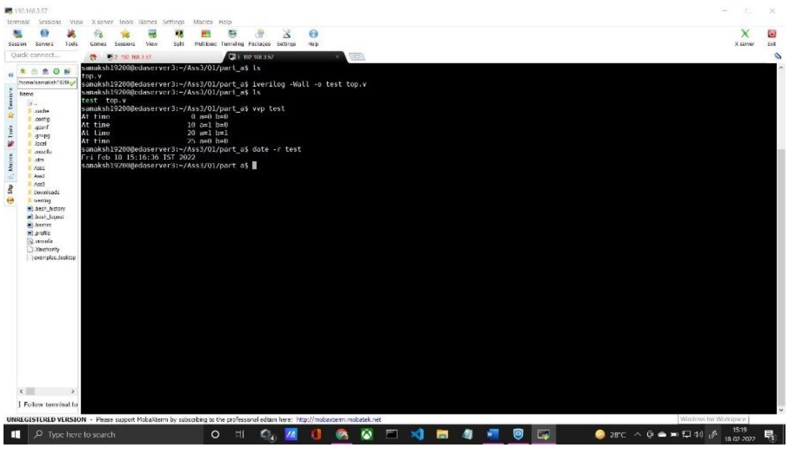
VDF ASSIGNMENT 3

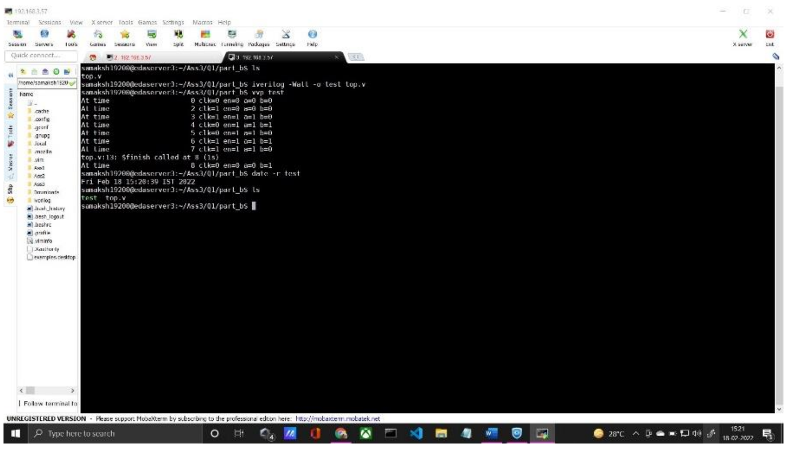
Samaksh Gupta

2019200

1. Log files require root privileges, which we don't have.



```
samaksh19200@edaserver3:~/Ass3/Q1/part_a$ ls
top.v
samaksh19200@edaserver3:~/Ass3/Q1/part_a$ iverilog -Wall -o test top.v
samaksh19200@edaserver3:~/Ass3/Q1/part_a$ ls
test  top.v
samaksh19200@edaserver3:~/Ass3/Q1/part_a$ vvp test
At time                    0 a=0 b=0
At time                   10 a=1 b=0
At time                   20 a=1 b=1
At time                   25 a=0 b=0
samaksh19200@edaserver3:~/Ass3/Q1/part_a$ date -r test
Fri Feb 18 15:16:36 IST 2022
samaksh19200@edaserver3:~/Ass3/Q1/part_a$ 
```



```
samaksh19200@edaserver3:~/Ass3/Q1/part_b$ ls
top.v
samaksh19200@edaserver3:~/Ass3/Q1/part_b$ iverilog -Wall -o test top.v
samaksh19200@edaserver3:~/Ass3/Q1/part_b$ vvp test
At time                    0 clk=0 en=0 a=0 b=0
At time                    2 clk=1 en=0 a=0 b=0
At time                    3 clk=1 en=1 a=1 b=0
At time                    4 clk=0 en=1 a=1 b=1
At time                    5 clk=0 en=1 a=1 b=0
At time                    6 clk=1 en=1 a=1 b=1
At time                    7 clk=1 en=1 a=1 b=0
top.v:13: $finish called at 8 (1s)
At time                    8 clk=0 en=0 a=0 b=1
samaksh19200@edaserver3:~/Ass3/Q1/part_b$ date -r test
Fri Feb 18 15:20:39 IST 2022
samaksh19200@edaserver3:~/Ass3/Q1/part_b$ ls
test  top.v
samaksh19200@edaserver3:~/Ass3/Q1/part_b$ 
```

The code files were given the name top.v and they contained the code given in Q-2. The 2 top files were present in different directories (part_a and part_b inside Q1 directory).

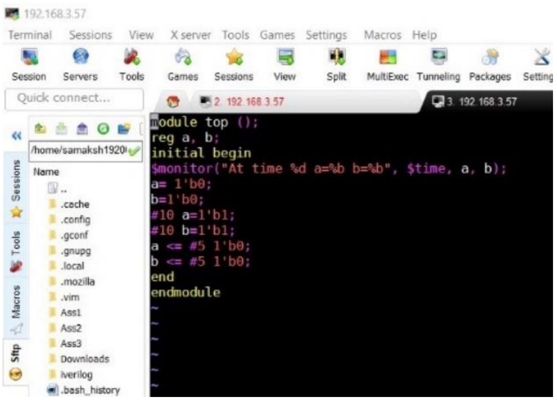
The test file has been generated using iverilog -Wall command as you can see.

The timestamp of the test file is mentioned and you can see the current time in the bottom right of my desktop window. The files in the folder can be seen before and after running the iverilog – Wall -o command.

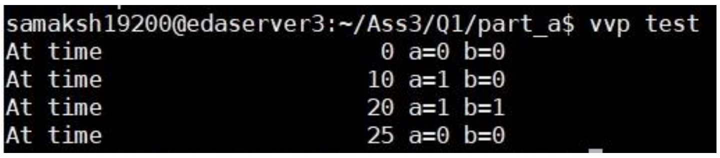
The test file was indeed generated from the command.

2. a

Explanation of the output



```
module top ();
reg a, b;
initial begin
$monitor("At time %d a=%b b=%b", $time, a, b);
a= 1'b0;
b=1'b0;
#10 a=1'b1;
#10 b=1'b1;
a <= #5 1'b0;
b <= #5 1'b0;
end
endmodule
~
~
~
~
~
~
```



```
samaksh19200@edaserver3:~/Ass3/Q1/part_a$ vvp test
At time                    0 a=0 b=0
At time                   10 a=1 b=0
At time                   20 a=1 b=1
At time                   25 a=0 b=0
```

1. Module is created and named top.

2. 2 register type variables are defined ‘a’ and ‘b’.

3. Monitor statement would act as a print command.
    a. It is triggered whenever its placeholders change their value.
    b. % behaves as a placeholder inside the monitor statement.
    c. $time refers to how much time in delay we have given so far.
    d. We want to print at what time, what value ‘a’ and ‘b’ have.
    e. {%d, %b, %b} → {$time, a, b} This is the mapping of placeholders.
    f. 1st placeholder will get $time, 2nd placeholder gets value(a) and 3rd gets value(b).
    g. We see the output in this format only.

4. Now we have initialized ‘a’ and ‘b’ to 0, without giving any delay. Thus, the monitor statement will print its first statement as we can see in the output. Time is 0, ‘a’ and ‘b’ both equal 0.

5. Now we give a delay to 10 units and set ‘a’ to 1 using a blocking statement (‘=’).

    a. Here we see the next printed line. The delay of 10 has caused Time to take the value of 10, ‘a’ has been set to 1, ‘b’ is still 0. This is exactly what is printed.

6. Now we again give a delay of ten units and change ‘b’ to 1 using a blocking statement (‘=’).
    a. Here Time will increment by 10 again (20 now) and ‘a’ already has the value 1. But now ‘b’ is also 1. This information is printed in the 3rd line of the output.

