VLSI Assignment-3

Aravind Narayanan

2019102014

Question 1

Objective:

Minimize average delay of the circuit while using pass transistor logic by adjusting it's widths.

Circuit:

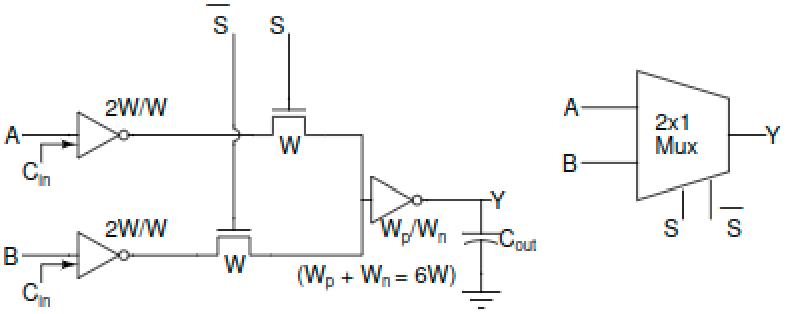


Figure 1

Constraints and Values given:

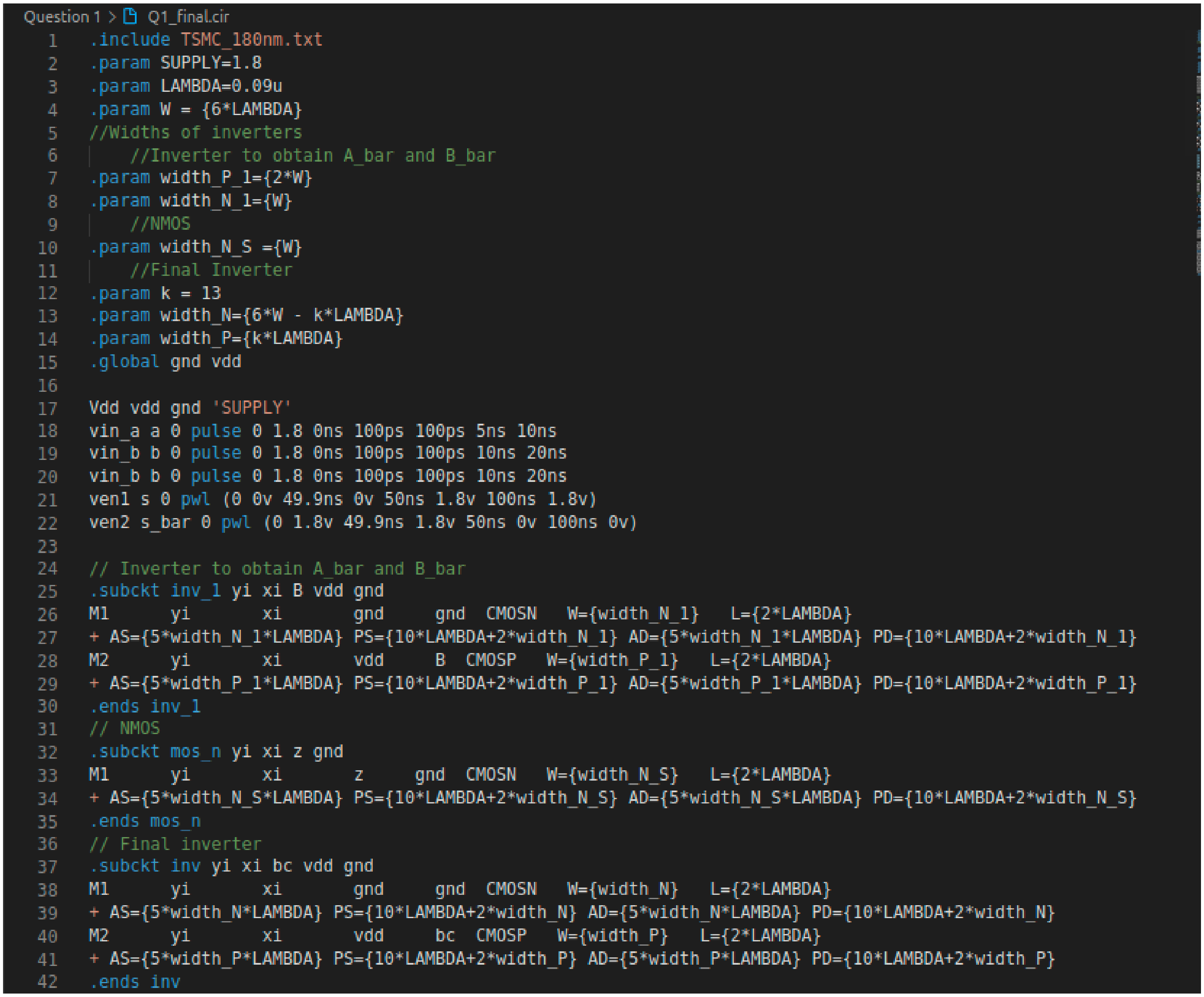
1) Cout/Cin = 2 for each path(input to output)
2) Length of transistor = 0.18um
3) Width constraint: Wp + Wn = 6W where Wp = PMOS Width, Wn = NMOS Width of final inverter while W is another parameter with which all widths are expressed.

Method:

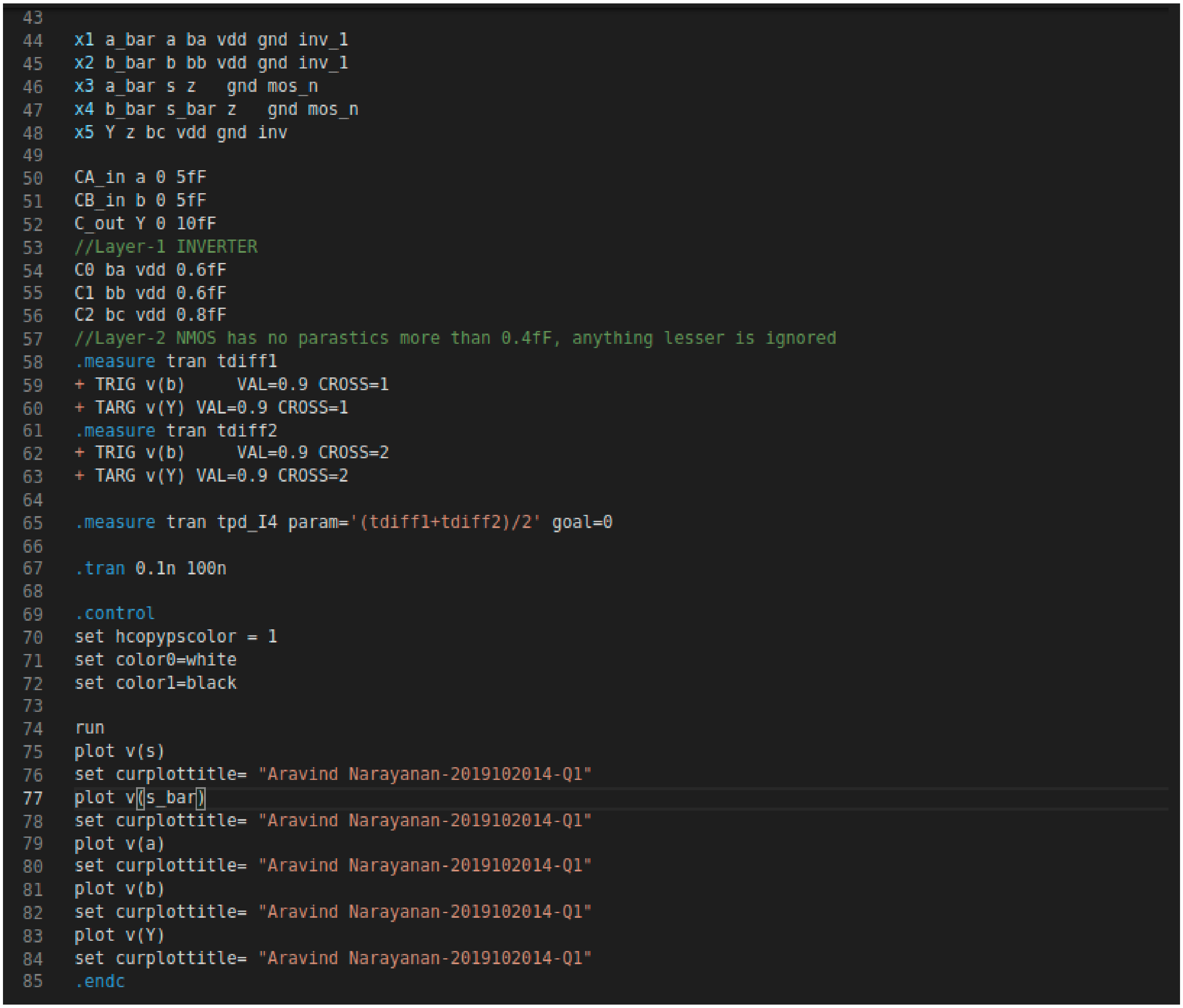
1) As Wn + Wp = 6W, express Wn and Wp in terms of lambda where **Wn = 6W - k(lambda) and Wp = k*(lambda)**.
2) Then we first identify a W value for which we obtain minimum delay for arbitrary k values so that we can observe the minimum delay for a particular W.
3) Then we use that W value and obtain parasitics for A, B and NMOS S and S_bar using their respective magic layout
4) After obtaining the parasticis for this constraint, we vary k to obtain the minimum value of delay by optimising on the final inverter.
5) Once we find that, we add parasticis with magic and confirm that it is optimized by changing k value to its nearby range.

