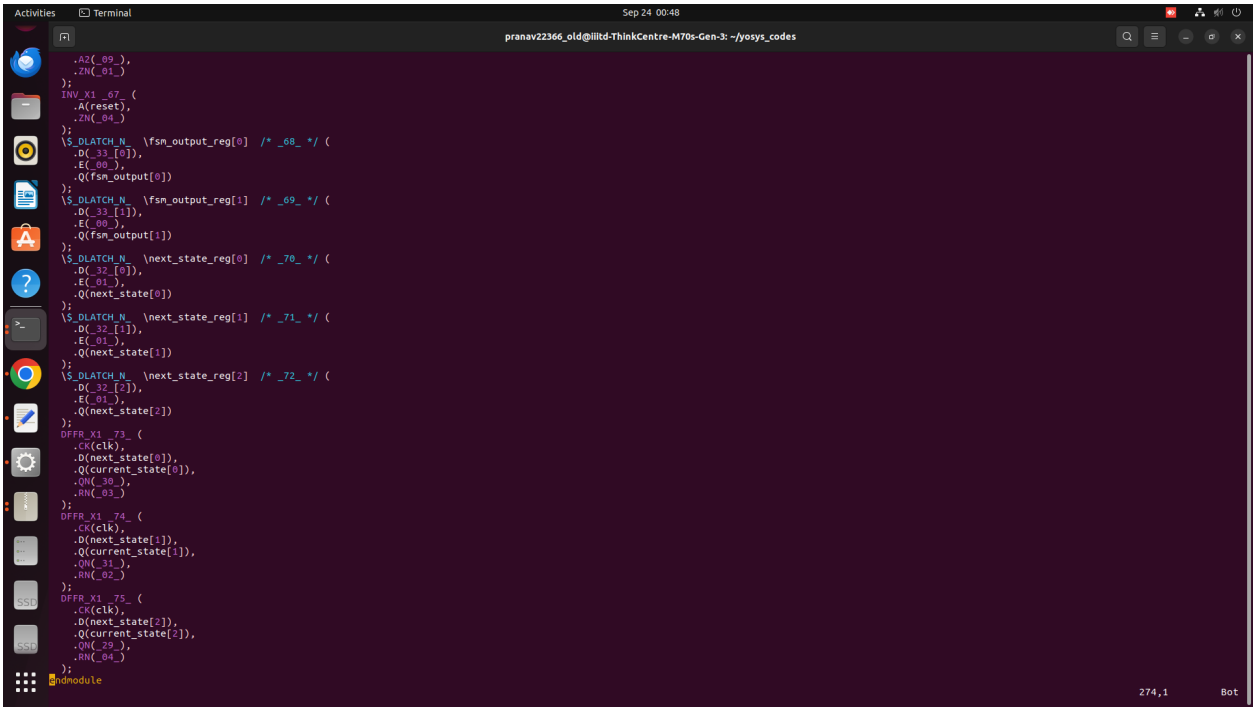
```
        current_state <= next_state; // Update current state
    end
end

// Combinational block for state transition and output logic
always @(*) begin
    case (current_state)
        s0: begin
            if (fsm_input == 2'b00) begin
                next_state = s0;
                fsm_output = 2'b01;
            end else if (fsm_input == 2'b01) begin
                next_state = s0;
                fsm_output = 2'b00;
            end else if (fsm_input == 2'b10) begin
                next_state = s1;
                fsm_output = 2'b01;
            end else begin
                next_state = s3;
                fsm_output = 2'b00;
            end
        end
        s1: begin
            if (fsm_input == 2'b00) begin
                next_state = s0;
                fsm_output = 2'b11;
            end else if (fsm_input == 2'b10) begin
                next_state = s1;
                fsm_output = 2'b10;
            end else if (fsm_input == 2'b01) begin
                next_state = s2;
                fsm_output = 2'b11;
            end else begin
                next_state = s3;
                fsm_output = 2'b10;
            end
```