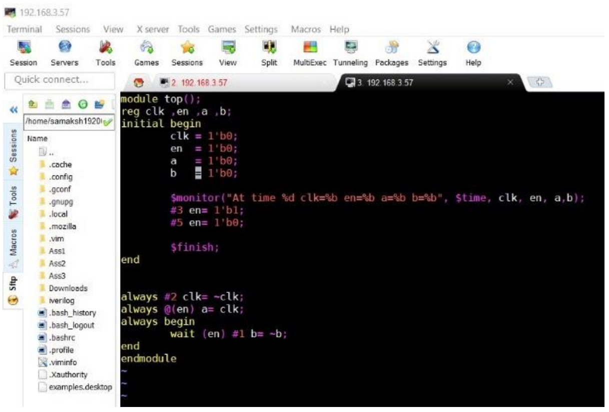
7. Now we are using non-blocking statements (<=).
   ‘a’ and ‘b’ are both being assigned to 0 after a delay of 5 units each, so the time should increment by 5 units twice and we should see statements containing this info
   Time 25, a=0, b=1
   Time 30, a=0, b=0
   However, that is not the case. In the case of non-blocking statements, we don’t wait in one line for the delay. Non-blocking makes both the lines execute together with a single delay of 5 units. Hence, a and b both become 0 together after a delay of 5 units only and we only print one line (last line) containing the information
   Time=25, a=0, b=0

8. We end the initial begin block using the ‘end’ keyword.

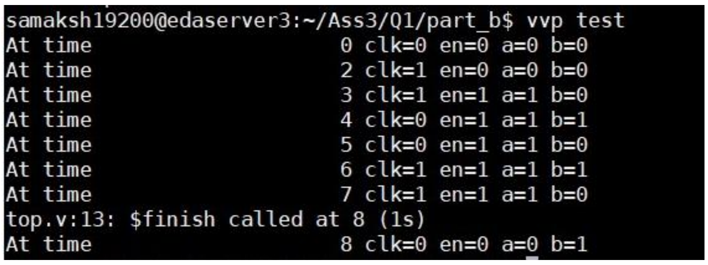
9. We end the module using the ‘endmodule’ keyword.

2. b

<u>Explanation of the output</u>



```
module top();
reg clk ,en ,a ,b;
initial begin
        clk = 1'b0;
        en  = 1'b0;
        a   = 1'b0;
        b   = 1'b0;

        $monitor("At time %d clk=%b en=%b a=%b b=%b", $time, clk, en, a,b);
        #3 en= 1'b1;
        #5 en= 1'b0;

        $finish;
end


always #2 clk= ~clk;
always @(en) a= clk;
always begin
        wait (en) #1 b= ~b;
end
endmodule
~
~
~
```



```
samaksh19200@edaserver3:~/Ass3/Q1/part_b$ vvp test
At time                    0 clk=0 en=0 a=0 b=0
At time                    2 clk=1 en=0 a=0 b=0
At time                    3 clk=1 en=1 a=1 b=0
At time                    4 clk=0 en=1 a=1 b=1
At time                    5 clk=0 en=1 a=1 b=0
At time                    6 clk=1 en=1 a=1 b=1
At time                    7 clk=1 en=1 a=1 b=0
top.v:13: $finish called at 8 (1s)
At time                    8 clk=0 en=0 a=0 b=1
```

Module has been named top, and we have declared 4 register type variables: clk, en, a, b

Initial begin block is created and we initialize the value of the 4 registers to 0, one by one.

Again there is a monitor statement with 5 placeholders this time. (Monitor statement working and placeholders are explained in part a, I won't do it again here). Whenever the value of any of the variables that are associated with the placeholder changes, the monitor statement is triggered. The first statement to print is when everything is 0.

Next, we give a delay of 3units and change en to 1, keeping everything else 0. However, there is also an always block that flips clk signal (0→1 or 1→) after every 2 units of delay. So, that would trigger during the delay of 3 units is going on. This would cause the second line to be printed at time=2 units and clk=1, keeping everything else at 0.

After 1 unit of time more, the delay of 3 units will be reached and 'en' would take the value 1. Because of 'en' changing from 0 to 1, the always block will be triggered which would make (a=clk) 'a' take the value 1 as well. So, both 'en' and 'a' will go to 1 together in the same time unit of 3. Hence, the third line is printed with Time=3, clk=en=a=1, b=0. **From here the countdown for #5 has begun and after 5 units of delay 'en' will change to 0. That is at Time 8, en will go to 0.**

After 1 more time unit, we would have completed 2 time units since the last time clk was flipped, as a result, clk will flip again at Time=4 units and become 0. Not only the clk, but 'b' would become 1 as well. Since 'en' was 1 at the 3rd Time unit, the last always block's condition is met and we wait for 1 time unit since 3 units and then flip ‘b’. Thus at 4th Time unit ‘b’ went to 1 as well. Time=4, clk=0, en=1, a=1, b=1.

Since ‘en’ is 1 we waited one more time unit and reversed ‘b’ to 0. Hence we get the next line printed. Time=5, clk=0, en=1, a=1, b=0

It has been 2 time units since the last time clk was flipped. Thus clk flips again at T=6 and clk is now 1. Since en is still 1, and 1-time unit has passed, ‘b’ will flip again to 1. Thus we get the next statement Time=6, clk=1, en=1, a=1, b=1.

Since en is 1 we waited 1 more time unit and reversed ‘b’ to 0. Hence we get the next line printed. Time=7, clk=1, en=1, a=1, b=0.

At Time 7 ‘en’ was still 1, hence we wait for 1-time unit more and make b=1 at Time 8. We reach Time=8 units and since all delays are over we print the finish statement but at the same time the #5 has completed and ‘en’ should go to zero now. Also, since it has been 2 units of time since clk last flipped, clk will flip again and go to zero. Thus the transitions happening here are → en (1→0), b(0→1), clk(1→0), a(1→0). The final line is now printed at Time=8, clk=0, en=0, a=1, b=0. Since, en is now zero; wait(en) will be false and since there are no more lines to be executed, en can never flip. Moreover, we have reached $finish, and code has completed now.

3)

I have assumed LR to be one signal as input and Speed to be one output.

LR is of 2 bits ➔ LR=10 means that L is input, LR=01 means R is input.

If 11 or 00 is received, then we remain in the same state.

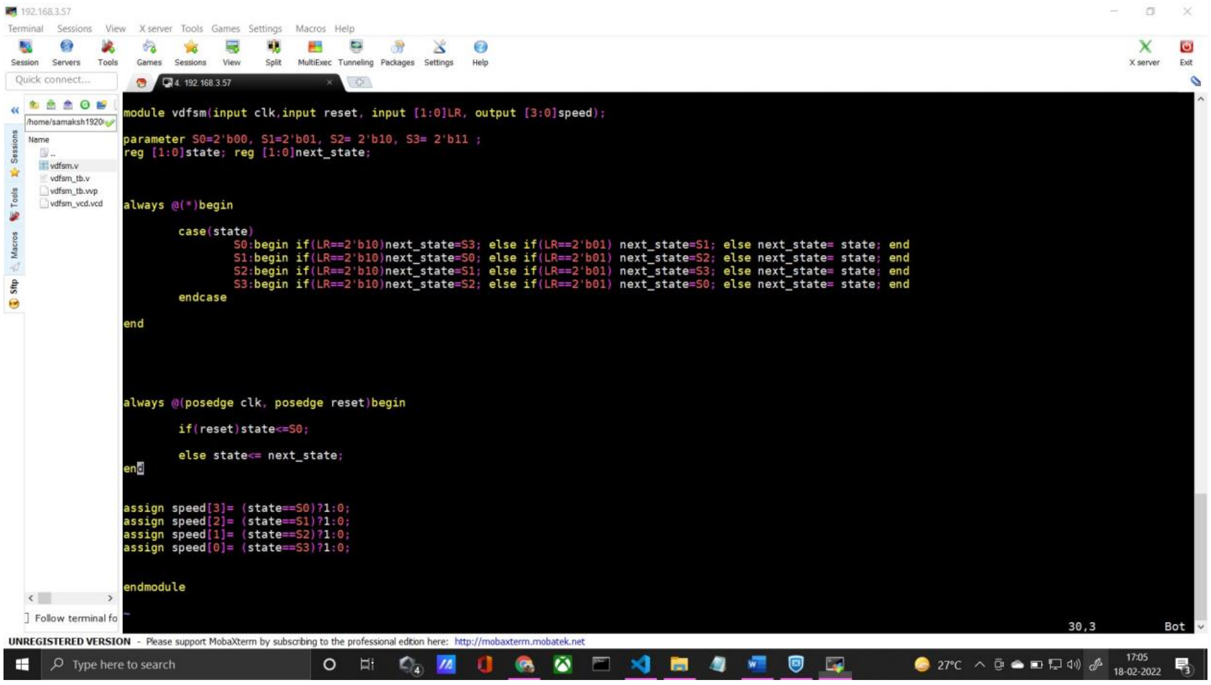
Speed is 4 bits➔ Speed = 1000 means Speed_0 is output, Speed= 0100 means Speed_1 is output, Speed= 0010 means Speed_2 is output, Speed= 0001 means speed_3 is output.

