

CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Department of Electrical and Computer Engineering

ECE 526L

LAB – 4: Model of an edge triggered flip/flop using a hierarchal modelling.

INTRODUCTION:

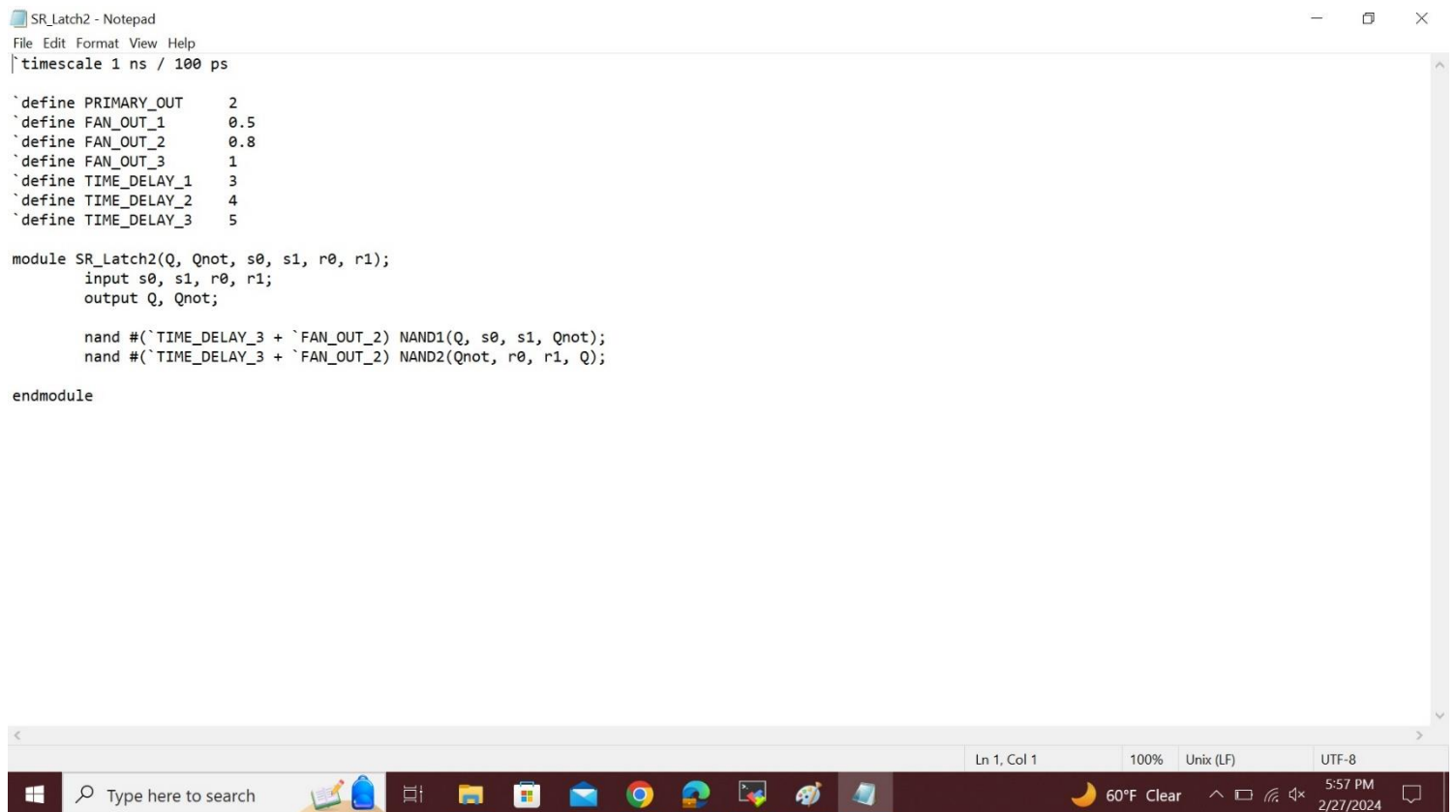
In this lab, we modelled an edge triggered flip-flop using a hierarchical modelling approach using **Synopsys VCS** in **Linux OS** environment and different terminal commands.

METHODOLOGY:

The first thing is being familiar with the Linux and Synopsys VCS, that will navigate your system using terminal. Now to write a Verilog code for the module using a text editor, always use Linux based text editors and the file should have “.v” extension.

In this lab, we will model an edge triggered flip-flop using a hierarchical modelling approach by using primitive gates in this question and we will write a code for **SR_latch**. And now save as “**SR_Latch2**”.