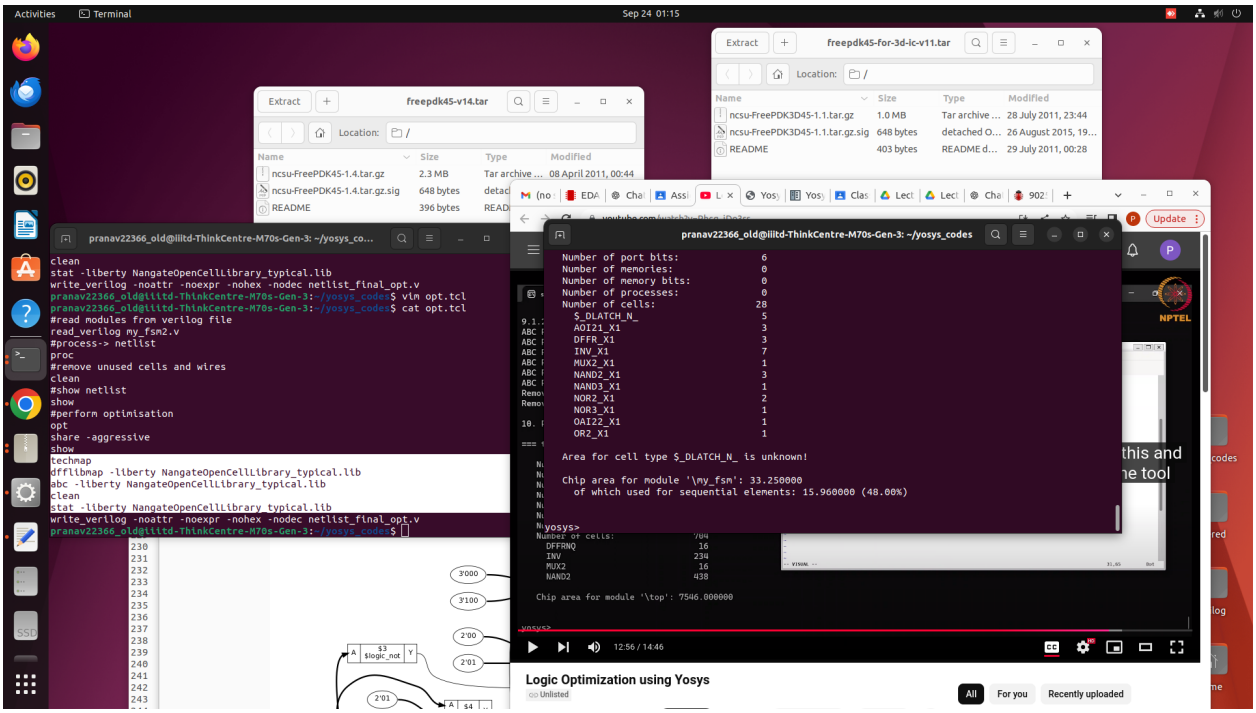
```
        end
        s2: begin
            if (fsm_input == 2'b00) begin
                next_state = s0;
                fsm_output = 2'b00;
            end else if (fsm_input == 2'b10) begin
                next_state = s2;
                fsm_output = 2'b00;
            end else if (fsm_input == 2'b01) begin
                next_state = s4;
                fsm_output = 2'b01;
            end else begin
                next_state = s3;
                fsm_output = 2'b00;
            end
        end
        s3: begin
            if (fsm_input == 2'b00) begin
                next_state = s2;
                fsm_output = 2'b11;
            end else if (fsm_input == 2'b10) begin
                next_state = s4;
                fsm_output = 2'b01;
            end else if (fsm_input == 2'b01) begin
                next_state = s4;
                fsm_output = 2'b01;
            end else begin
                next_state = s4;
                fsm_output = 2'b11;
            end
        end
        s4: begin
            if (fsm_input == 2'b00) begin
                next_state = s4;
                fsm_output = 2'b00;
            end else if (fsm_input == 2'b10) begin
```