Also, I am assuming that we begin from S_0 state and I have given an external reset signal for this. Reset is high in the beginning so that we can initialize from S_0.

Clk has been also given as input, whenever the posedge of the clock is detected, we transition to the next_state. The output will only change when the clock arrives depending on what the new current state is. The new current state will be the whatever value next_state variable held just before the positive edge of the clock came. Inputs can come whenever they want, but the transitions can only occur on the positive edge of the clock.

Let's say when clk was 0, we were in State S_0 and we gave 01 as input. This would make next_state variable to go to S_1. However, the output will not change yet as our current state is still S_0. Let's say in the same example the input changes to 10 before the clk edge has arrived. This will cause our next_state to go to S_3 and it will no longer be S_1. When the clk finally comes, the output would Speed_3 and current_state would take S_3. So, the latest input will be the one whose effect will be considered when the clk comes. This way we can overwrite between the clk signals too.

Code

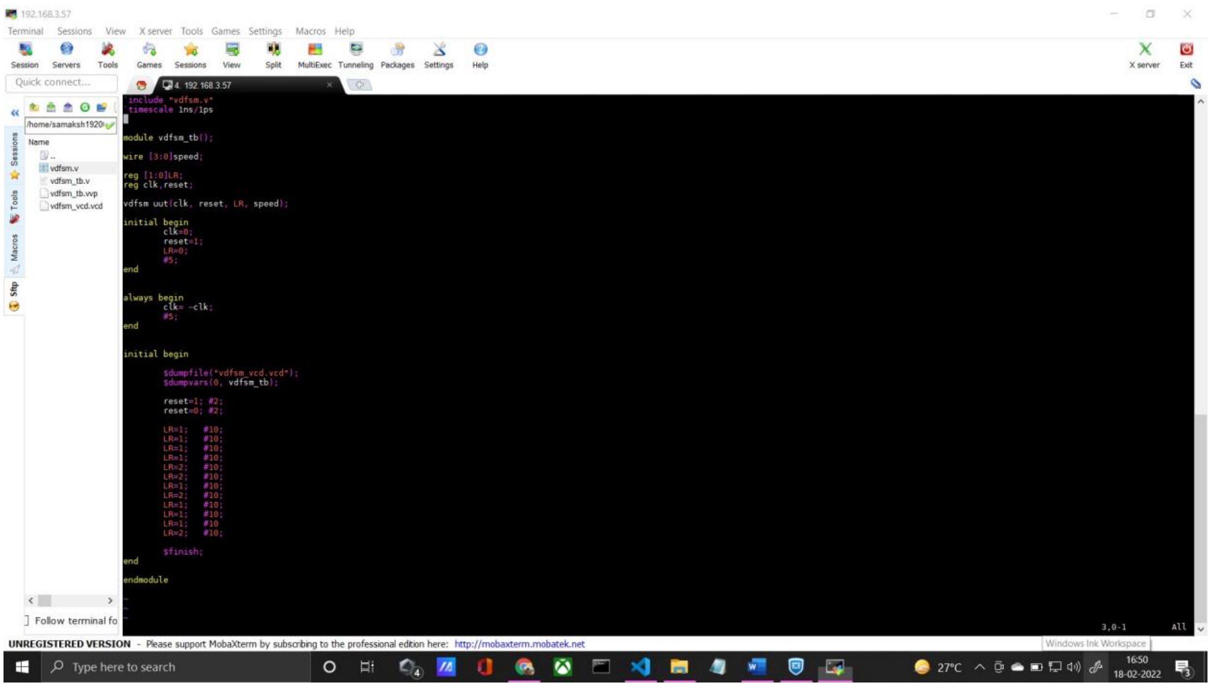


```
module vdfsm(input clk,input reset, input [1:0]LR, output [3:0]speed);

parameter S0=2'b00, S1=2'b01, S2= 2'b10, S3= 2'b11 ;
reg [1:0]state; reg [1:0]next_state;



always @(*)begin

        case(state)
                S0:begin if(LR==2'b10)next_state=S3; else if(LR==2'b01) next_state=S1; else next_state= state; end
                S1:begin if(LR==2'b10)next_state=S0; else if(LR==2'b01) next_state=S2; else next_state= state; end
                S2:begin if(LR==2'b10)next_state=S1; else if(LR==2'b01) next_state=S3; else next_state= state; end
                S3:begin if(LR==2'b10)next_state=S2; else if(LR==2'b01) next_state=S0; else next_state= state; end
        endcase

end




always @(posedge clk, posedge reset)begin

        if(reset)state<=S0;

        else state<= next_state;
end

assign speed[3]= (state==S0)?1:0;
assign speed[2]= (state==S1)?1:0;
assign speed[1]= (state==S2)?1:0;
assign speed[0]= (state==S3)?1:0;


endmodule
~
                                                                                          30,3          Bot
```

192.168.3.57
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help X server Exit
Quick connect...
4. 192.168.3.57
/home/samaksh1920
Name
..
vdfsm.v
vdfsm_tb.v
vdfsm_tb.vvp
vdfsm_vcd.vcd
Follow terminal fo
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net
Type here to search
27°C
17:05
18-02-2022

I know in the case statement we can add a default, but that is handled by reset only… We will never reach any other state. The functionality will not be affected.

Test bench

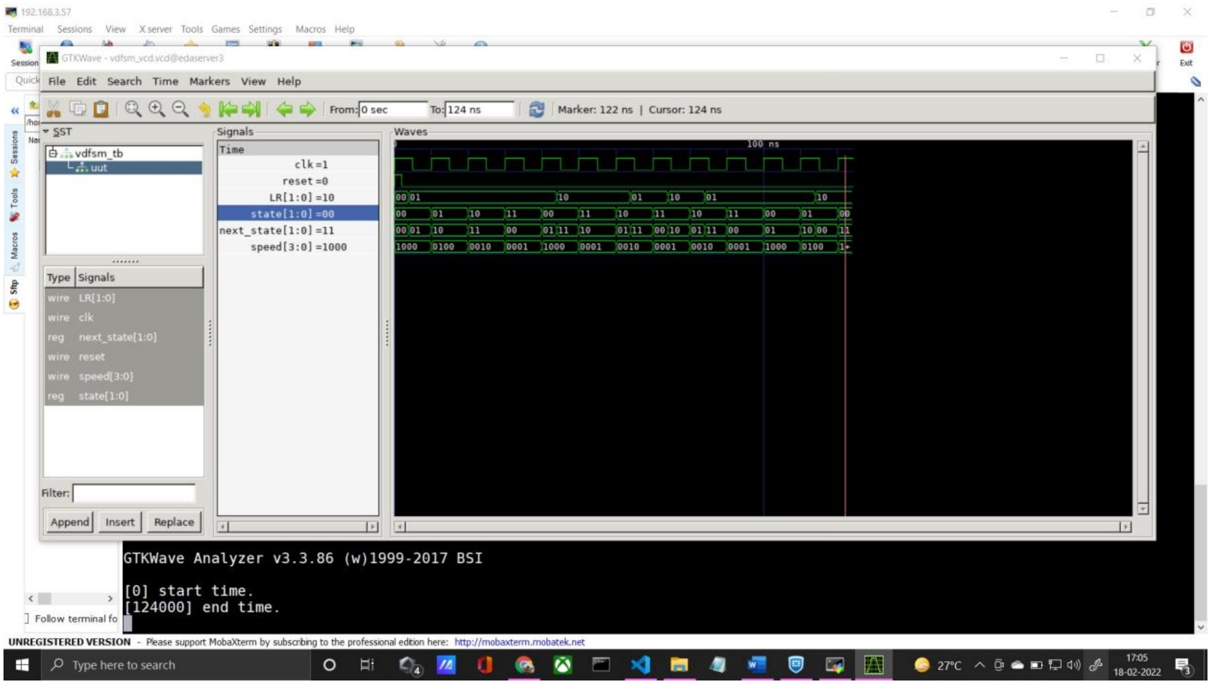


```
`include "vdfsm.v"
`timescale 1ns/1ps

module vdfsm_tb();

wire [3:0]speed;

reg [1:0]LR;
reg clk,reset;

vdfsm uut(clk, reset, LR, speed);

initial begin
        clk=0;
        reset=1;
        LR=0;
        #5;
end


always begin
        clk= ~clk;
        #5;
end


initial begin

        $dumpfile("vdfsm_vcd.vcd");
        $dumpvars(0, vdfsm_tb);

        reset=1; #2;
        reset=0; #2;

        LR=1;   #10;
        LR=1;   #10;
        LR=1;   #10;
        LR=1;   #10;
        LR=2;   #10;
        LR=2;   #10;
        LR=1;   #10;
        LR=2;   #10;
        LR=1;   #10;
        LR=1;   #10;
        LR=1;   #10
        LR=2;   #10;

        $finish;
end

endmodule
```

LR=2 ➔ 10 ➔ Left is the input ||| LR=1 ➔ 01 ➔ Right is the input

Clk has a period of 10ps (It flips after every 5ps). New input is given after a delay of 10ps.