The module name and file name should be consistent,



```
SR_Latch2 - Notepad
File Edit Format View Help
timescale 1 ns / 100 ps

`define PRIMARY_OUT      2
`define FAN_OUT_1        0.5
`define FAN_OUT_2        0.8
`define FAN_OUT_3        1
`define TIME_DELAY_1     3
`define TIME_DELAY_2     4
`define TIME_DELAY_3     5

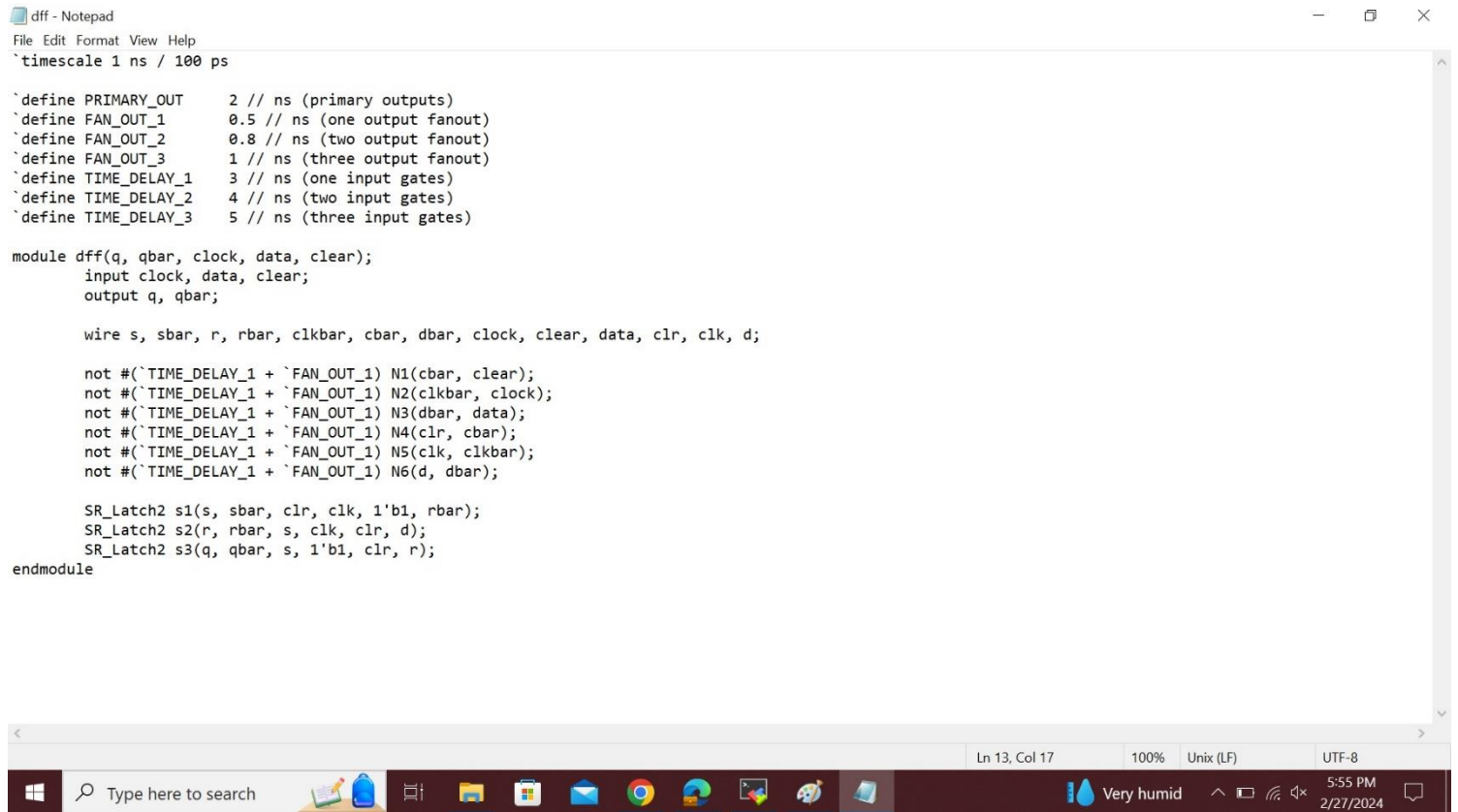
module SR_Latch2(Q, Qnot, s0, s1, r0, r1);
    input s0, s1, r0, r1;
    output Q, Qnot;

    nand #(`TIME_DELAY_3 + `FAN_OUT_2) NAND1(Q, s0, s1, Qnot);
    nand #(`TIME_DELAY_3 + `FAN_OUT_2) NAND2(Qnot, r0, r1, Q);

endmodule
```

This is the fig of “**SR_Latch2**”, store it in new file, also in folder “**Lab4**”.