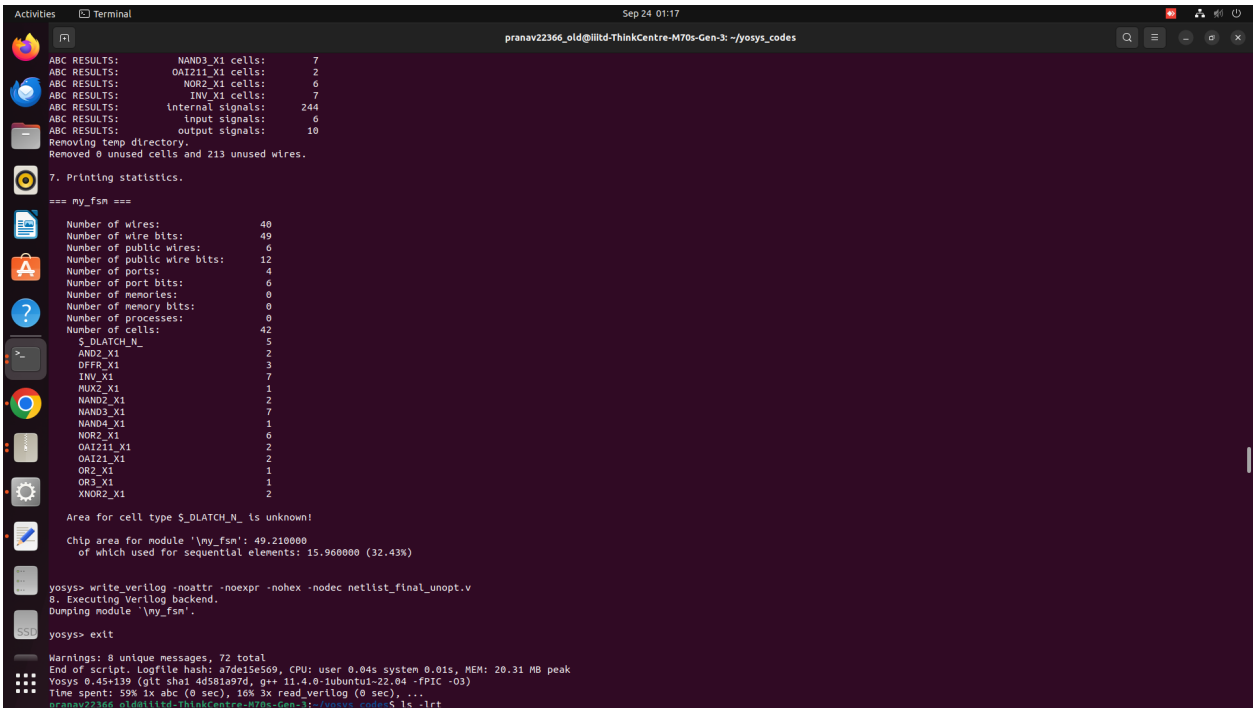
```
                next_state = s0;
                fsm_output = 2'b10;
            end else if (fsm_input == 2'b01) begin
                next_state = s0;
                fsm_output = 2'b00;
            end else begin
                next_state = s4;
                fsm_output = 2'b11;
            end
        end
    endcase
end

endmodule
```