Waveform



```
From: 0 sec   To: 124 ns   Marker: 122 ns | Cursor: 124 ns

Time
              clk=1
            reset=0
         LR[1:0]=10
      state[1:0]=00
 next_state[1:0]=11
      speed[3:0]=1000
```

```
GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

[0] start time.
[124000] end time.
```

The last values (where the red line is) are cut, you can read their values on the left.

Explanation of the waveform

We begin in state S_0 when no input is given and reset is high. The reset is then set to low and we begin checking the functionality of our FSM.

We can see that the first valid input 01 (Right) is given before the clock edge arrives. Hence, the next_state is calculated as S_1, however, the current state remains S_0 and thus the output will not change yet. As soon as the clock comes, the current state will take the value of next_state and output will change depending on what the current state is now. Since on the first clock edge current state would change from S_0 to S_1, the output correctly changes from Speed_0 (1000) to Speed_1 (0100).

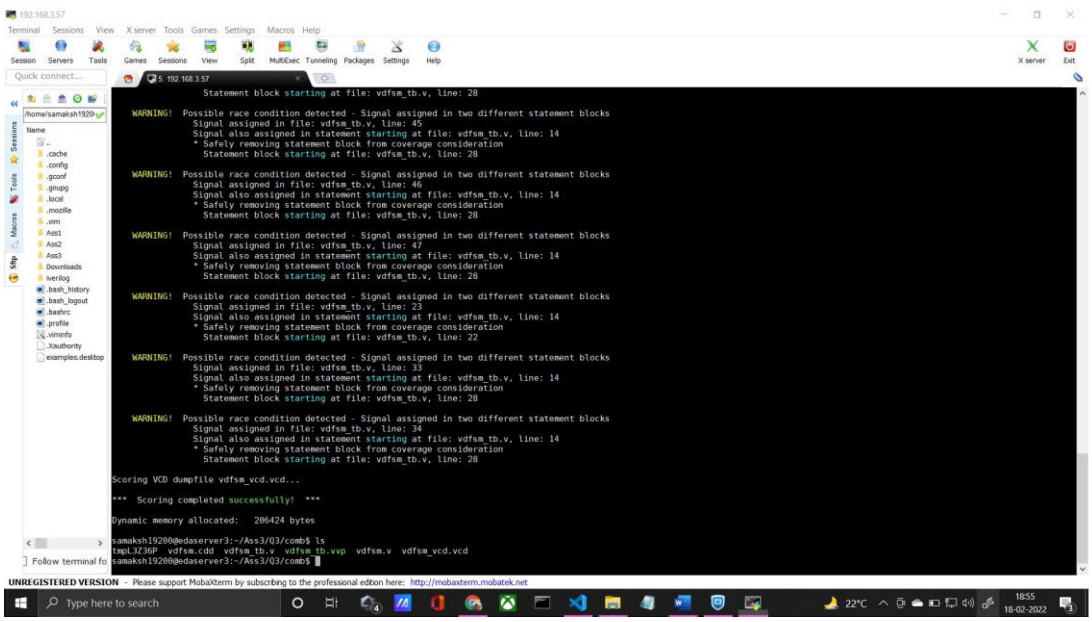
The input is still 01(R), the next_state now becomes S_2. As soon as the next positive edge of the clock comes. The current state which was S_1, takes the value S_2 and output changes from 0100 to 0010 (Speed_2).

The input is still 01(R), the next_state now becomes S_3. As soon as the next positive edge of the clock comes. The current state which was S_2, takes the value S_3 and output changes from 0010 to 0001 (Speed_3).

Similarly, we can see for the input 10(L), the FSM is working correctly.

The key idea is that the output will only change when the positive edge of the clock comes. Reset is asynchronous.

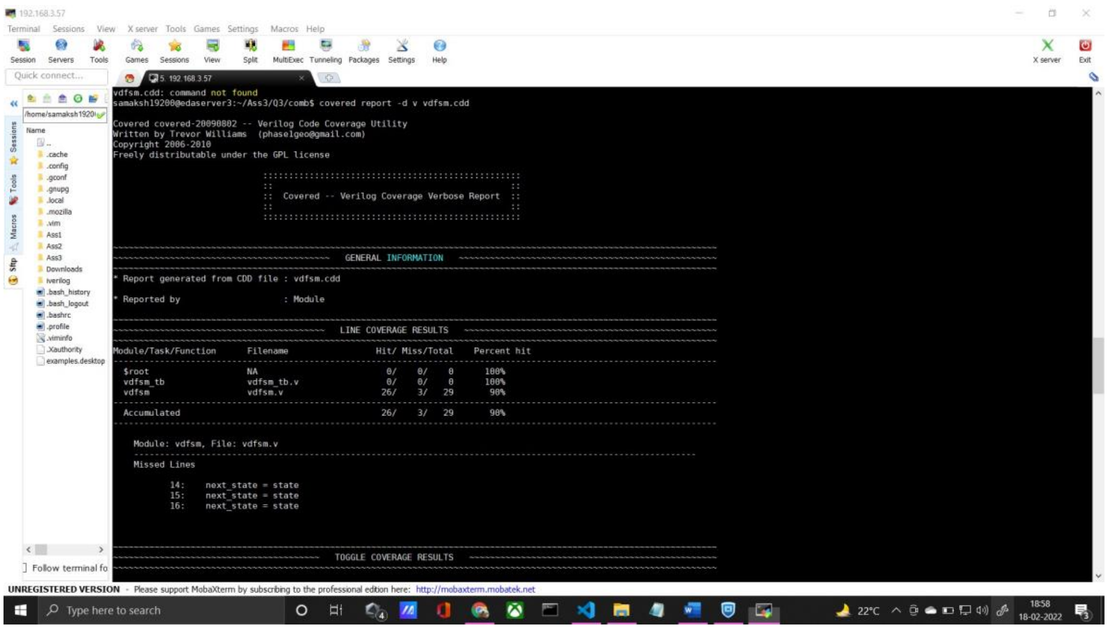
COVERAGE



```
              Statement block starting at file: vdfsm_tb.v, line: 28

    WARNING!  Possible race condition detected - Signal assigned in two different statement blocks
                Signal assigned in file: vdfsm_tb.v, line: 45
                Signal also assigned in statement starting at file: vdfsm_tb.v, line: 14
                * Safely removing statement block from coverage consideration
                  Statement block starting at file: vdfsm_tb.v, line: 28

    WARNING!  Possible race condition detected - Signal assigned in two different statement blocks
                Signal assigned in file: vdfsm_tb.v, line: 46
                Signal also assigned in statement starting at file: vdfsm_tb.v, line: 14
                * Safely removing statement block from coverage consideration
                  Statement block starting at file: vdfsm_tb.v, line: 28

    WARNING!  Possible race condition detected - Signal assigned in two different statement blocks
                Signal assigned in file: vdfsm_tb.v, line: 47
                Signal also assigned in statement starting at file: vdfsm_tb.v, line: 14
                * Safely removing statement block from coverage consideration
                  Statement block starting at file: vdfsm_tb.v, line: 28

    WARNING!  Possible race condition detected - Signal assigned in two different statement blocks
                Signal assigned in file: vdfsm_tb.v, line: 23
                Signal also assigned in statement starting at file: vdfsm_tb.v, line: 14
                * Safely removing statement block from coverage consideration
                  Statement block starting at file: vdfsm_tb.v, line: 22

    WARNING!  Possible race condition detected - Signal assigned in two different statement blocks
                Signal assigned in file: vdfsm_tb.v, line: 33
                Signal also assigned in statement starting at file: vdfsm_tb.v, line: 14
                * Safely removing statement block from coverage consideration
                  Statement block starting at file: vdfsm_tb.v, line: 28

    WARNING!  Possible race condition detected - Signal assigned in two different statement blocks
                Signal assigned in file: vdfsm_tb.v, line: 34
                Signal also assigned in statement starting at file: vdfsm_tb.v, line: 14
                * Safely removing statement block from coverage consideration
                  Statement block starting at file: vdfsm_tb.v, line: 28

Scoring VCD dumpfile vdfsm_vcd.vcd...

***  Scoring completed successfully!  ***

Dynamic memory allocated:   206424 bytes

samaksh19200@edaserver3:~/Ass3/Q3/comb$ ls
tmpL3Z36P  vdfsm.cdd  vdfsm_tb.v  vdfsm_tb.vvp  vdfsm.v  vdfsm_vcd.vcd
samaksh19200@edaserver3:~/Ass3/Q3/comb$
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

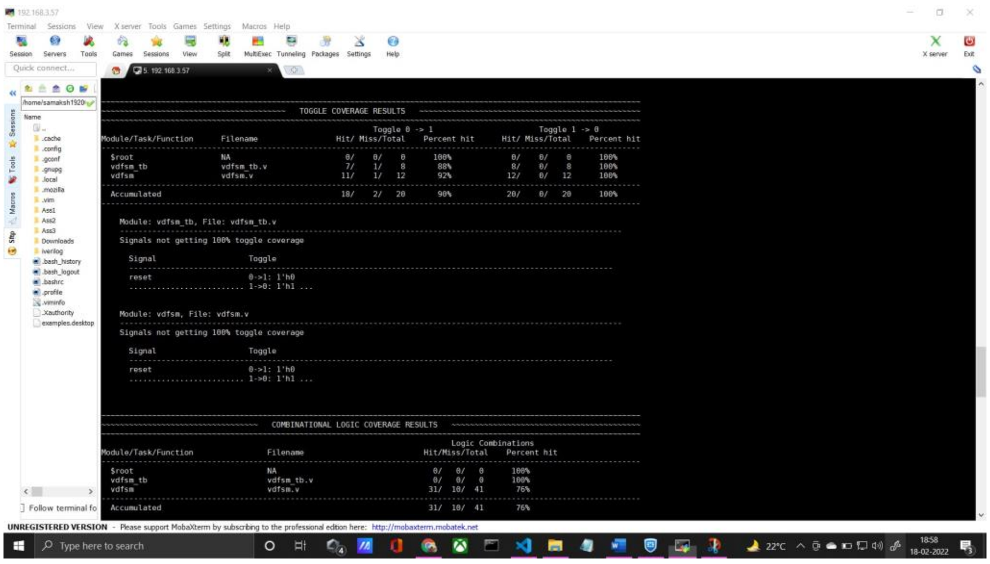
LINE COVERAGE