Now for D-flip flop circuit, write the code using above module SR Latch for design of D- flip flop and save it as “dff.v”



```
dff - Notepad
File Edit Format View Help
`timescale 1 ns / 100 ps

`define PRIMARY_OUT      2 // ns (primary outputs)
`define FAN_OUT_1        0.5 // ns (one output fanout)
`define FAN_OUT_2        0.8 // ns (two output fanout)
`define FAN_OUT_3        1 // ns (three output fanout)
`define TIME_DELAY_1     3 // ns (one input gates)
`define TIME_DELAY_2     4 // ns (two input gates)
`define TIME_DELAY_3     5 // ns (three input gates)

module dff(q, qbar, clock, data, clear);
    input clock, data, clear;
    output q, qbar;

    wire s, sbar, r, rbar, clkbar, cbar, dbar, clock, clear, data, clr, clk, d;

    not #(`TIME_DELAY_1 + `FAN_OUT_1) N1(cbar, clear);
    not #(`TIME_DELAY_1 + `FAN_OUT_1) N2(clkbar, clock);
    not #(`TIME_DELAY_1 + `FAN_OUT_1) N3(dbar, data);
    not #(`TIME_DELAY_1 + `FAN_OUT_1) N4(clr, cbar);
    not #(`TIME_DELAY_1 + `FAN_OUT_1) N5(clk, clkbar);
    not #(`TIME_DELAY_1 + `FAN_OUT_1) N6(d, dbar);

    SR_Latch2 s1(s, sbar, clr, clk, 1'b1, rbar);
    SR_Latch2 s2(r, rbar, s, clk, clr, d);
    SR_Latch2 s3(q, qbar, s, 1'b1, clr, r);
endmodule

Ln 13, Col 17    100%    Unix (LF)    UTF-8
```

The above figure is the “dff.v” deisgn code.

Then we should write a test bench module to verify the functionality of your flip-flop, in this module we use each of the output system tasks, \$monitor, \$display, \$write and \$strobe and name it as “dff_tb.v”.

This below is “dff_tb.v” test bench code.

```
dff_tb - Notepad
File Edit Format View Help
`timescale 1ns / 1ns
module dff_tb();

    reg clock;
    reg clear;
    reg data;
    wire q, qbar;

    dff UUT(q, qbar, clock, data, clear);

    initial begin
        $vcdpluson;
        clock = 0;
        clear = 0;
        data = 0;

        forever #20 clock = ~clock;
    end

    // Testbench Behavior

    // Simulate test scenario
    initial begin
        #20 clear = 0;

        // Apply data to D input
        #50 data = 1;
        #50 data = 0;
        #50 data = 1;

        #50 clear = 1;

        #50 data = 1;
        #50 data = 0;
        #50 data = 1;
    end
endmodule
```