NETLISTS:

A)



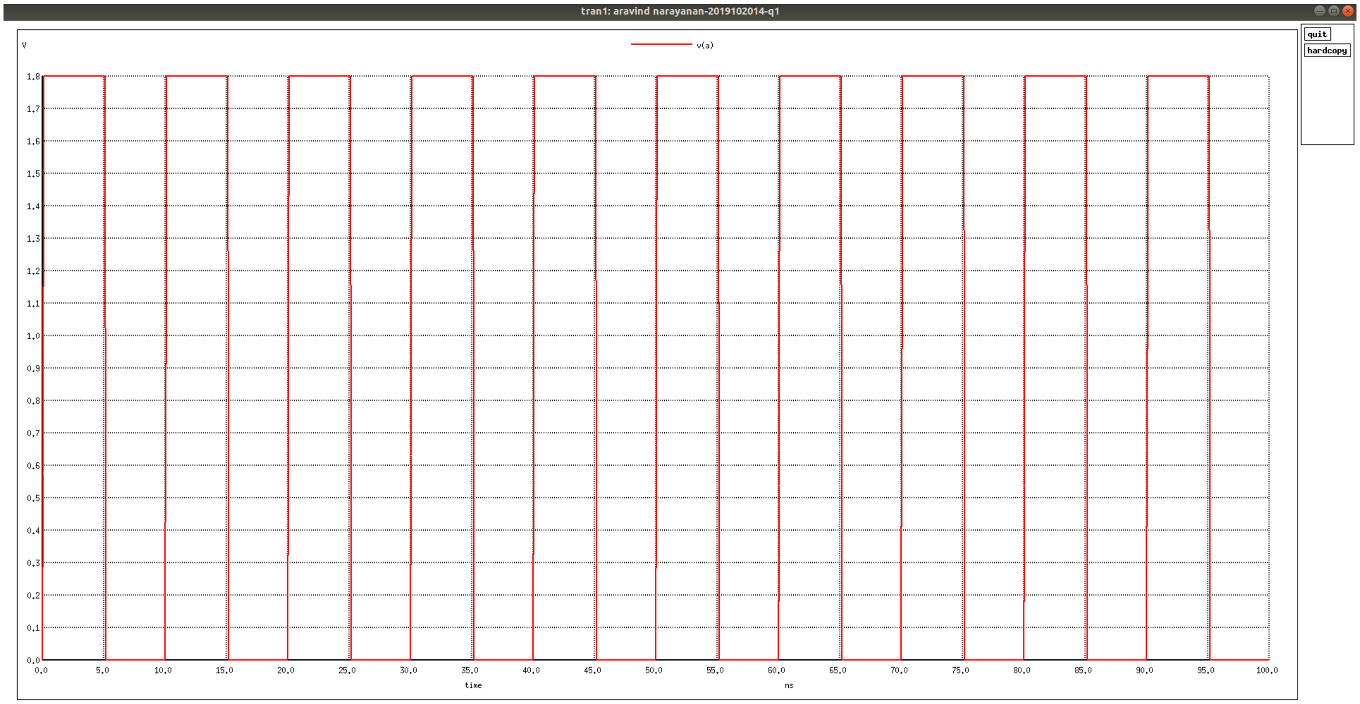
```
.include TSMC_180nm.txt
.param SUPPLY=1.8
.param LAMBDA=0.09u
.param W = {6*LAMBDA}
//Widths of inverters
    //Inverter to obtain A_bar and B_bar
.param width_P_1={2*W}
.param width_N_1={W}
    //NMOS
.param width_N_S ={W}
    //Final Inverter
.param k = 13
.param width_N={6*W - k*LAMBDA}
.param width_P={k*LAMBDA}
.global gnd vdd

Vdd vdd gnd 'SUPPLY'
vin_a a 0 pulse 0 1.8 0ns 100ps 100ps 5ns 10ns
vin_b b 0 pulse 0 1.8 0ns 100ps 100ps 10ns 20ns
vin_b b 0 pulse 0 1.8 0ns 100ps 100ps 10ns 20ns
ven1 s 0 pwl (0 0v 49.9ns 0v 50ns 1.8v 100ns 1.8v)
ven2 s_bar 0 pwl (0 1.8v 49.9ns 1.8v 50ns 0v 100ns 0v)

// Inverter to obtain A_bar and B_bar
.subckt inv_1 yi xi B vdd gnd
M1      yi      xi      gnd     gnd  CMOSN   W={width_N_1}   L={2*LAMBDA}
+ AS={5*width_N_1*LAMBDA} PS={10*LAMBDA+2*width_N_1} AD={5*width_N_1*LAMBDA} PD={10*LAMBDA+2*width_N_1}
M2      yi      xi      vdd     B  CMOSP   W={width_P_1}   L={2*LAMBDA}
+ AS={5*width_P_1*LAMBDA} PS={10*LAMBDA+2*width_P_1} AD={5*width_P_1*LAMBDA} PD={10*LAMBDA+2*width_P_1}
.ends inv_1
// NMOS
.subckt mos_n yi xi z gnd
M1      yi      xi      z     gnd  CMOSN   W={width_N_S}   L={2*LAMBDA}
+ AS={5*width_N_S*LAMBDA} PS={10*LAMBDA+2*width_N_S} AD={5*width_N_S*LAMBDA} PD={10*LAMBDA+2*width_N_S}
.ends mos_n
// Final inverter
.subckt inv yi xi bc vdd gnd
M1      yi      xi      gnd     gnd  CMOSN   W={width_N}   L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}
M2      yi      xi      vdd     bc  CMOSP   W={width_P}   L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA} PD={10*LAMBDA+2*width_P}
.ends inv
```



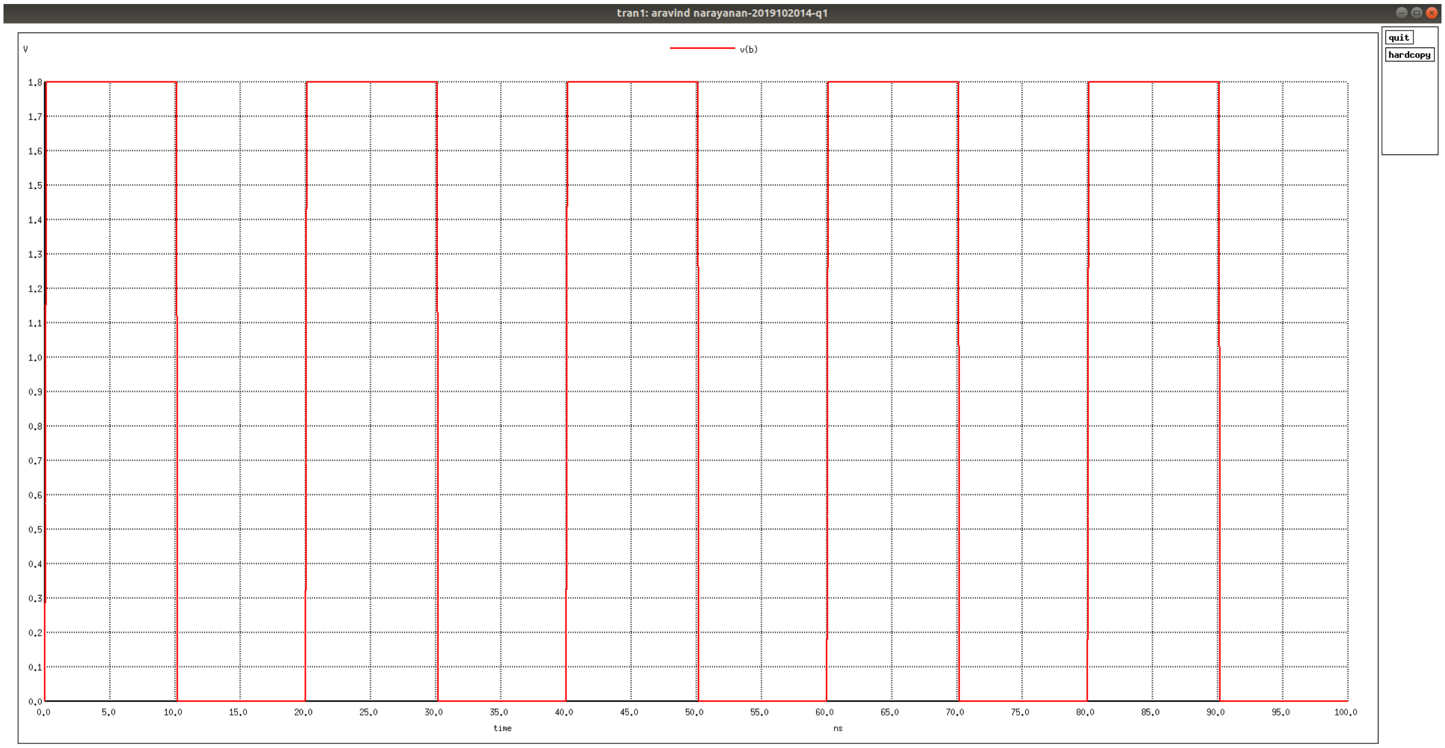
```

x1 a_bar a ba vdd gnd inv_1
x2 b_bar b bb vdd gnd inv_1
x3 a_bar s z   gnd mos_n
x4 b_bar s_bar z   gnd mos_n
x5 Y z bc vdd gnd inv

CA_in a 0 5fF
CB_in b 0 5fF
C_out Y 0 10fF
//Layer-1 INVERTER
C0 ba vdd 0.6fF
C1 bb vdd 0.6fF
C2 bc vdd 0.8fF
//Layer-2 NMOS has no parastics more than 0.4fF, anything lesser is ignored
.measure tran tdiff1
+ TRIG v(b)     VAL=0.9 CROSS=1
+ TARG v(Y) VAL=0.9 CROSS=1
.measure tran tdiff2
+ TRIG v(b)     VAL=0.9 CROSS=2
+ TARG v(Y) VAL=0.9 CROSS=2

.measure tran tpd_I4 param='(tdiff1+tdiff2)/2' goal=0

.tran 0.1n 100n

.control
set hcopypscolor = 1
set color0=white
set color1=black

run
plot v(s)
set curplottitle= "Aravind Narayanan-2019102014-Q1"
plot v(s_bar)
set curplottitle= "Aravind Narayanan-2019102014-Q1"
plot v(a)
set curplottitle= "Aravind Narayanan-2019102014-Q1"
plot v(b)
set curplottitle= "Aravind Narayanan-2019102014-Q1"
plot v(Y)
set curplottitle= "Aravind Narayanan-2019102014-Q1"
.endc
```