```
vdfsm.cdd: command not found
samaksh19200@edaserver3:~/Ass3/Q3/comb$ covered report -d v vdfsm.cdd

Covered covered-20090802 -- Verilog Code Coverage Utility
Written by Trevor Williams  (phaselgeo@gmail.com)
Copyright 2006-2010
Freely distributable under the GPL license

                              :::::::::::::::::::::::::::::::::::::::::::::::::
                              ::                                             ::
                              ::  Covered -- Verilog Coverage Verbose Report ::
                              ::                                             ::
                              :::::::::::::::::::::::::::::::::::::::::::::::::


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   GENERAL INFORMATION   ~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Report generated from CDD file : vdfsm.cdd

* Reported by                    : Module

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   LINE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Module/Task/Function      Filename                Hit/ Miss/Total    Percent hit
------------------------------------------------------------------------------------------
  $root                   NA                        0/    0/    0      100%
  vdfsm_tb                vdfsm_tb.v                0/    0/    0      100%
  vdfsm                   vdfsm.v                  26/    3/   29       90%
------------------------------------------------------------------------------------------
  Accumulated                                      26/    3/   29       90%
------------------------------------------------------------------------------------------

    Module: vdfsm, File: vdfsm.v
    --------------------------------------------------------------------------------------
    Missed Lines

           14:    next_state = state
           15:    next_state = state
           16:    next_state = state


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   TOGGLE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Line coverage refers to how many lines in my code were executed. The lines that are not executed are part of the 'else' block. This 'else' condition will never be met as the input can never be '11'. So, the line coverage is 98%. It is a good practice to always include 'else', hence I did it.

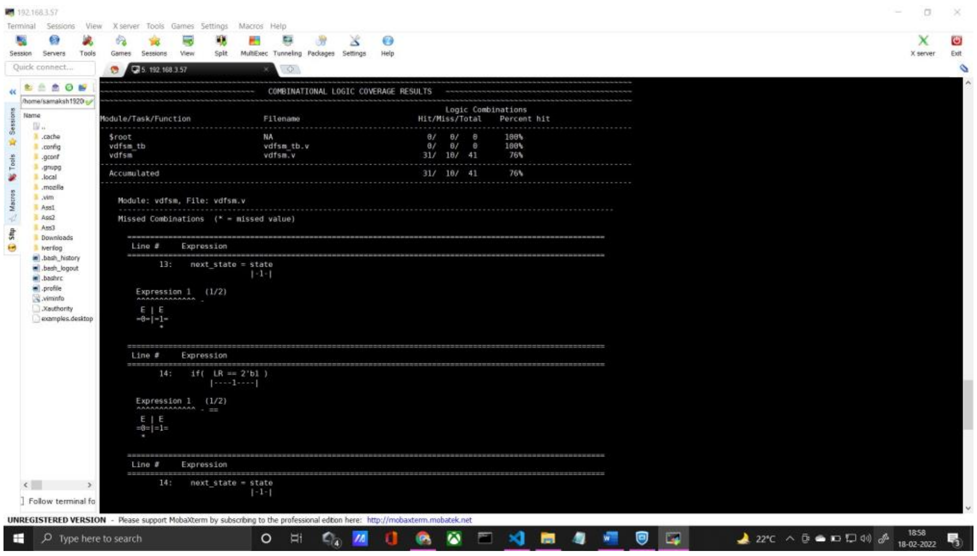
TOGGLE COVERAGE



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   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%
  vdfsm_tb                vdfsm_tb.v                7/    1/    8       88%           8/    0/    8      100%
  vdfsm                   vdfsm.v                  11/    1/   12       92%          12/    0/   12      100%
----------------------------------------------------------------------------------------------------------------
  Accumulated                                      18/    2/   20       90%          20/    0/   20      100%
----------------------------------------------------------------------------------------------------------------

    Module: vdfsm_tb, File: vdfsm_tb.v
    ------------------------------------------------------------------------------------------------------
    Signals not getting 100% toggle coverage

      Signal                    Toggle
      ----------------------------------------------------------------------------------------------------
      reset                     0->1: 1'h0
      ......................... 1->0: 1'h1 ...

    Module: vdfsm, File: vdfsm.v
    ------------------------------------------------------------------------------------------------------
    Signals not getting 100% toggle coverage

      Signal                    Toggle
      ----------------------------------------------------------------------------------------------------
      reset                     0->1: 1'h0
      ......................... 1->0: 1'h1 ...