Ln 27, Col 16 100% Unix (LF) UTF-8 5:56 PM 2/27/2024 Very humid

```
dff_tb - Notepad
File Edit Format View Help
    // Apply data to D input
    #50 data = 1;
    #50 data = 0;
    #50 data = 1;

    #50 clear = 1;

    #50 data = 1;
    #50 data = 0;
    #50 data = 1;

    //Finish simulation after some time
    #100 $finish;

end

always @(posedge clock or posedge clear) begin
    if (clear) begin

        //Use $monitor to display clear information
        $monitor("Clear asserted at time %t", $time);
    end else begin

        //Use $display to display D flip-flop input and output values
        $display("Input d = %b, output q = %b, output qbar = %b at time %t", data, q, qbar, $time);

        //Use $write to write input and output values to a file
        $write("d=%b q=%b qbar=%b\n", data, q, qbar);

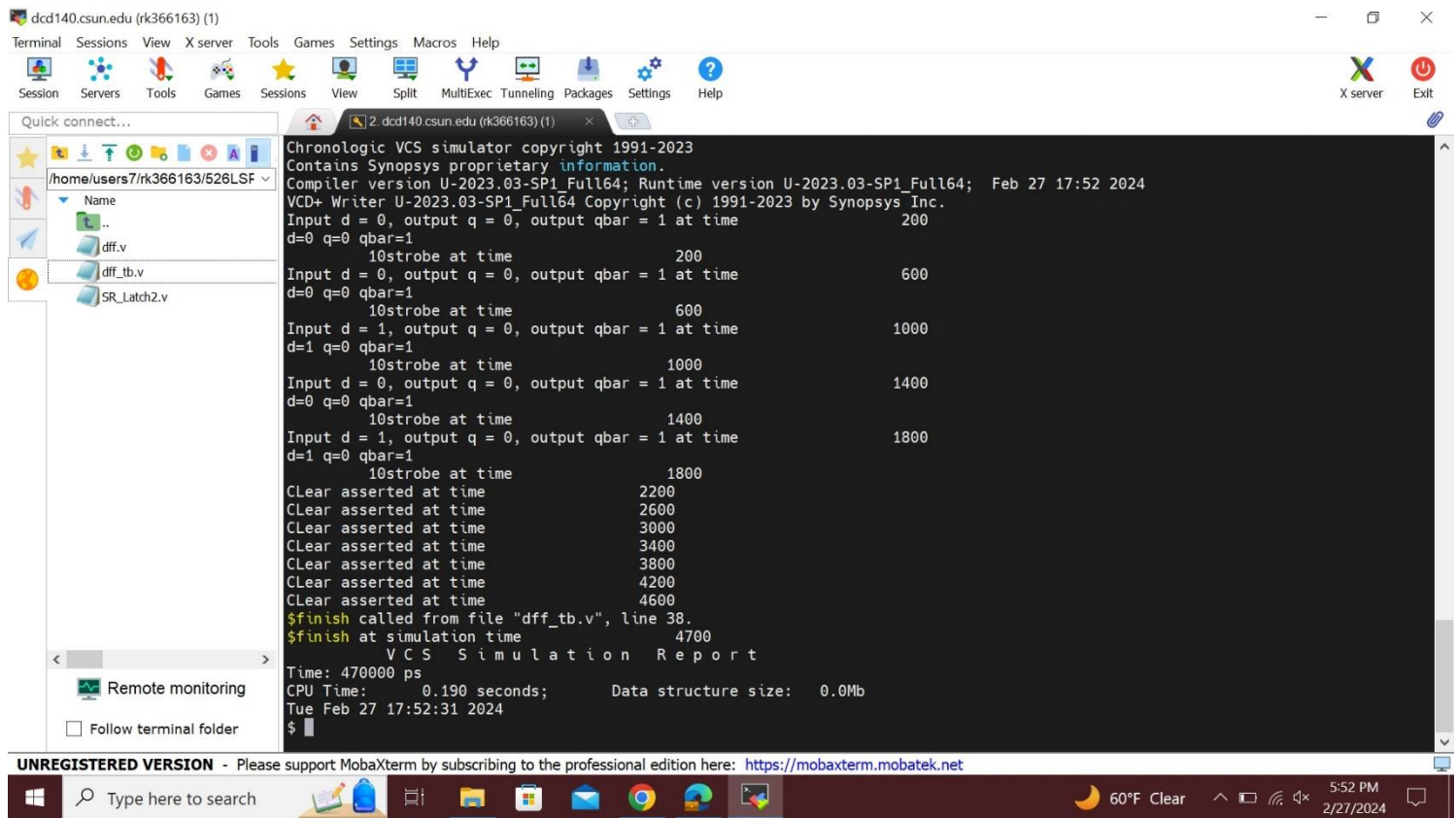
        //Use $strobe to create a strobe signal for debugging
        $strobe(10, "strobe at time %t", $time);
    end
end

endmodule
```