PLOTS:

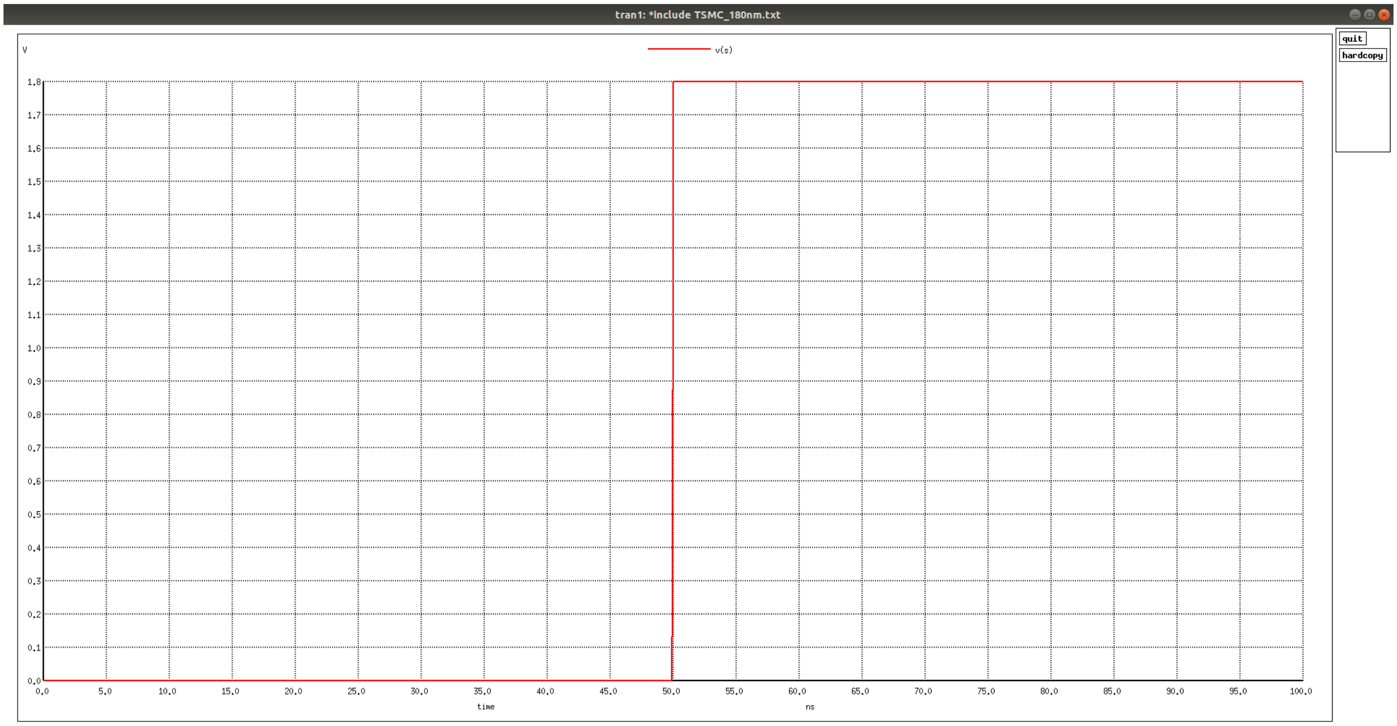
1) V(A)



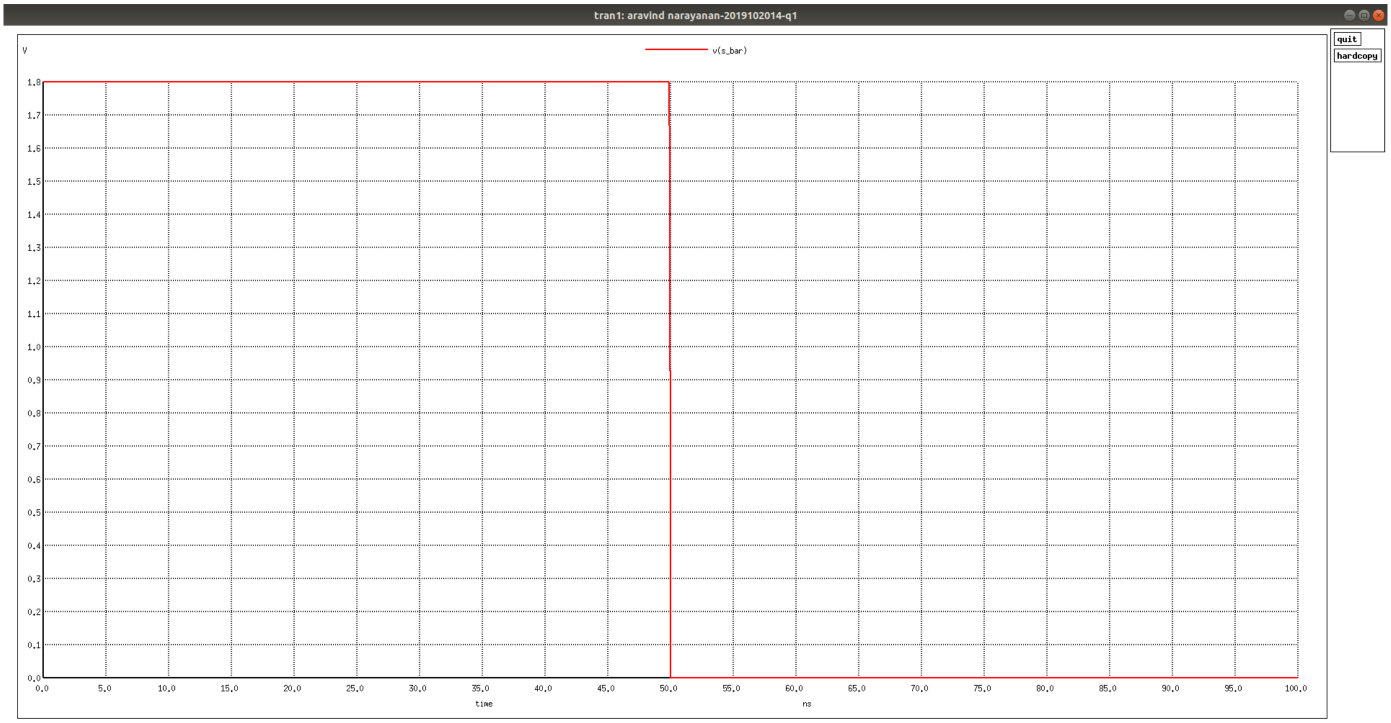
2) V(B)



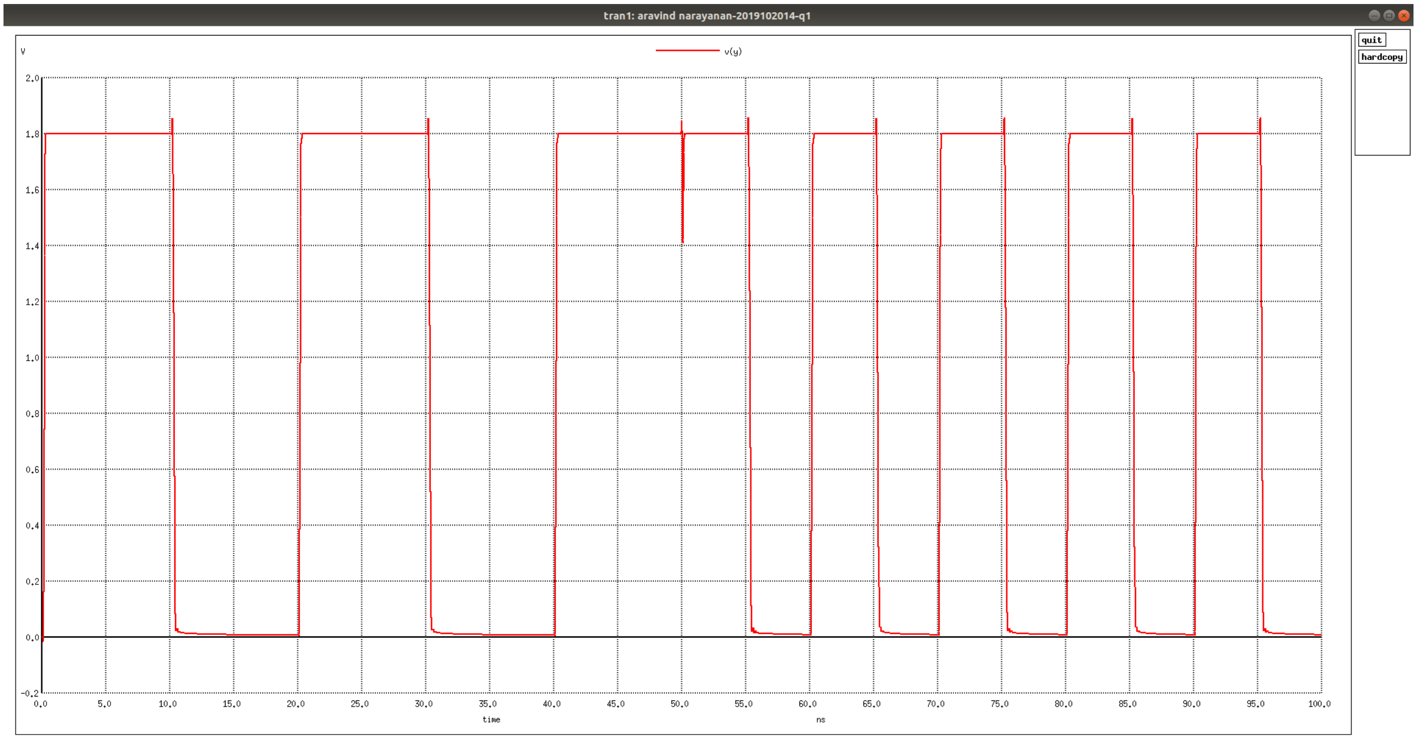
3) V(S)