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   COMBINATIONAL LOGIC COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                                Logic Combinations
Module/Task/Function             Filename                     Hit/Miss/Total    Percent hit
----------------------------------------------------------------------------------------------------------------
  $root                          NA                             0/   0/   0      100%
  vdfsm_tb                       vdfsm_tb.v                     0/   0/   0      100%
  vdfsm                          vdfsm.v                       31/  10/  41       76%
----------------------------------------------------------------------------------------------------------------
  Accumulated                                                  31/  10/  41       76%
```

Toggle coverage measures how much my data has toggled. Since reset was only given 1 toggling from 1 to 0, it has been mentioned explicitly that it isn't being toggled 100%.

COMBINATIONAL LOGIC COVERAGE



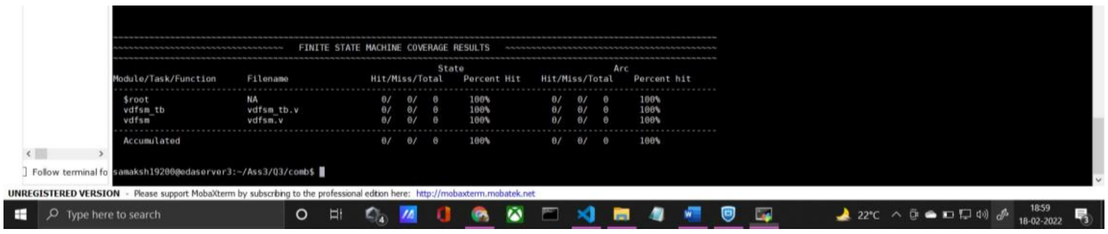
```
                         COMBINATIONAL LOGIC COVERAGE RESULTS
                                                                  Logic Combinations
Module/Task/Function              Filename                      Hit/Miss/Total    Percent hit
---------------------------------------------------------------------------------------------
  $root                           NA                              0/   0/   0       100%
  vdfsm_tb                        vdfsm_tb.v                      0/   0/   0       100%
  vdfsm                           vdfsm.v                        31/  10/  41        76%
---------------------------------------------------------------------------------------------
  Accumulated                                                    31/  10/  41        76%
---------------------------------------------------------------------------------------------

    Module: vdfsm, File: vdfsm.v
    -----------------------------------------------------------------------------------------
    Missed Combinations  (* = missed value)

      ===================================================================================
       Line #     Expression
      ===================================================================================
            13:    next_state = state
                                |-1-|

        Expression 1   (1/2)
        ^^^^^^^^^^^^^ -
         E | E
        =0=|=1=
            *

      ===================================================================================
       Line #     Expression
      ===================================================================================
            14:    if(  LR == 2'b1 )
                        |----1----|

        Expression 1   (1/2)
        ^^^^^^^^^^^^^ - ==
         E | E
        =0=|=1=
          *

      ===================================================================================
       Line #     Expression
      ===================================================================================
            14:    next_state = state
                                |-1-|
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

It measures the unique values our expressions evaluated. I expect this to be below, as we have used 4 bits to show our output which can only take 4 unique values. So, it is obvious that output will not evaluate to the remaining 12 values. Similarly, the input is given by 2 bits, however, the input can take only 2 values. This means again 2 values will never occur in the input stage and as a result, the 'else' condition in the transition logic will never be achieved.

So, the Comb Logic Coverage being around 76% makes sense to me.

FINITE STATE MACHINE COVERAGE

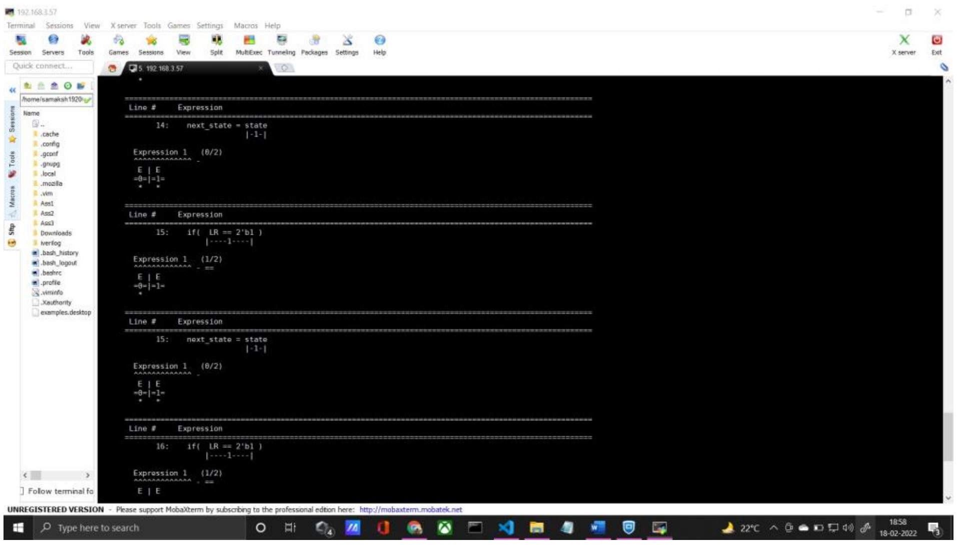


```
                          FINITE STATE MACHINE COVERAGE RESULTS
                                                   State                        Arc
Module/Task/Function      Filename           Hit/Miss/Total   Percent Hit   Hit/Miss/Total   Percent hit
---------------------------------------------------------------------------------------------------------
  $root                   NA                   0/   0/   0       100%          0/   0/   0       100%
  vdfsm_tb                vdfsm_tb.v           0/   0/   0       100%          0/   0/   0       100%
  vdfsm                   vdfsm.v              0/   0/   0       100%          0/   0/   0       100%
---------------------------------------------------------------------------------------------------------
  Accumulated                                  0/   0/   0       100%          0/   0/   0       100%

samaksh19200@edaserver3:~/Ass3/Q3/comb$
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

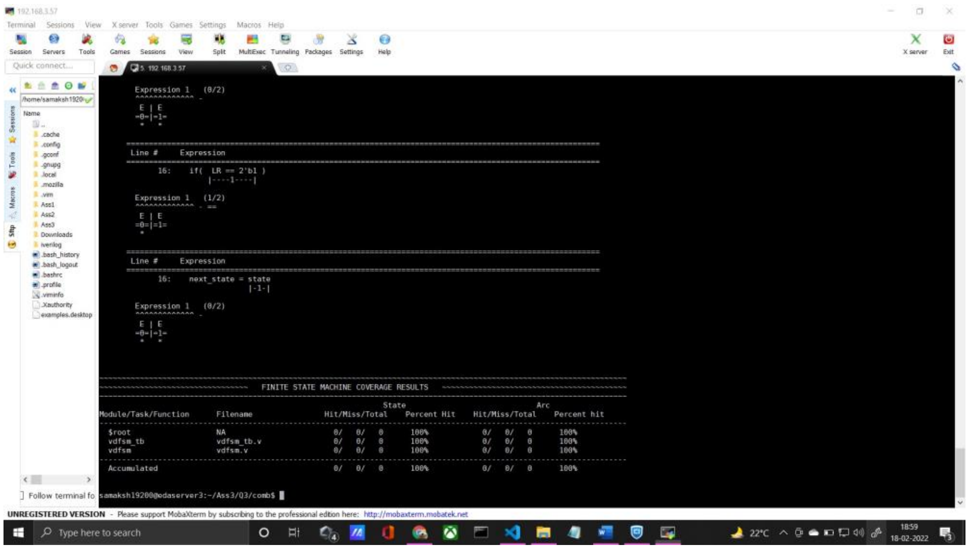
This describes the robustness of my test bench. It answers if I have reached all possible states and taken all the paths possible. Since my state and next_state variables are of 2 bits only and we have just 4 unique states, this coverage being 100% makes sense.

SUGGESTIONS



```
 *

 ================================================================================
  Line #     Expression
 ================================================================================
       14:    next_state = state
                           |-1-|

  Expression 1   (0/2)
  ^^^^^^^^^^^^^^ -
   E | E
  =0=|=1=
   *   *

 ================================================================================
  Line #     Expression
 ================================================================================
       15:    if(  LR == 2'b1 )
                  |----1----|

  Expression 1   (1/2)
  ^^^^^^^^^^^^^^ - ==
   E | E
  =0=|=1=
   *

 ================================================================================
  Line #     Expression
 ================================================================================
       15:    next_state = state
                           |-1-|

  Expression 1   (0/2)
  ^^^^^^^^^^^^^^ -
   E | E
  =0=|=1=
   *   *

 ================================================================================
  Line #     Expression
 ================================================================================
       16:    if(  LR == 2'b1 )
                  |----1----|