Ln 59, Col 1 100% Unix (LF) UTF-8 60°F Clear 5:56 PM 2/27/2024

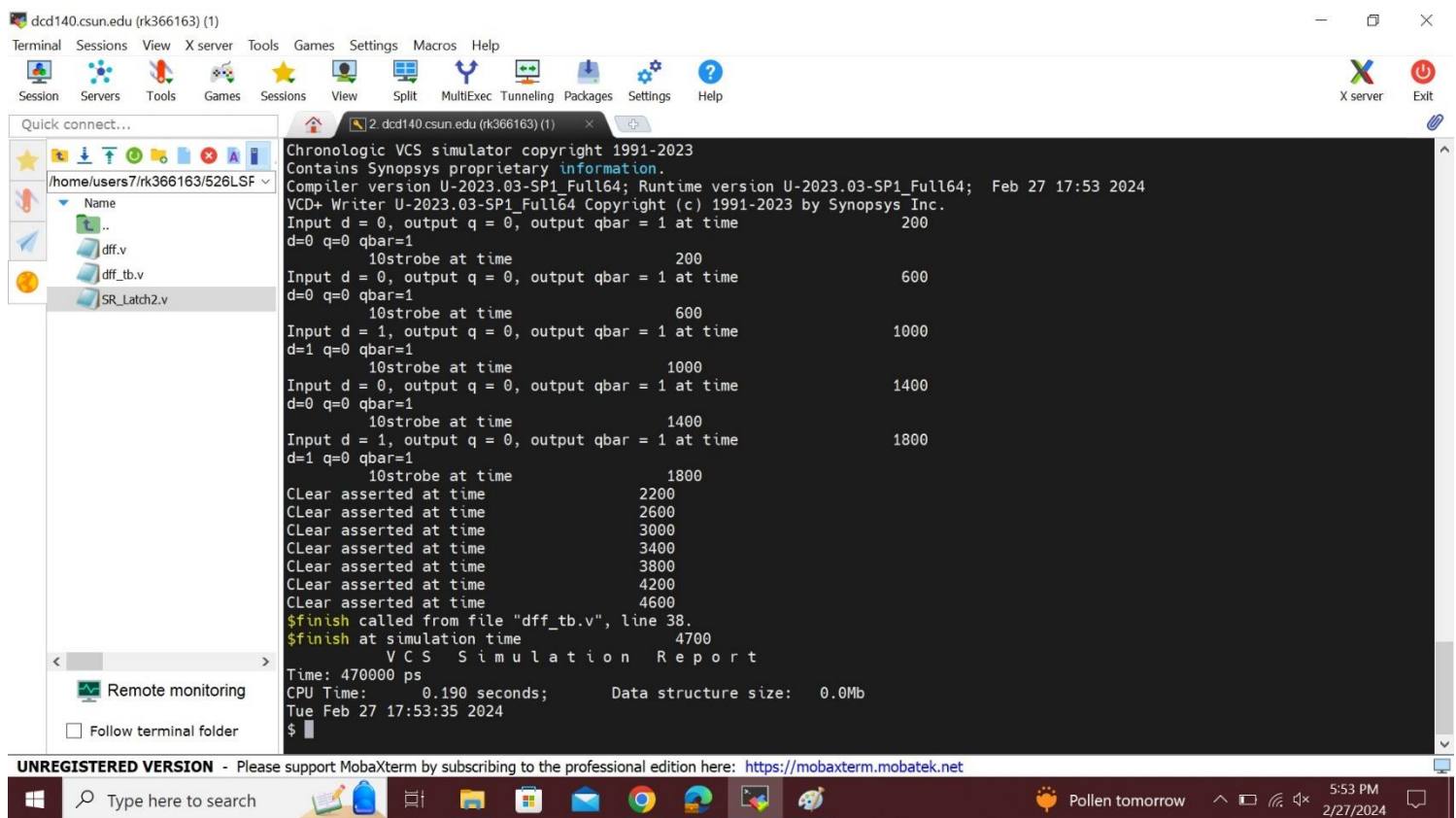
Now, after saving the SR Latch and D-flip/flop. Then, use the command as follows, “**vcs -debug_access+all SRLatch.v dff.v dff_tb.v**” If we don’t see any error messages if everything is typed correctly then output should look like below figure;

Simv:



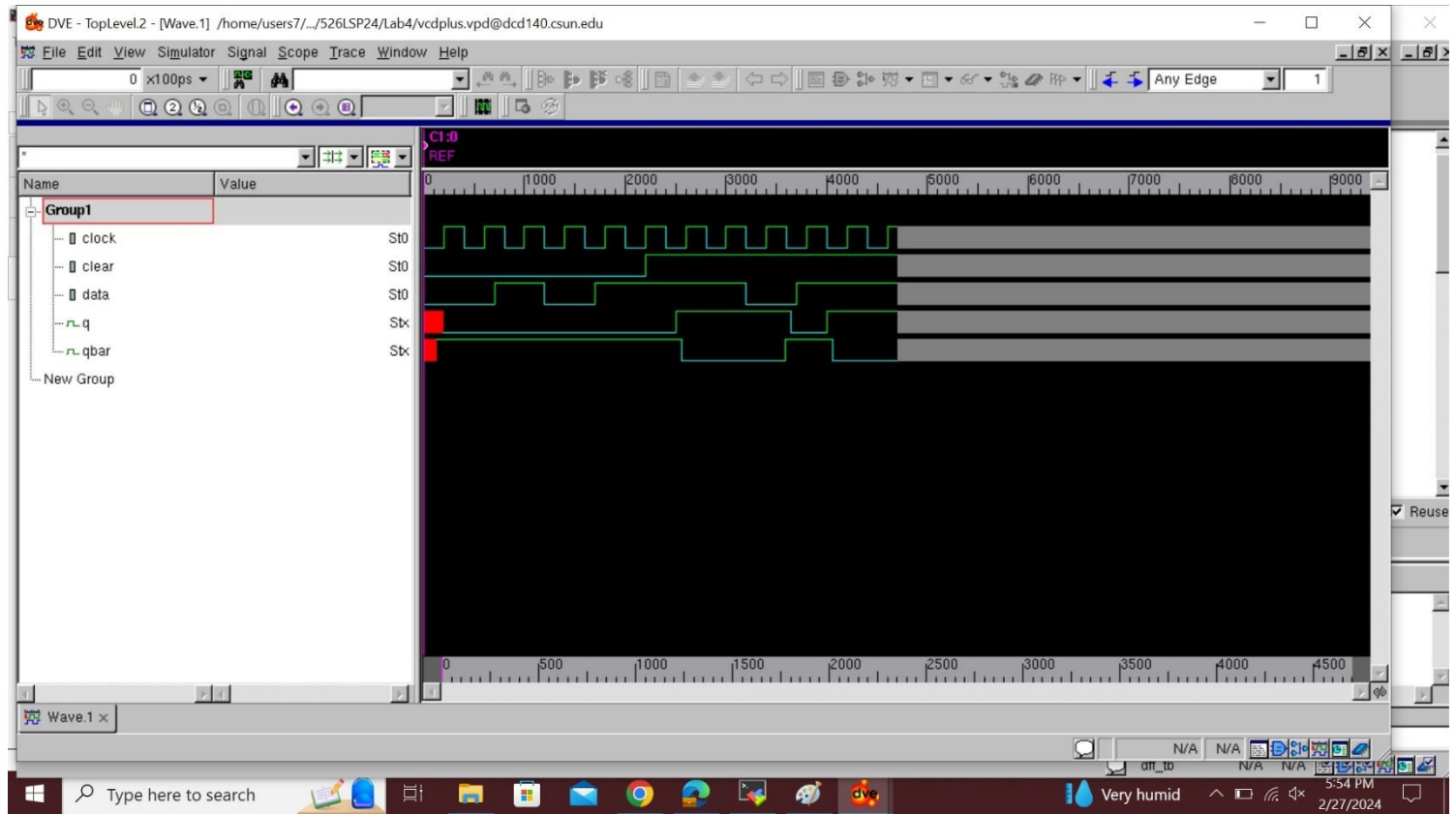
```
Chronologic VCS simulator copyright 1991-2023
Contains Synopsys proprietary information.
Compiler version U-2023.03-SP1_Full64; Runtime version U-2023.03-SP1_Full64; Feb 27 17:52 2024
VCD+ Writer U-2023.03-SP1_Full64 Copyright (c) 1991-2023 by Synopsys Inc.
Input d = 0, output q = 0, output qbar = 1 at time 200
d=0 q=0 qbar=1
10strobe at time 200
Input d = 0, output q = 0, output qbar = 1 at time 600
d=0 q=0 qbar=1
10strobe at time 600
Input d = 1, output q = 0, output qbar = 1 at time 1000
d=1 q=0 qbar=1
10strobe at time 1000
Input d = 0, output q = 0, output qbar = 1 at time 1400
d=0 q=0 qbar=1
10strobe at time 1400
Input d = 1, output q = 0, output qbar = 1 at time 1800
d=1 q=0 qbar=1
10strobe at time 1800
Cclear asserted at time 2200
Cclear asserted at time 2600
Cclear asserted at time 3000
Cclear asserted at time 3400
Cclear asserted at time 3800
Cclear asserted at time 4200
Cclear asserted at time 4600
$finish called from file "dff_tb.v", line 38.
$finish at simulation time 4700
VCS Simulation Report
Time: 470000 ps
CPU Time: 0.190 seconds; Data structure size: 0.0Mb
Tue Feb 27 17:52:31 2024
$
```