tran1: *include TSMC_180nm.txt

4) V(S_Bar)

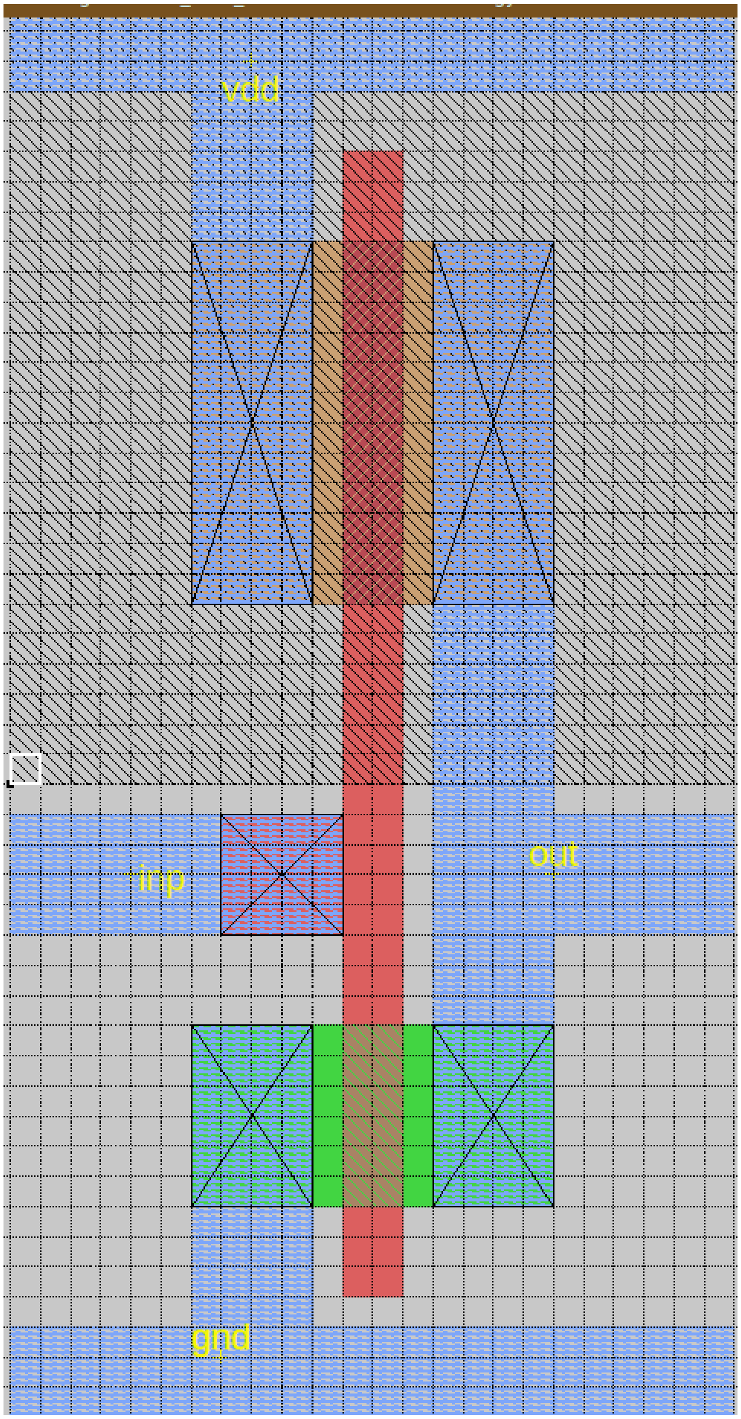

tran1: aravind narayanan-2019102014-q1

5) V(Y)

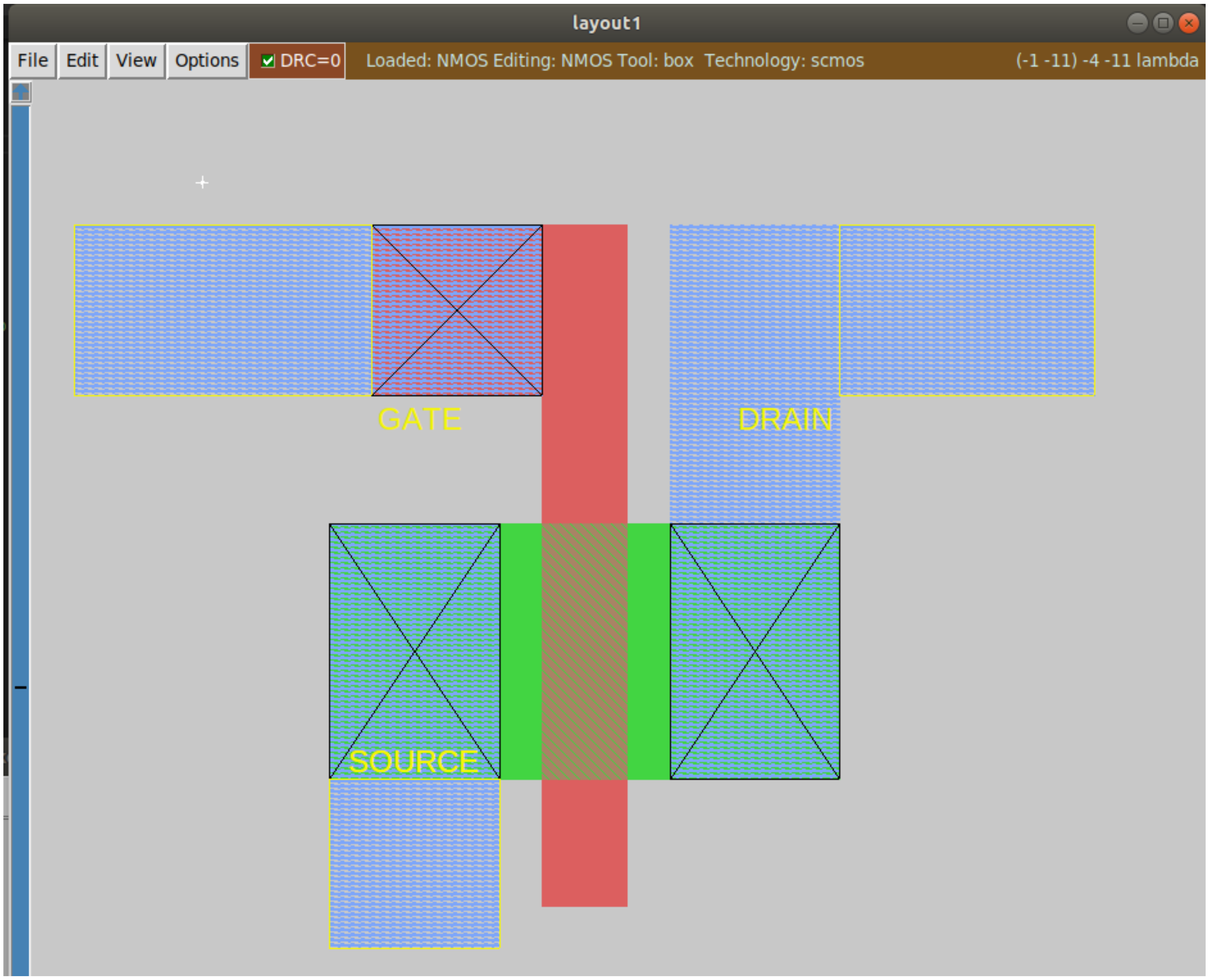

tran1: aravind narayanan-2019102014-q1
quit
hardcopy
V
v(y)
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0.0
5.0
10.0
15.0
20.0
25.0
30.0
35.0
40.0
45.0
50.0
55.0
60.0
65.0
70.0
75.0
80.0
85.0
90.0
95.0
100.0
time
ns

MAGIC Layouts to Identify Parasitic Capacitances:

1) Inverters used to obtain A’ and B’ from A and B respectively:



2) NMOS used in Pass-Transistor Logic



Input-Output Table:

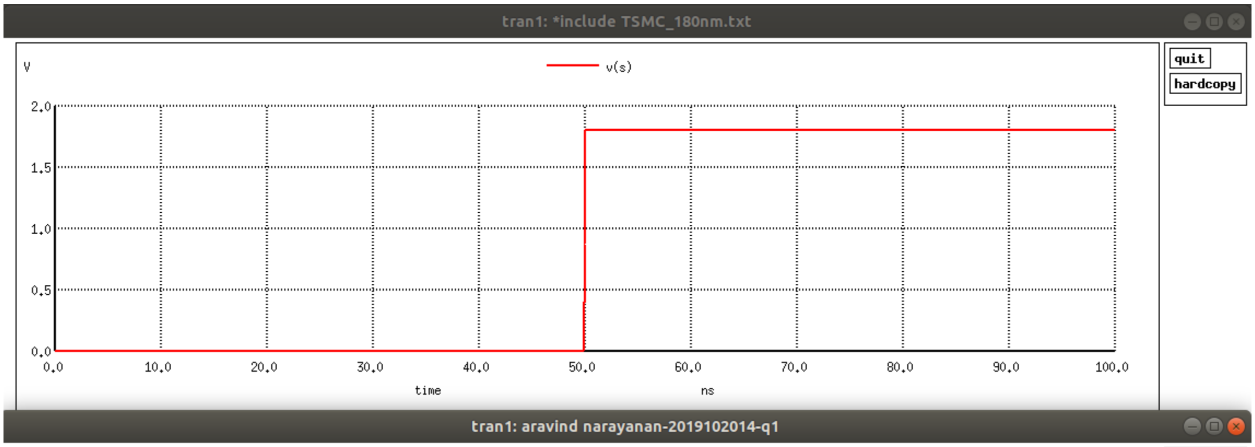
NOTE: Here 0 = LOW and 1 = HIGH

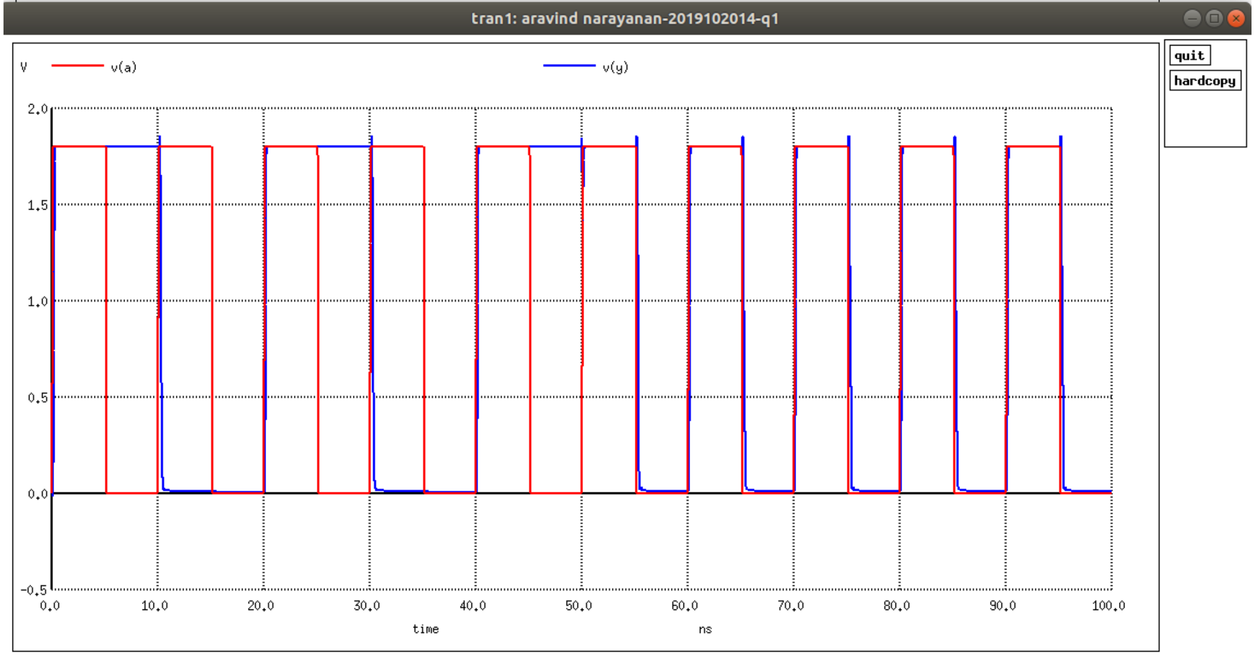
Time	A	B	S	S_bar	Y
0-5	1	1	0	1	1
5-10	0	1	0	1	1
10-15	1	0	0	1	0
15-20	0	0	0	1	0
20-25	1	1	0	1	1
25-30	0	1	0	1	1
35-35	1	0	0	1	0
35-40	0	0	0	1	0
40-45	1	1	0	1	1
45-50	0	1	0	1	1
50-55	1	0	1	0	1
55-60	0	0	1	0	0
60-65	1	1	1	0	1
65-70	0	1	1	0	0
70-75	1	0	1	0	1
75-80	0	0	1	0	0
80-85	1	1	1	0	1
85-90	0	1	1	0	0
90-95	1	0	1	0	1
95-100	0	0	1	0	0

We see that the MUX operation works correctly as when

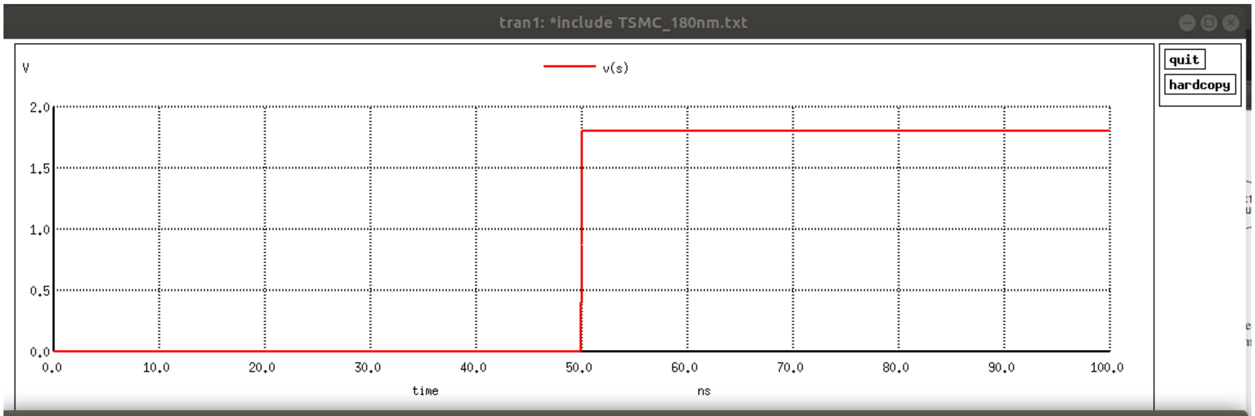
- **S = LOW, Y = B**
- **S = HIGH, Y = A**

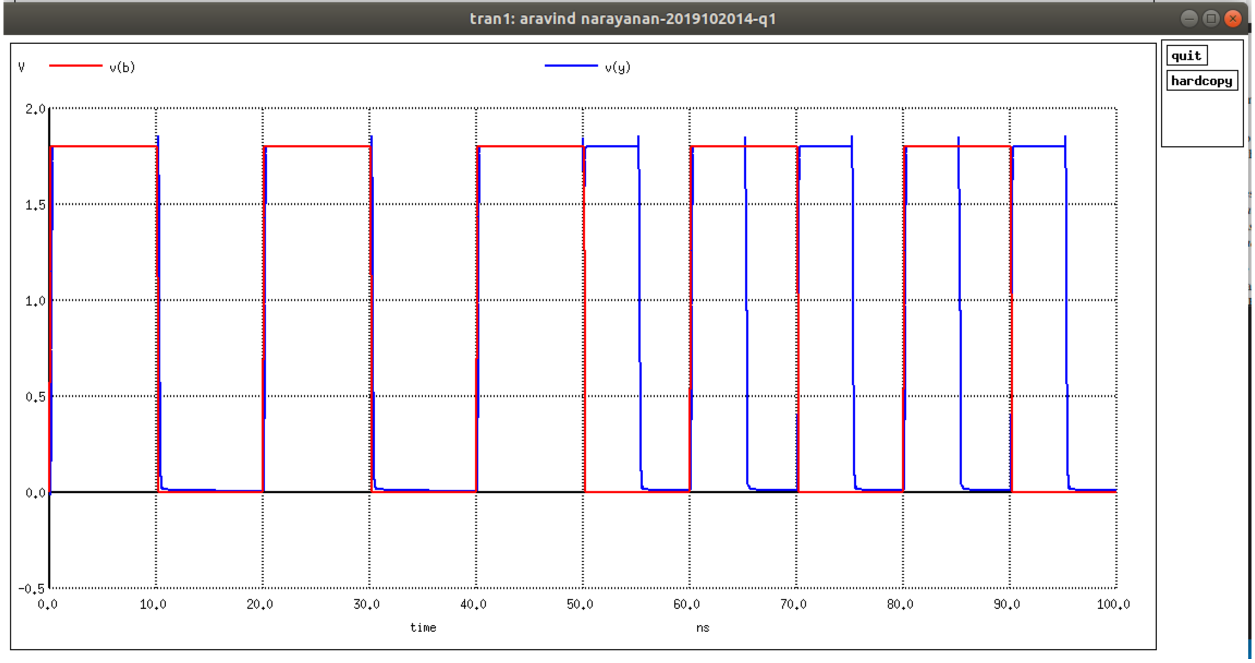
To show that this relationship is true, we will plot Y,A and Y,B together respectively to show the overlapping with respect to S.