  Expression 1   (1/2)
  ^^^^^^^^^^^^^^ - ==
   E | E
```

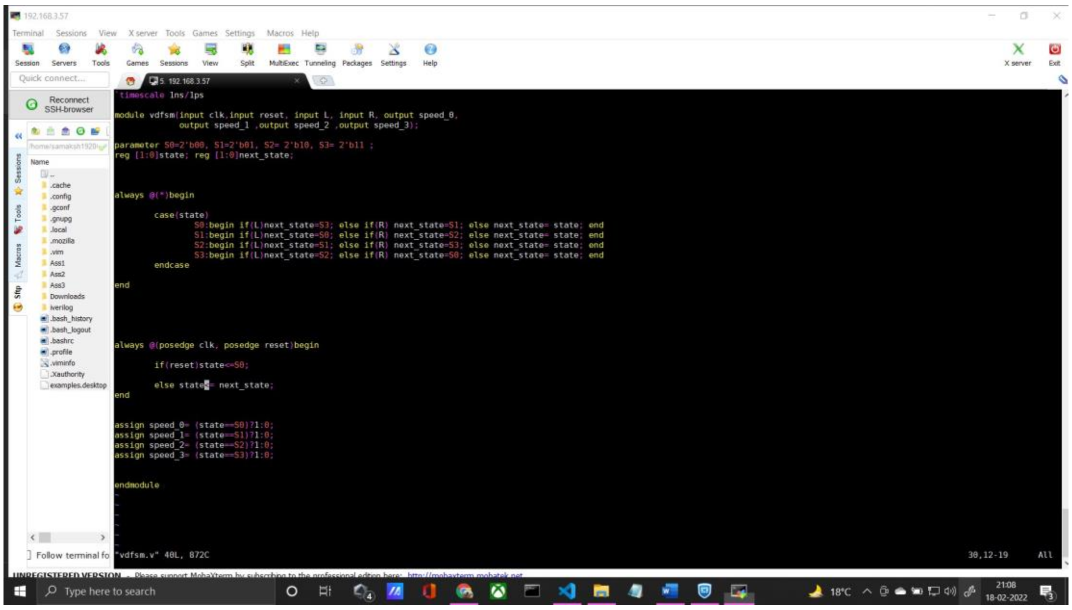
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net



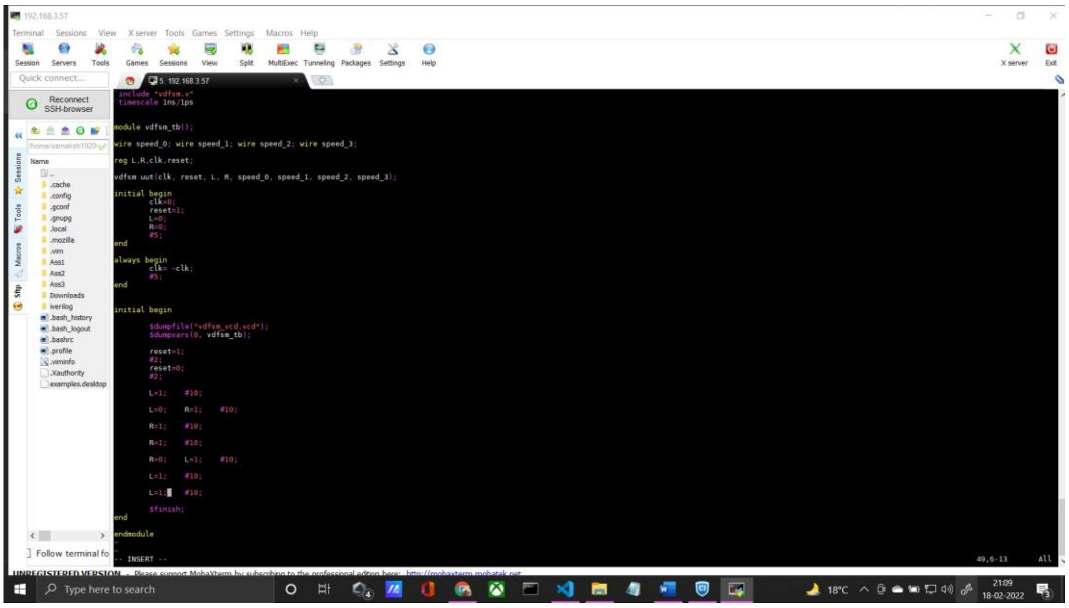
```
  Expression 1   (0/2)
  ^^^^^^^^^^^^^^ -
   E | E
  =0=|=1=
   *   *

 ================================================================================
  Line #     Expression
 ================================================================================
       16:    if(  LR == 2'b1 )
                  |----1----|

  Expression 1   (1/2)
  ^^^^^^^^^^^^^^ - ==
   E | E
  =0=|=1=
   *

 ================================================================================
  Line #     Expression
 ================================================================================
       16:    next_state = state
                           |-1-|

  Expression 1   (0/2)
  ^^^^^^^^^^^^^^ -
   E | E
  =0=|=1=
   *   *


-----------------------------------------------------------------------------------------------
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   FINITE STATE MACHINE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-----------------------------------------------------------------------------------------------
                                                     State                               Arc
Module/Task/Function      Filename                Hit/Miss/Total    Percent Hit    Hit/Miss/Total    Percent hit
-----------------------------------------------------------------------------------------------
  $root                   NA                         0/    0/    0       100%           0/    0/    0       100%
  vdfsm_tb                vdfsm_tb.v                 0/    0/    0       100%           0/    0/    0       100%
  vdfsm                   vdfsm.v                    0/    0/    0       100%           0/    0/    0       100%
-----------------------------------------------------------------------------------------------
  Accumulated                                        0/    0/    0       100%           0/    0/    0       100%

samaksh19200@edaserver3:~/Ass3/Q3/comb$
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

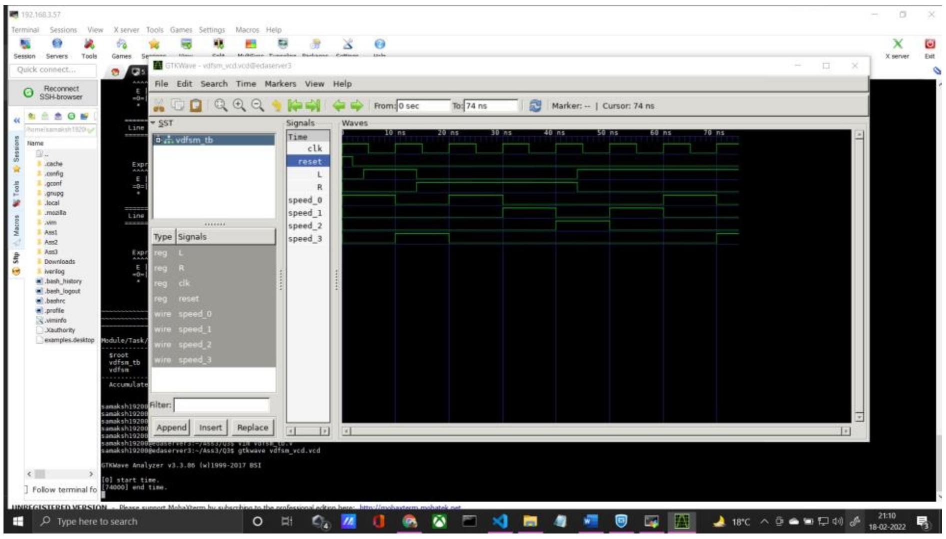
I designed FSM in another way. Here I have kept individual signals for input and output. Just putting in the report because I did it, so I might as well.



```
`timescale 1ns/1ps

module vdfsm(input clk,input reset, input L, input R, output speed_0,
             output speed_1 ,output speed_2 ,output speed_3);

parameter S0=2'b00, S1=2'b01, S2= 2'b10, S3= 2'b11 ;
reg [1:0]state; reg [1:0]next_state;



always @(*)begin

        case(state)
                S0:begin if(L)next_state=S3; else if(R) next_state=S1; else next_state= state; end
                S1:begin if(L)next_state=S0; else if(R) next_state=S2; else next_state= state; end
                S2:begin if(L)next_state=S1; else if(R) next_state=S3; else next_state= state; end
                S3:begin if(L)next_state=S2; else if(R) next_state=S0; else next_state= state; end
        endcase

end




always @(posedge clk, posedge reset)begin

        if(reset)state<=S0;

        else state<= next_state;
end


assign speed_0= (state==S0)?1:0;
assign speed_1= (state==S1)?1:0;
assign speed_2= (state==S2)?1:0;
assign speed_3= (state==S3)?1:0;


endmodule
~
~
~
~
~
~
"vdfsm.v" 40L, 872C                                                    30,12-19      All
```