Log File Lab4: Command: “simv -l Lab4.log” .



```
Chronologic VCS simulator copyright 1991-2023
Contains Synopsys proprietary information.
Compiler version U-2023.03-SP1_Full64; Runtime version U-2023.03-SP1_Full64; Feb 27 17:53 2024
VCD+ Writer U-2023.03-SP1_Full64 Copyright (c) 1991-2023 by Synopsys Inc.
Input d = 0, output q = 0, output qbar = 1 at time 200
d=0 q=0 qbar=1
10strobe at time 200
Input d = 0, output q = 0, output qbar = 1 at time 600
d=0 q=0 qbar=1
10strobe at time 600
Input d = 1, output q = 0, output qbar = 1 at time 1000
d=1 q=0 qbar=1
10strobe at time 1000
Input d = 0, output q = 0, output qbar = 1 at time 1400
d=0 q=0 qbar=1
10strobe at time 1400
Input d = 1, output q = 0, output qbar = 1 at time 1800
d=1 q=0 qbar=1
10strobe at time 1800
Cclear asserted at time 2200
Cclear asserted at time 2600
Cclear asserted at time 3000
Cclear asserted at time 3400
Cclear asserted at time 3800
Cclear asserted at time 4200
Cclear asserted at time 4600
$finish called from file "dff_tb.v", line 38.
$finish at simulation time 4700
VCS Simulation Report
Time: 470000 ps
CPU Time: 0.190 seconds; Data structure size: 0.0Mb
Tue Feb 27 17:53:35 2024
$
```

To see the wave form first type “**dve-full64**” in command line, then we will get the blank simulation window, then go to the file and open database then choose **vcdplus.vpd** and open the file. Then finally select all the signals with the mouse, then right click and select to “**add to wave ->new wave view**”.



Analysis of result:

The D-Flip/Flop's inputs and outputs are identical, that means the result will be 0 if the input is 0 => 0. Additionally, if the input is 1 => 1, then the output will also be 1. Here, "d" stands for the data input provided by the table mentioned earlier details the D - input as 0, 0, 1, 1 and the 'qbar' output as being identical to the data supplied as 0, 0, 1, 1.

Furthermore, the data clock will alternate with the letter "q", which will be 0, 1, 0, 1. The D-Flip/Flop completely express what is said in the above waveform, to provide alternative values for the clock, clear, and data, we also use a "NOT" gate here as inverter, which are substitute values for them. Here, the “q” and “qbar” outputs are obtained using the three SR-latches, simply adding the values specified in the code given by the values from the provided circuit. Finally, positive edge triggered d flip/flop becomes active when its clock signal goes from low to high and ignores the high-low transition.

Conclusion:

In this lab we done model of an edge triggered flip/flop using a hierarchal modelling approach using Synopsys VCS in Linux OS environment and different terminal commands

I hereby attest that this lab report is entirely my own work. I have not copied either code or text from anyone, neither have I allowed nor I will let anyone to copy my work.

Name(printed) Raj Kumar

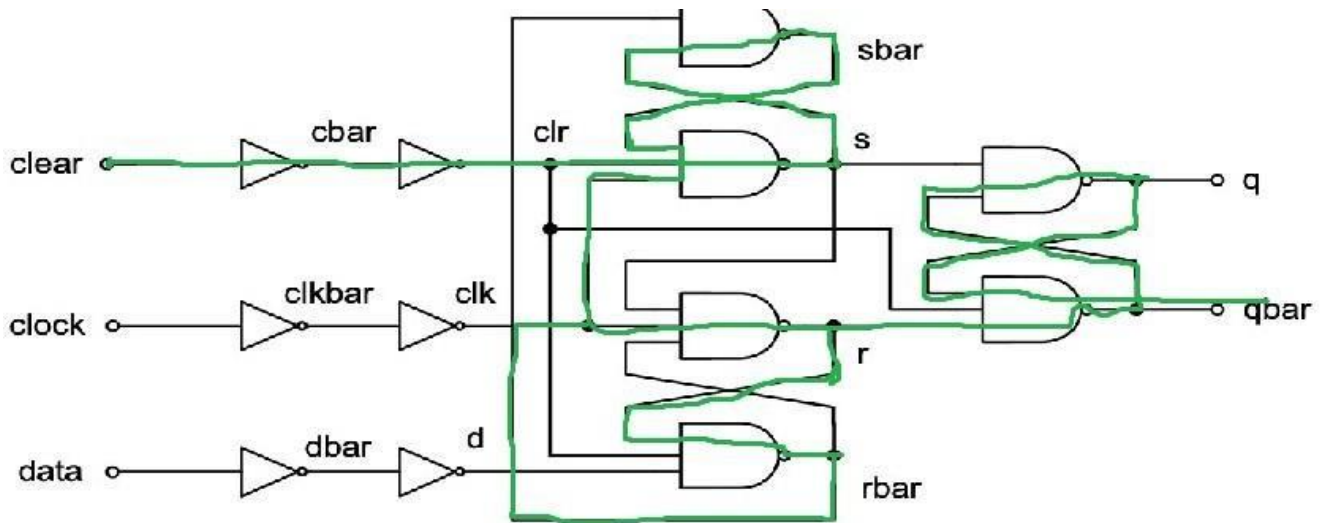
Name(signed) Raj Kumar

Date 03/01/2024

Lab Report Solutions:

1. Calculate the longest path delay, show how you calculated it (which gates and what delays) and show the value?

Solution:



Positive Edge-Triggered D Flip-Flop with Clear
Figure 2

The longest path is calculated as

=> NOT + NOT + NAND + NAND + NAND + NAND + NAND + NAND

=> $(3+0.5) + (3+0.5) + (5+0.8) + (5+0.8) + (5+0.8) + (5+0.8) + (5+1.0) + (5+2.0)$

=> 43.2 ns

2. What is the maximum operating frequency for your circuit?

Solution:

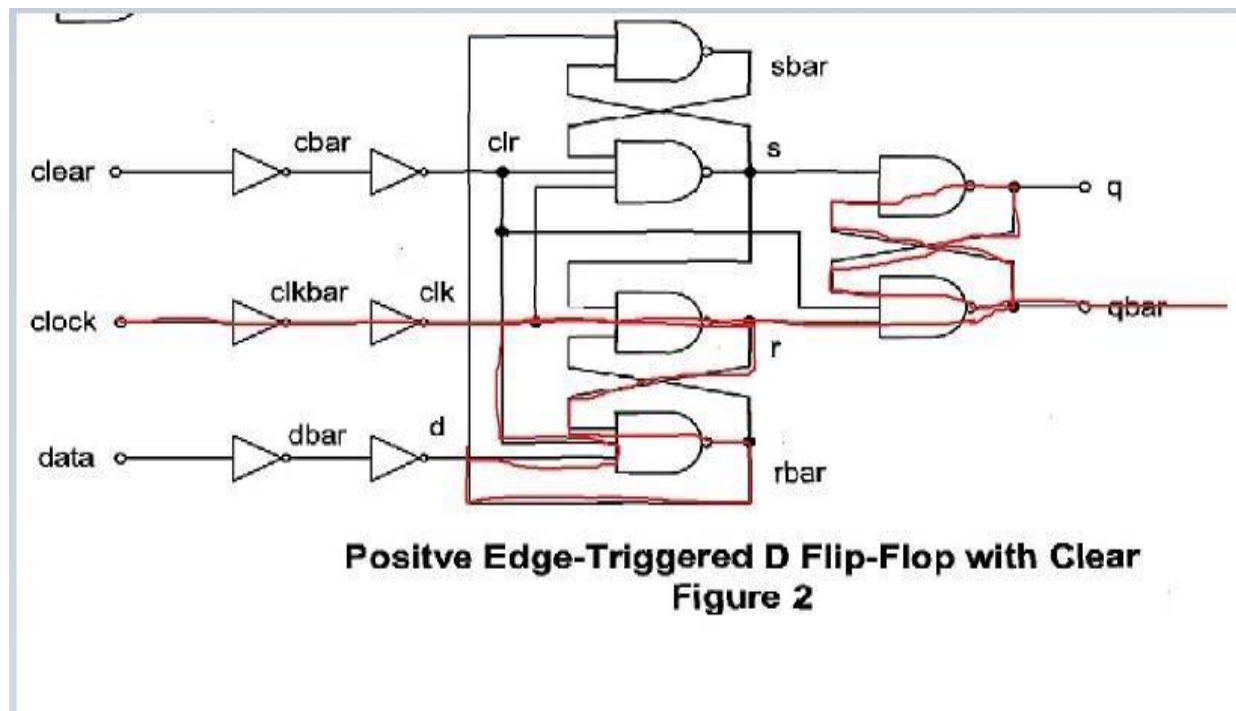
It is calculated by $1/\text{shortest delay}$

Clock to qbar :

=> NOT + NOT + NAND + NAND + NAND + NAND

=> $(3+0.5) + (3+0.5) + (5+0.8) + (5+0.8) + (5+1.0) + (5+2.0) = 31.6\text{ns}$

=> $1/31.6 = 3.174603175 \times 10^7$



- Therefore, the above picture is the shortest delay path.