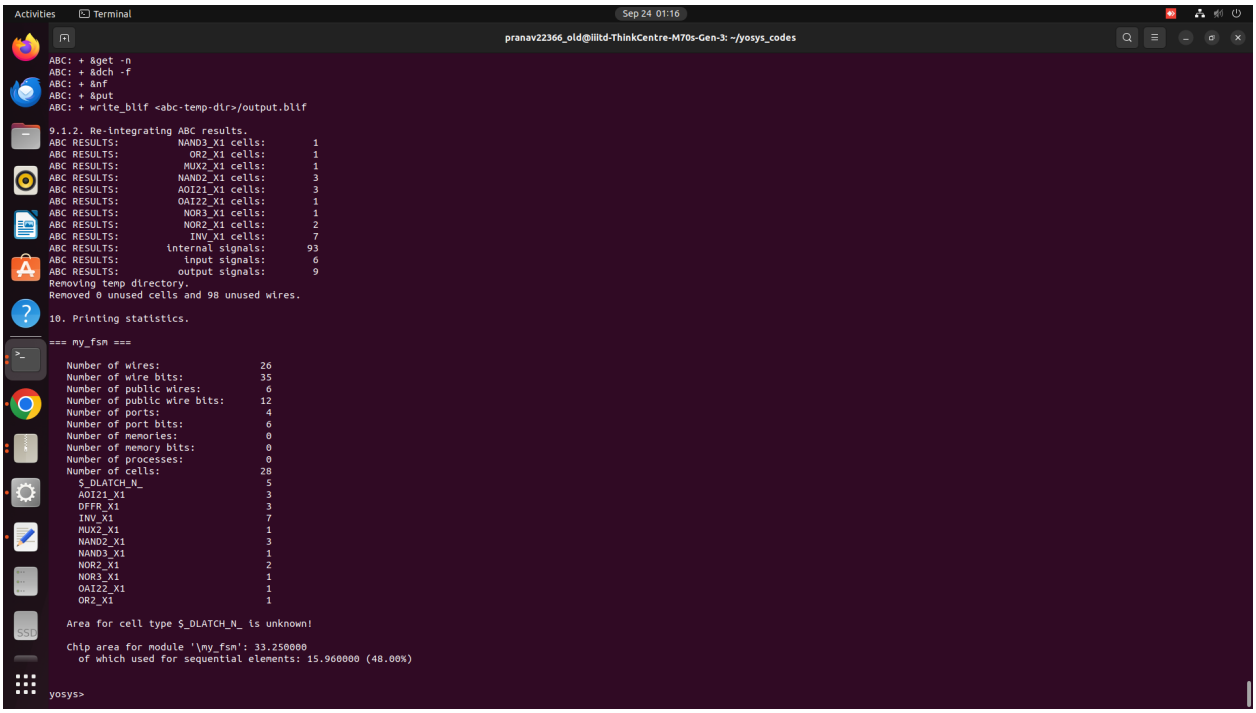


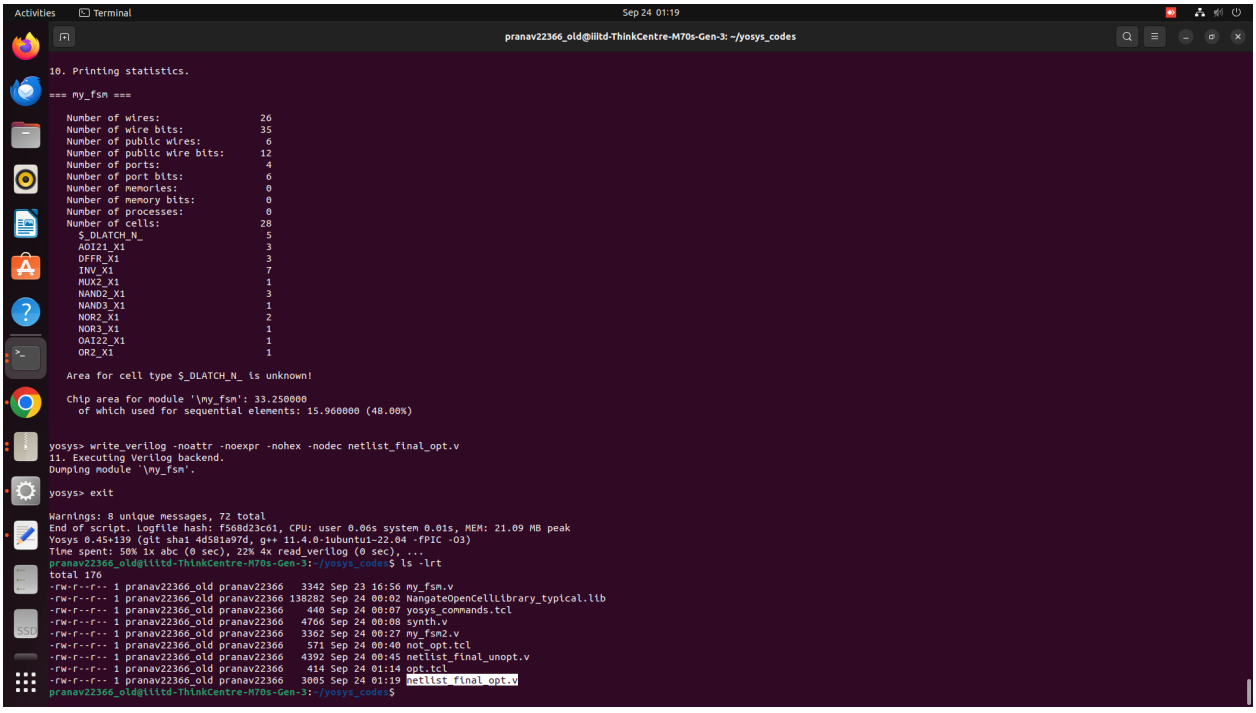

Activities
Terminal
Sep 24 00:48
pranav22366_old@iiitd-ThinkCentre-M70s-Gen-3: ~/yosys_codes
/* Generated by Yosys 0.45+139 (git sha1 4d581a97d, g++ 11.4.0-1ubuntu1~22.04 -fPIC -O3) */
module my_fsm(clk, reset, fsm_input, fsm_output);
wire _00_;
wire _01_;
wire _02_;
wire _03_;
wire _04_;
wire _05_;
wire _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_;
wire _30_;
wire _31_;
wire [2:0] _32_;
wire [1:0] _33_;
input clk;
wire clk;
wire [2:0] current_state;
input [1:0] fsm_input;
wire [1:0] fsm_input;
output [1:0] fsm_output;
wire [1:0] fsm_output;
wire [2:0] next_state;
input reset;
wire reset;
INV_X1 _34_ (
.A(reset),
.ZN(_02_)
);
INV_X1 _35_ (
.A(current_state[2]),
.ZN(_05_)
10,1
Top