In the above figure, as S = High from 50-100ns, the v(y) overlaps with v(a) perfectly.





In the above figure, as S = Low from 0-50ns, the v(y) overlaps with v(b) perfectly.

From the above two figures, we can prove that th

Observation Table:

k	Delay
2	0.232913ns
4	0.180142ns
9	0.151583ns
12	0.148952ns
13	0.148023ns
14	0.148661ns
20	0.158045ns
25	0.184007ns
30	0.273390ns
32	0.396401ns

We notice that the minimum delay for the circuit is obtained when the **W** = **6*Lambda, and Wp** = **13*Lambda, Wn** = **23*Lambda** we get the least possible delay. This Wp, Wn value also satisfies the constraints of Wp + Wn = 6W.

The output/input capacitances

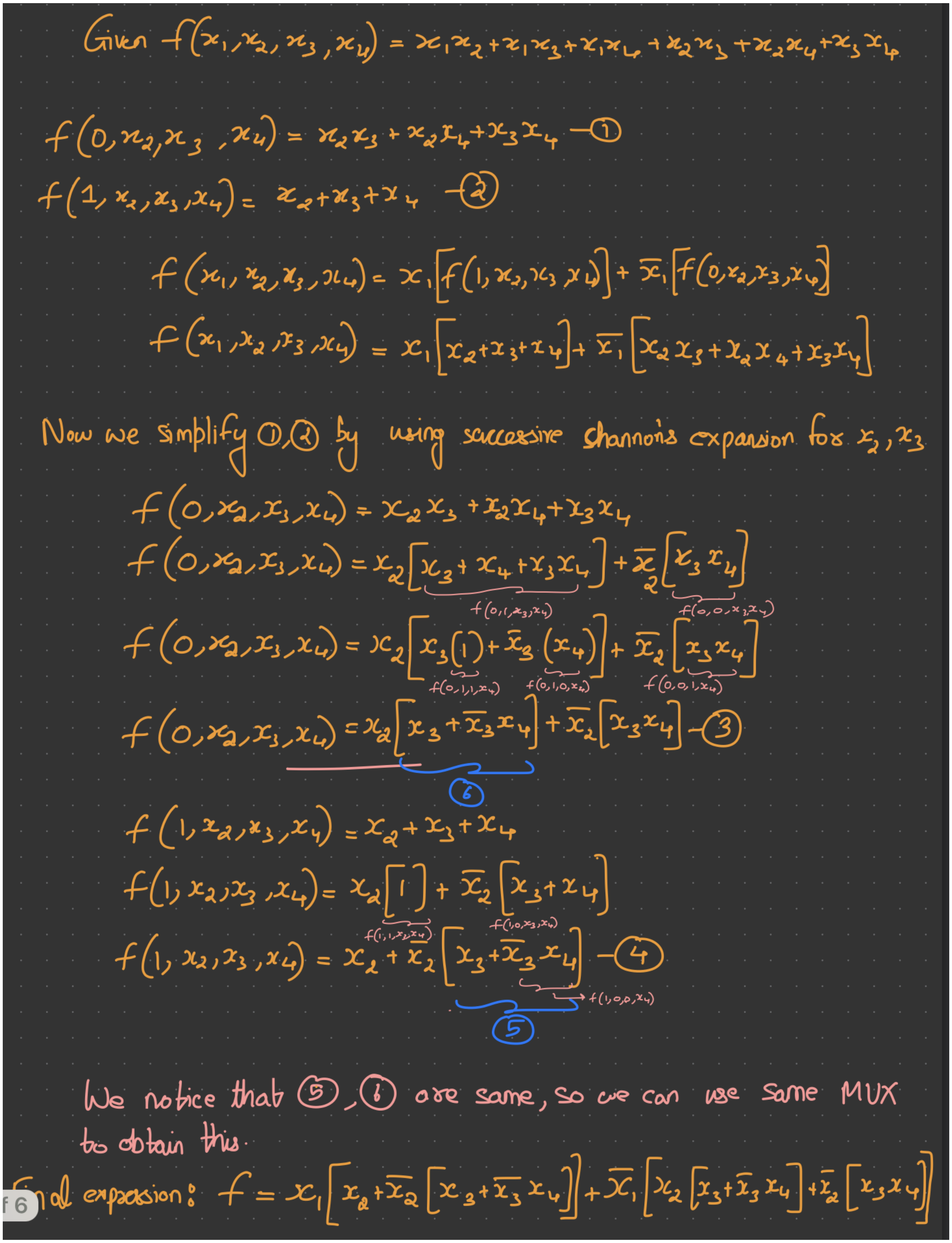
NOTE: While doing the MAGIC Layout parasitic capacitances(>0.4fF) are not present in the NMOS Layout, and in the CMOS layout it is only present between the body and the Vdd of PMOS.

Parasites obtained in subcircuits:

1) **Inverters at A,B**: 0.6fF between body and vdd of PMOS
2) **Inverter at output**: 0.8fF between body and vdd of PMOS

Question 2

A)



Given $f(x_1,x_2,x_3,x_4) = x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4$

$f(0,x_2,x_3,x_4) = x_2x_3+x_2x_4+x_3x_4$ — ①

$f(1,x_2,x_3,x_4) = x_2+x_3+x_4$ — ②

$$f(x_1,x_2,x_3,x_4) = x_1\left[f(1,x_2,x_3,x_4)\right] + \bar{x}_1\left[f(0,x_2,x_3,x_4)\right]$$

$$f(x_1,x_2,x_3,x_4) = x_1\left[x_2+x_3+x_4\right] + \bar{x}_1\left[x_2x_3+x_2x_4+x_3x_4\right]$$

Now we simplify ①,② by using successive shannon's expansion for x_2, x_3

$$f(0,x_2,x_3,x_4) = x_2x_3+x_2x_4+x_3x_4$$

$$f(0,x_2,x_3,x_4) = x_2\Big[\underbrace{x_3+x_4+x_3x_4}_{f(0,1,x_3,x_4)}\Big] + \bar{x}_2\Big[\underbrace{x_3x_4}_{f(0,0,x_3,x_4)}\Big]$$

$$f(0,x_2,x_3,x_4) = x_2\Big[x_3\underbrace{(1)}_{f(0,1,1,x_4)}+\bar{x}_3\underbrace{(x_4)}_{f(0,1,0,x_4)}\Big] + \bar{x}_2\Big[\underbrace{x_3x_4}_{f(0,0,1,x_4)}\Big]$$

$$f(0,x_2,x_3,x_4) = x_2\Big[\underbrace{x_3+\bar{x}_3x_4}_{⑥}\Big] + \bar{x}_2\left[x_3x_4\right] — ③$$

$$f(1,x_2,x_3,x_4) = x_2+x_3+x_4$$

$$f(1,x_2,x_3,x_4) = x_2\Big[\underbrace{1}_{f(1,1,x_3,x_4)}\Big] + \bar{x}_2\Big[\overbrace{x_3+x_4}^{f(1,0,x_3,x_4)}\Big]$$

$$f(1,x_2,x_3,x_4) = x_2+\bar{x}_2\Big[\underbrace{x_3+\bar{x}_3\underbrace{x_4}_{\to f(1,0,0,x_4)}}_{⑤}\Big] — ④$$

We notice that ⑤, ⑥ are same, so we can use same MUX to obtain this.

Final expansion: $f = x_1\left[x_2+\bar{x}_2\left[x_3+\bar{x}_3x_4\right]\right]+\bar{x}_1\left[x_2\left[x_3+\bar{x}_3x_4\right]+\bar{x}_2\left[x_3x_4\right]\right]$