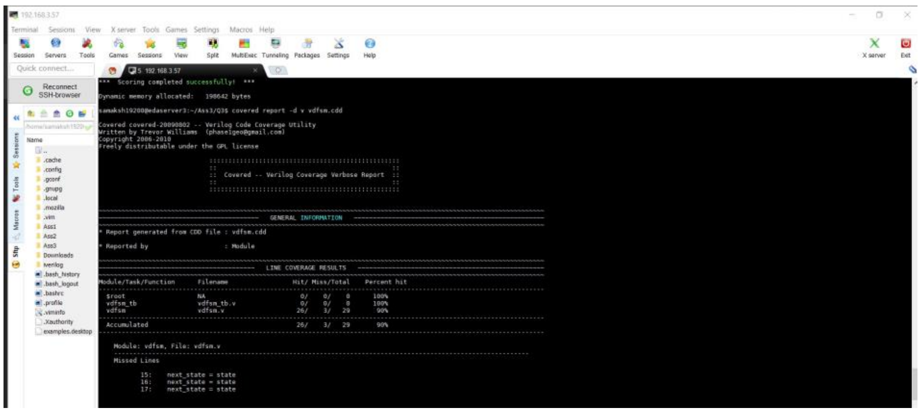


```
`include "vdfsm.v"
`timescale 1ns/1ps

module vdfsm_tb();

wire speed_0; wire speed_1; wire speed_2; wire speed_3;

reg L,R,clk,reset;

vdfsm uut(clk, reset, L, R, speed_0, speed_1, speed_2, speed_3);

initial begin
        clk=0;
        reset=1;
        L=0;
        R=0;
        #5;
end

always begin
        clk= ~clk;
        #5;
end


initial begin

        $dumpfile("vdfsm_vcd.vcd");
        $dumpvars(0, vdfsm_tb);

        reset=1;
        #2;
        reset=0;
        #2;

        L=1;    #10;

        L=0;    R=1;    #10;

        R=1;    #10;

        R=1;    #10;

        R=0;    L=1;    #10;

        L=1;    #10;

        L=1;    #10;

        $finish;
end

endmodule
~
~
-- INSERT --                                                           49,6-13       All
```

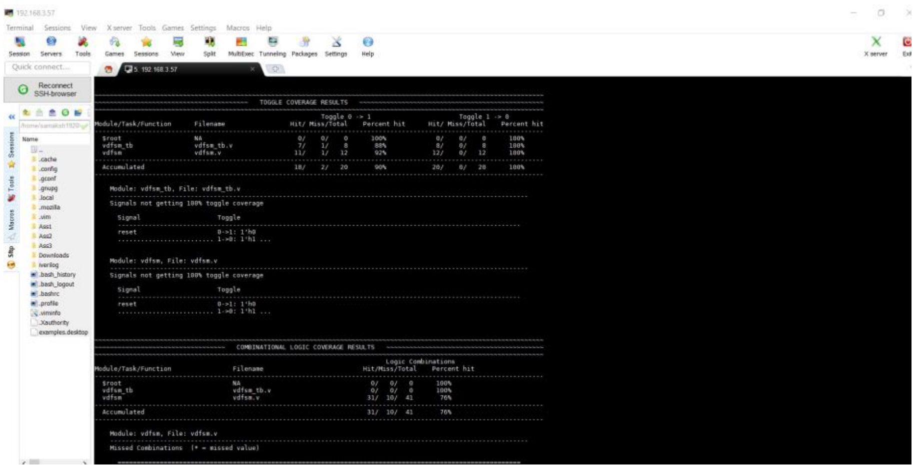


```
samaksh19200@edaserver3:~/Ass3/Q3$ vim vdfsm_tb.v
samaksh19200@edaserver3:~/Ass3/Q3$ gtkwave vdfsm_vcd.vcd

GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

[0] start time.
[74000] end time.
```

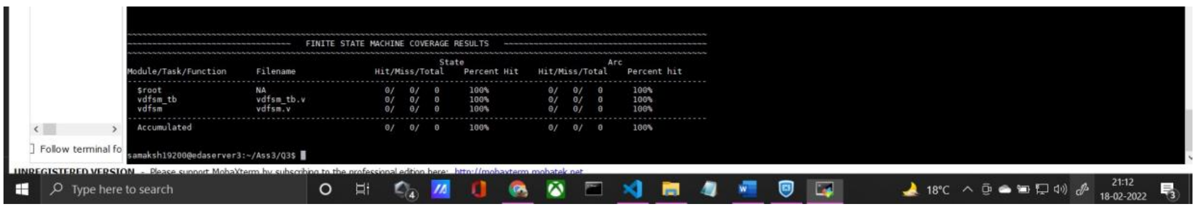
COVERAGE



```
***  Scoring completed successfully!  ***

Dynamic memory allocated:   198642 bytes

samaksh19200@edaserver3:~/Ass3/Q3$ covered report -d v vdfsm.cdd

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 : vdfsm.cdd

* Reported by                    : Module

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   LINE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Module/Task/Function      Filename                Hit/ Miss/Total    Percent hit
---------------------------------------------------------------------------------------------------
  $root                   NA                        0/    0/    0      100%
  vdfsm_tb                vdfsm_tb.v                0/    0/    0      100%
  vdfsm                   vdfsm.v                  26/    3/   29       90%
---------------------------------------------------------------------------------------------------
  Accumulated                                      26/    3/   29       90%
---------------------------------------------------------------------------------------------------

    Module: vdfsm, File: vdfsm.v
    -----------------------------------------------------------------------------------------------
    Missed Lines

          15:    next_state = state
          16:    next_state = state
          17:    next_state = state
```



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   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%
  vdfsm_tb                vdfsm_tb.v                7/    1/    8       88%             8/    0/    8      100%
  vdfsm                   vdfsm.v                  11/    1/   12       92%            12/    0/   12      100%
-------------------------------------------------------------------------------------------------------------------------
  Accumulated                                      18/    2/   20       90%            20/    0/   20      100%
-------------------------------------------------------------------------------------------------------------------------

    Module: vdfsm_tb, File: vdfsm_tb.v
    -------------------------------------------------------------------------------------------------------------------
    Signals not getting 100% toggle coverage

      Signal                    Toggle
      ---------------------------------------------------------------------------------------------------------------
      reset                     0->1: 1'h0
      ......................... 1->0: 1'h1 ...


    Module: vdfsm, File: vdfsm.v
    -------------------------------------------------------------------------------------------------------------------
    Signals not getting 100% toggle coverage

      Signal                    Toggle
      ---------------------------------------------------------------------------------------------------------------
      reset                     0->1: 1'h0
      ......................... 1->0: 1'h1 ...


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   COMBINATIONAL LOGIC COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                                  Logic Combinations
Module/Task/Function                Filename                    Hit/Miss/Total    Percent hit
-------------------------------------------------------------------------------------------------------------------------
  $root                             NA                            0/    0/    0       100%
  vdfsm_tb                          vdfsm_tb.v                    0/    0/    0       100%
  vdfsm                             vdfsm.v                      31/   10/   41        76%
-------------------------------------------------------------------------------------------------------------------------
  Accumulated                                                    31/   10/   41        76%
-------------------------------------------------------------------------------------------------------------------------

    Module: vdfsm, File: vdfsm.v
    -------------------------------------------------------------------------------------------------------------------
    Missed Combinations  (* = missed value)

      ===============================================================================================================
```



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   FINITE STATE MACHINE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                         State                              Arc
Module/Task/Function      Filename               Hit/Miss/Total    Percent Hit    Hit/Miss/Total    Percent hit
-------------------------------------------------------------------------------------------------------------------------
  $root                   NA                       0/    0/    0      100%          0/    0/    0      100%
  vdfsm_tb                vdfsm_tb.v               0/    0/    0      100%          0/    0/    0      100%
  vdfsm                   vdfsm.v                  0/    0/    0      100%          0/    0/    0      100%
-------------------------------------------------------------------------------------------------------------------------
  Accumulated                                      0/    0/    0      100%          0/    0/    0      100%


samaksh19200@edaserver3:~/Ass3/Q3$ 
```