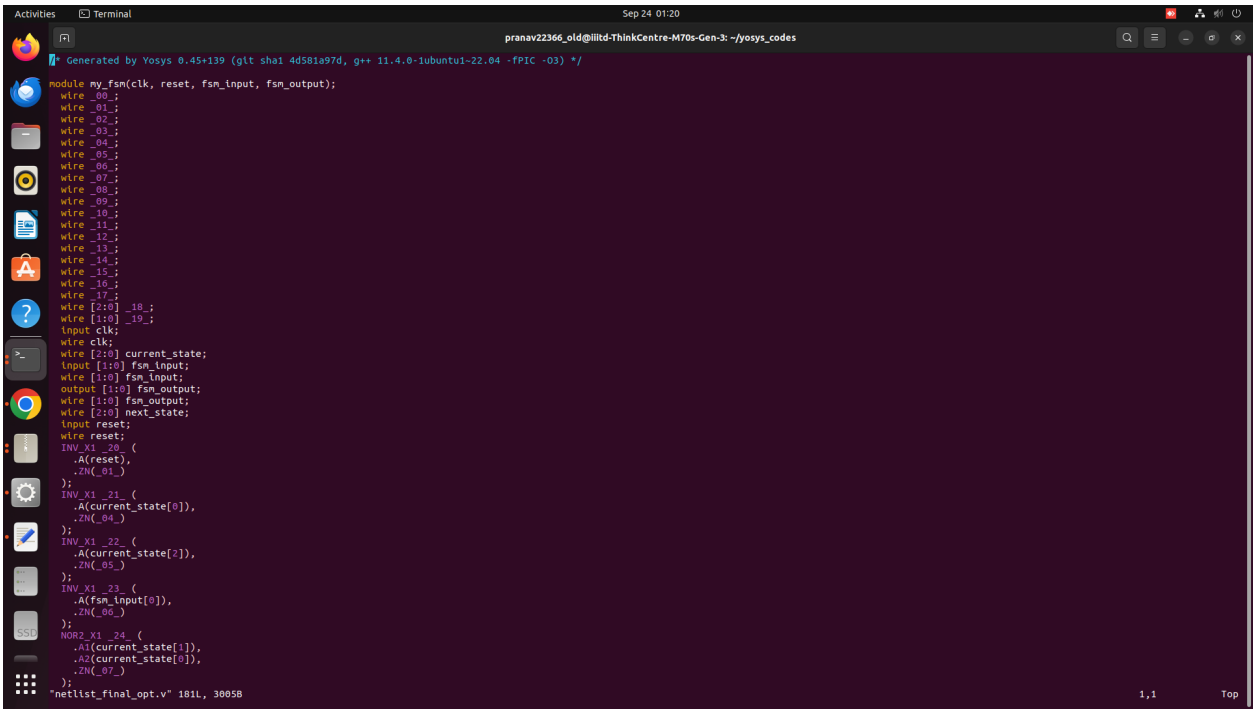

Activities
Terminal
Sep 24 00:48
pranav22366_old@iiitd-ThinkCentre-M70s-Gen-3: ~/yosys_codes
.A2(_09_),
.ZN(_01_)
);
INV_X1 _67_ (
.A(reset),
.ZN(_04_)
);
\$_DLATCH_N_ \fsm_output_reg[0] /* _68_ */ (
.D(_33_[0]),
.E(_00_),
.Q(fsm_output[0])
);
\$_DLATCH_N_ \fsm_output_reg[1] /* _69_ */ (
.D(_33_[1]),
.E(_00_),
.Q(fsm_output[1])
);
\$_DLATCH_N_ \next_state_reg[0] /* _70_ */ (
.D(_32_[0]),
.E(_01_),
.Q(next_state[0])
);
\$_DLATCH_N_ \next_state_reg[1] /* _71_ */ (
.D(_32_[1]),
.E(_01_),
.Q(next_state[1])
);
\$_DLATCH_N_ \next_state_reg[2] /* _72_ */ (
.D(_32_[2]),
.E(_01_),
.Q(next_state[2])
);
DFFR_X1 _73_ (
.CK(clk),
.D(next_state[0]),
.Q(current_state[0]),
.QN(_30_),
.RN(_03_)
);
DFFR_X1 _74_ (
.CK(clk),
.D(next_state[1]),
.Q(current_state[1]),
.QN(_31_),
.RN(_02_)
);
DFFR_X1 _75_ (
.CK(clk),
.D(next_state[2]),
.Q(current_state[2]),
.QN(_29_),
.RN(_04_)
);
endmodule
274,1
Bot


Activities
Terminal
Sep 24 01:15
freepdk45-v14.tar
freepdk45-for-3d-ic-v11.tar
pranav22366_old@iiitd-ThinkCentre-M70s-Gen-3: ~/yosys_co...
pranav22366_old@iiitd-ThinkCentre-M70s-Gen-3: ~/yosys_codes
clean
stat -liberty NangateOpenCellLibrary_typical.lib
write_verilog -noattr -noexpr -nohex -nodec netlist_final_opt.v
#read modules from verilog file
read_verilog my_fsm2.v
#process-> netlist
proc
#remove unused cells and wires
clean
#show netlist
show
#perform optimisation
opt
share -aggressive
show
techmap
dfflibmap -liberty NangateOpenCellLibrary_typical.lib
abc -liberty NangateOpenCellLibrary_typical.lib
clean
stat -liberty NangateOpenCellLibrary_typical.lib
write_verilog -noattr -noexpr -nohex -nodec netlist_final_opt.v
Number of port bits: 6
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 28
$_DLATCH_N_ 5
AOI21_X1 3
DFFR_X1 3
INV_X1 7
MUX2_X1 1
NAND2_X1 3
NAND3_X1 1
NOR2_X1 2
NOR3_X1 1
OAI22_X1 1
OR2_X1 1
Area for cell type $_DLATCH_N_ is unknown!
Chip area for module '\my_fsm': 33.250000
of which used for sequential elements: 15.960000 (48.00%)
yosys>
Number of cells: 704
DFFRNQ 16
INV 234
MUX2 16
NAND2 438
Chip area for module '\top': 7546.000000
yosys>
12:56 / 14:46
Logic Optimization using Yosys
Unlisted
All
For you
Recently uploaded


Activities
Terminal
Sep 24 01:17
pranav22366_old@iiitd-ThinkCentre-M70s-Gen-3: ~/yosys_codes
ABC RESULTS: NAND3_X1 cells: 7
ABC RESULTS: OAI211_X1 cells: 2
ABC RESULTS: NOR2_X1 cells: 6
ABC RESULTS: INV_X1 cells: 7
ABC RESULTS: internal signals: 244
ABC RESULTS: input signals: 6
ABC RESULTS: output signals: 10
Removing temp directory.
Removed 0 unused cells and 213 unused wires.
7. Printing statistics.
=== my_fsm ===
Number of wires: 40
Number of wire bits: 49
Number of public wires: 6
Number of public wire bits: 12
Number of ports: 4
Number of port bits: 6
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 42
$_DLATCH_N_ 5
AND2_X1 2
DFFR_X1 3
INV_X1 7
MUX2_X1 1
NAND2_X1 2
NAND3_X1 7
NAND4_X1 1
NOR2_X1 6
OAI211_X1 2
OAI21_X1 2
OR2_X1 1
OR3_X1 1
XNOR2_X1 2
Area for cell type $_DLATCH_N_ is unknown!
Chip area for module '\my_fsm': 49.210000
of which used for sequential elements: 15.960000 (32.43%)
yosys> write_verilog -noattr -noexpr -nohex -nodec netlist_final_unopt.v
8. Executing Verilog backend.
Dumping module `\my_fsm'.
yosys> exit
Warnings: 8 unique messages, 72 total
End of script. Logfile hash: a7de15e569, CPU: user 0.04s system 0.01s, MEM: 20.31 MB peak
Yosys 0.45+139 (git sha1 4d581a97d, g++ 11.4.0-1ubuntu1~22.04 -fPIC -O3)
Time spent: 59% 1x abc (0 sec), 16% 3x read_verilog (0 sec), ...


Activities
Terminal
Sep 24 01:16
pranav22366_old@iiitd-ThinkCentre-M70s-Gen-3: ~/yosys_codes
ABC: + &get -n
ABC: + &dch -f
ABC: + &nf
ABC: + &put
ABC: + write_blif <abc-temp-dir>/output.blif
9.1.2. Re-integrating ABC results.
ABC RESULTS: NAND3_X1 cells: 1
ABC RESULTS: OR2_X1 cells: 1
ABC RESULTS: MUX2_X1 cells: 1
ABC RESULTS: NAND2_X1 cells: 3
ABC RESULTS: AOI21_X1 cells: 3
ABC RESULTS: OAI22_X1 cells: 1
ABC RESULTS: NOR3_X1 cells: 1
ABC RESULTS: NOR2_X1 cells: 2
ABC RESULTS: INV_X1 cells: 7
ABC RESULTS: internal signals: 93
ABC RESULTS: input signals: 6
ABC RESULTS: output signals: 9
Removing temp directory.
Removed 0 unused cells and 98 unused wires.
10. Printing statistics.
=== my_fsm ===
Number of wires: 26
Number of wire bits: 35
Number of public wires: 6
Number of public wire bits: 12
Number of ports: 4
Number of port bits: 6
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 28
$_DLATCH_N_ 5
AOI21_X1 3
DFFR_X1 3
INV_X1 7
MUX2_X1 1
NAND2_X1 3
NAND3_X1 1
NOR2_X1 2
NOR3_X1 1
OAI22_X1 1
OR2_X1 1
Area for cell type $_DLATCH_N_ is unknown!
Chip area for module '\my_fsm': 33.250000
of which used for sequential elements: 15.960000 (48.00%)
yosys>


Activities
Terminal
Sep 24 01:19
pranav22366_old@iiitd-ThinkCentre-M70s-Gen-3: ~/yosys_codes
10. Printing statistics.
=== my_fsm ===
Number of wires: 26
Number of wire bits: 35
Number of public wires: 6
Number of public wire bits: 12
Number of ports: 4
Number of port bits: 6
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 28
$_DLATCH_N_ 5
AOI21_X1 3
DFFR_X1 3
INV_X1 7
MUX2_X1 1
NAND2_X1 3
NAND3_X1 1
NOR2_X1 2
NOR3_X1 1
OAI22_X1 1
OR2_X1 1
Area for cell type $_DLATCH_N_ is unknown!
Chip area for module '\my_fsm': 33.250000
of which used for sequential elements: 15.960000 (48.00%)
yosys> write_verilog -noattr -noexpr -nohex -nodec netlist_final_opt.v
11. Executing Verilog backend.
Dumping module `\my_fsm'.
yosys> exit
Warnings: 8 unique messages, 72 total
End of script. Logfile hash: f568d23c61, CPU: user 0.06s system 0.01s, MEM: 21.09 MB peak
Yosys 0.45+139 (git sha1 4d581a97d, g++ 11.4.0-1ubuntu1~22.04 -fPIC -O3)
Time spent: 50% 1x abc (0 sec), 22% 4x read_verilog (0 sec), ...
pranav22366_old@iiitd-ThinkCentre-M70s-Gen-3:~/yosys_codes$ ls -lrt
total 176
-rw-r--r-- 1 pranav22366_old pranav22366 3342 Sep 23 16:56 my_fsm.v
-rw-r--r-- 1 pranav22366_old pranav22366 138282 Sep 24 00:02 NangateOpenCellLibrary_typical.lib
-rw-r--r-- 1 pranav22366_old pranav22366 440 Sep 24 00:07 yosys_commands.tcl
-rw-r--r-- 1 pranav22366_old pranav22366 4766 Sep 24 00:08 synth.v
-rw-r--r-- 1 pranav22366_old pranav22366 3362 Sep 24 00:27 my_fsm2.v
-rw-r--r-- 1 pranav22366_old pranav22366 571 Sep 24 00:40 not_opt.tcl
-rw-r--r-- 1 pranav22366_old pranav22366 4392 Sep 24 00:45 netlist_final_unopt.v
-rw-r--r-- 1 pranav22366_old pranav22366 414 Sep 24 01:14 opt.tcl
-rw-r--r-- 1 pranav22366_old pranav22366 3005 Sep 24 01:19 netlist_final_opt.v
pranav22366_old@iiitd-ThinkCentre-M70s-Gen-3:~/yosys_codes$



The one hot encoding code just changes the 3 bit states to 5 bit states . The synthesis file for the one hot encoding file has more number of flip flops as number of bits of state has increased from 3 to 5.

Example in the synthesis file:

```
verilog
Copy code
// Binary Encoding
DFFR_X1 _65_ ( .CK(clk), .D(next_state[0]), .Q(current_state
[0]), .RN(reset));
DFFR_X1 _66_ ( .CK(clk), .D(next_state[1]), .Q(current_state
[1]), .RN(reset));

// One-Hot Encoding
DFFR_X1 _67_ ( .CK(clk), .D(state1), .Q(current_state[1]), .R
N(reset));
DFFR_X1 _68_ ( .CK(clk), .D(state2), .Q(current_state[2]), .R
N(reset));
```

```
DFFR_X1 _69_ ( .CK(clk), .D(state3), .Q(current_state[3]), .R
N(reset));
```

Flip-Flop Usage: One-hot encoding increases the number of flip-flops because each state gets its own flip-flop, while binary encoding minimizes flip-flop usage.

Reset Logic:Also , now the reset logic has to be changed to ensure all flip flops except one are reset.The reset logic will now have to initialize N−1 flip-flops to 0 and one flip-flop to 1.

Synthesis file has extra INV gates to reset flip flops .

Example:

// Binary Encoding
INV_X1
30 (.A(reset), .ZN(*04*));

// One-Hot Encoding
INV_X1
30 (.A(reset), .ZN(state_1_reset));

Area:Also the difference in chip area due to the difference in chip area : 33.25 and 48 .