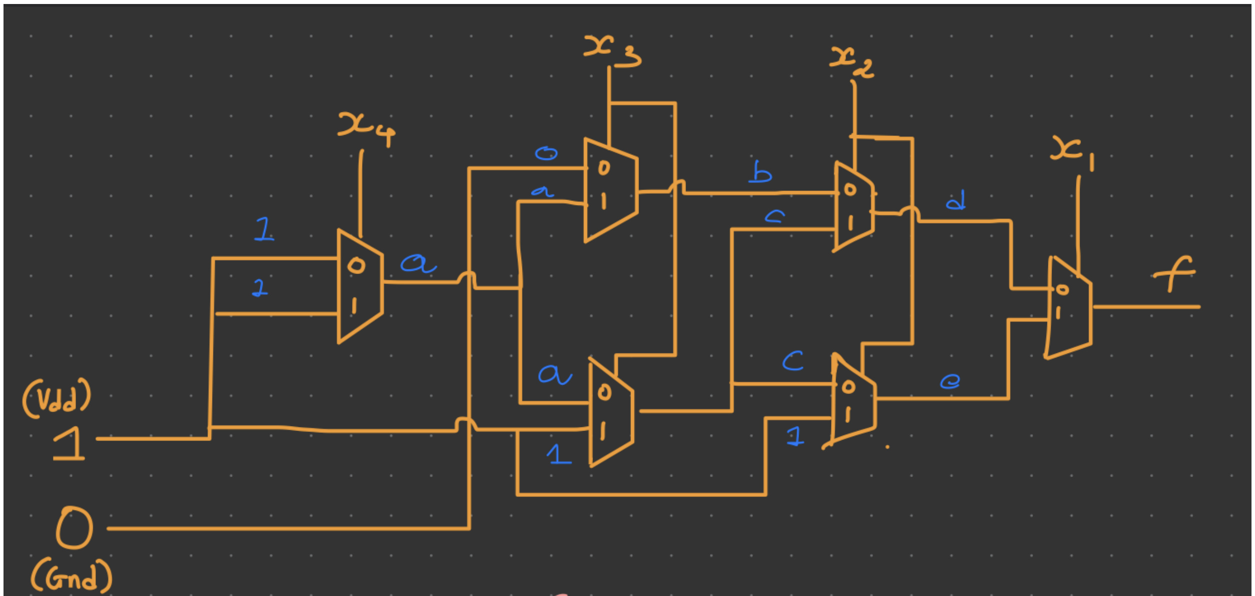


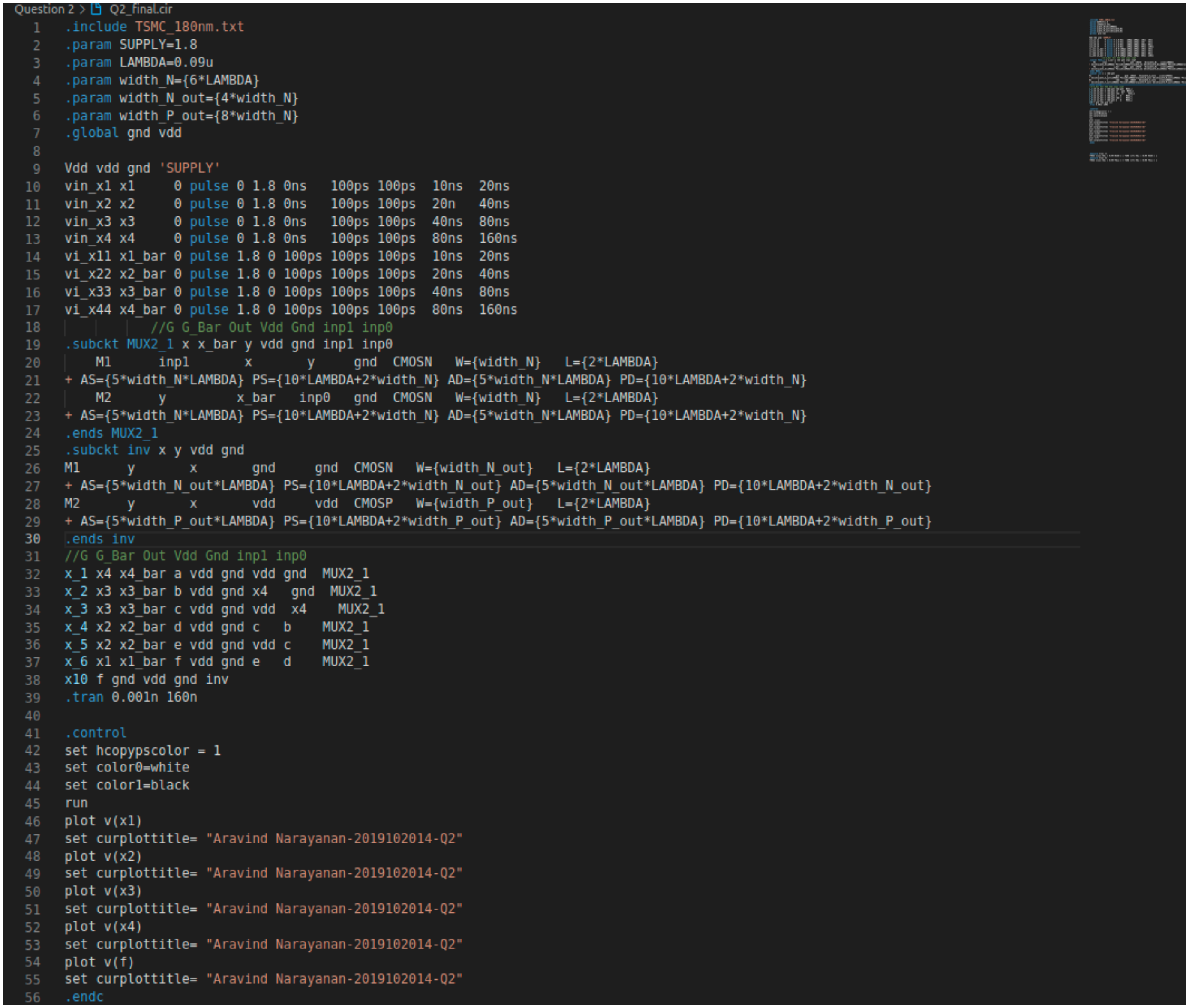
Fig: Circuit Diagram

B)

Parameters:

- Width = 6*Lambda calculated from Q1
- Load Capacitance = 4*Times the minimum sized inverter

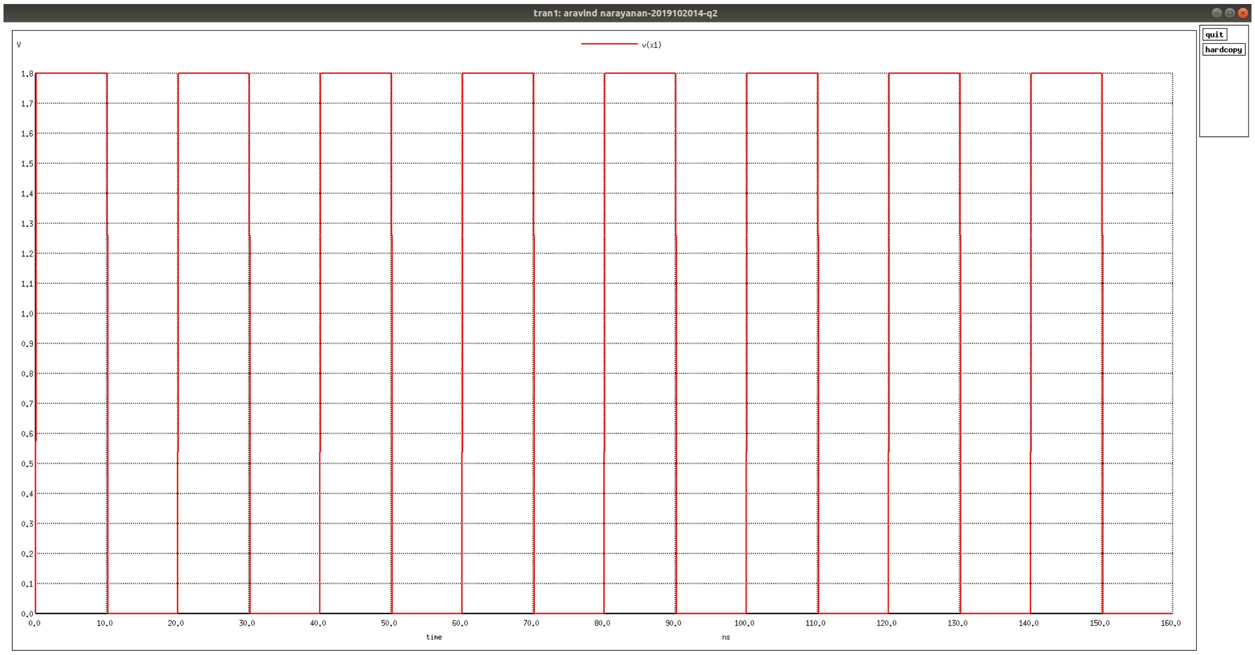
NETLIST:



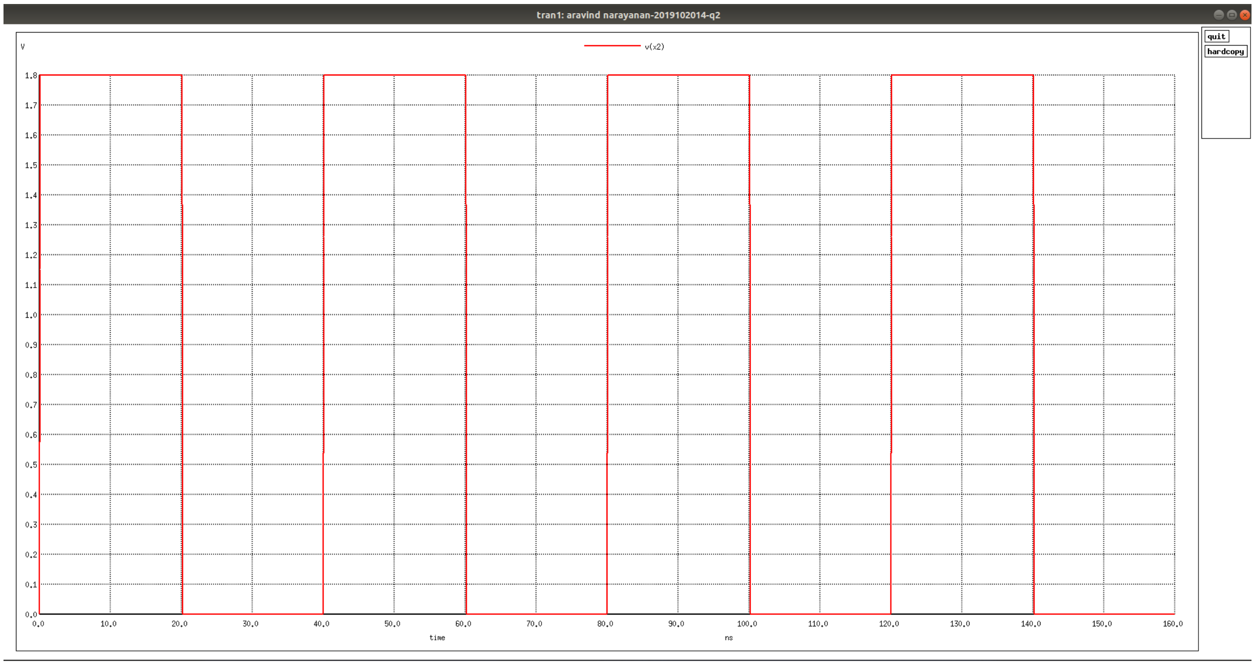
```
.include TSMC_180nm.txt
.param SUPPLY=1.8
.param LAMBDA=0.09u
.param width_N={6*LAMBDA}
.param width_N_out={4*width_N}
.param width_P_out={8*width_N}
.global gnd vdd

Vdd vdd gnd 'SUPPLY'
vin_x1 x1     0 pulse 0 1.8 0ns   100ps 100ps  10ns  20ns
vin_x2 x2     0 pulse 0 1.8 0ns   100ps 100ps  20n   40ns
vin_x3 x3     0 pulse 0 1.8 0ns   100ps 100ps  40ns  80ns
vin_x4 x4     0 pulse 0 1.8 0ns   100ps 100ps  80ns  160ns
vi_x11 x1_bar 0 pulse 1.8 0 100ps 100ps 100ps  10ns  20ns
vi_x22 x2_bar 0 pulse 1.8 0 100ps 100ps 100ps  20ns  40ns
vi_x33 x3_bar 0 pulse 1.8 0 100ps 100ps 100ps  40ns  80ns
vi_x44 x4_bar 0 pulse 1.8 0 100ps 100ps 100ps  80ns  160ns
           //G G_Bar Out Vdd Gnd inp1 inp0
.subckt MUX2_1 x x_bar y vdd gnd inp1 inp0
    M1      inp1       x       y     gnd  CMOSN   W={width_N}   L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}
    M2      y         x_bar   inp0   gnd  CMOSN   W={width_N}   L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}
.ends MUX2_1
.subckt inv x y vdd gnd
M1      y       x       gnd     gnd  CMOSN   W={width_N_out}   L={2*LAMBDA}
+ AS={5*width_N_out*LAMBDA} PS={10*LAMBDA+2*width_N_out} AD={5*width_N_out*LAMBDA} PD={10*LAMBDA+2*width_N_out}
M2      y       x       vdd     vdd  CMOSP   W={width_P_out}   L={2*LAMBDA}
+ AS={5*width_P_out*LAMBDA} PS={10*LAMBDA+2*width_P_out} AD={5*width_P_out*LAMBDA} PD={10*LAMBDA+2*width_P_out}
.ends inv
//G G_Bar Out Vdd Gnd inp1 inp0
x_1 x4 x4_bar a vdd gnd vdd gnd  MUX2_1
x_2 x3 x3_bar b vdd gnd x4   gnd  MUX2_1
x_3 x3 x3_bar c vdd gnd vdd  x4    MUX2_1
x_4 x2 x2_bar d vdd gnd c   b    MUX2_1
x_5 x2 x2_bar e vdd gnd vdd c    MUX2_1
x_6 x1 x1_bar f vdd gnd e   d    MUX2_1
x10 f gnd vdd gnd inv
.tran 0.001n 160n

.control
set hcopypscolor = 1
set color0=white
set color1=black
run
plot v(x1)
set curplottitle= "Aravind Narayanan-2019102014-Q2"
plot v(x2)
set curplottitle= "Aravind Narayanan-2019102014-Q2"
plot v(x3)
set curplottitle= "Aravind Narayanan-2019102014-Q2"
plot v(x4)
set curplottitle= "Aravind Narayanan-2019102014-Q2"
plot v(f)
set curplottitle= "Aravind Narayanan-2019102014-Q2"
.endc
```

Plots:

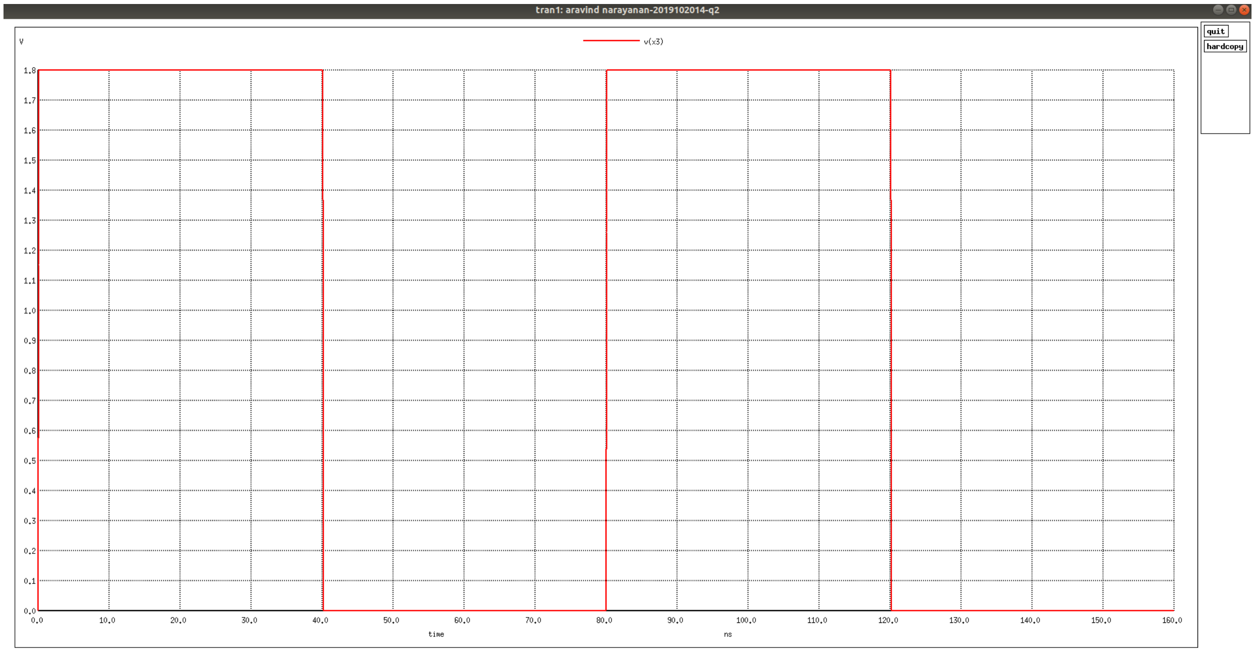
1) V(X1)



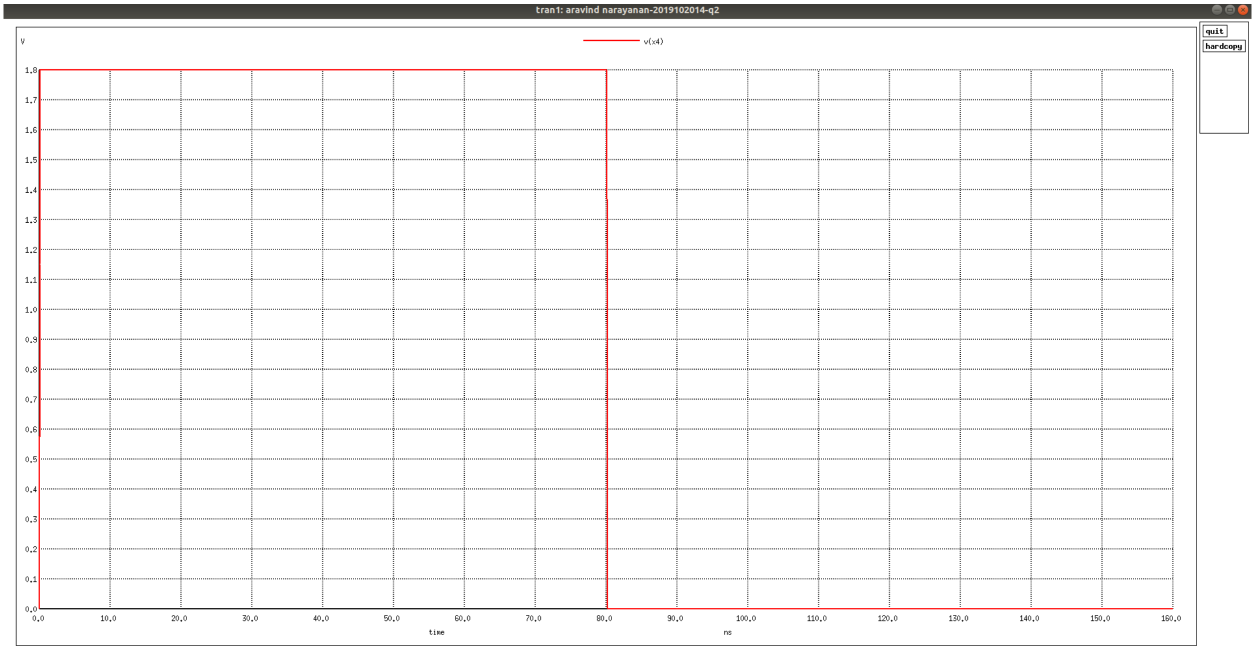
2) V(X2)



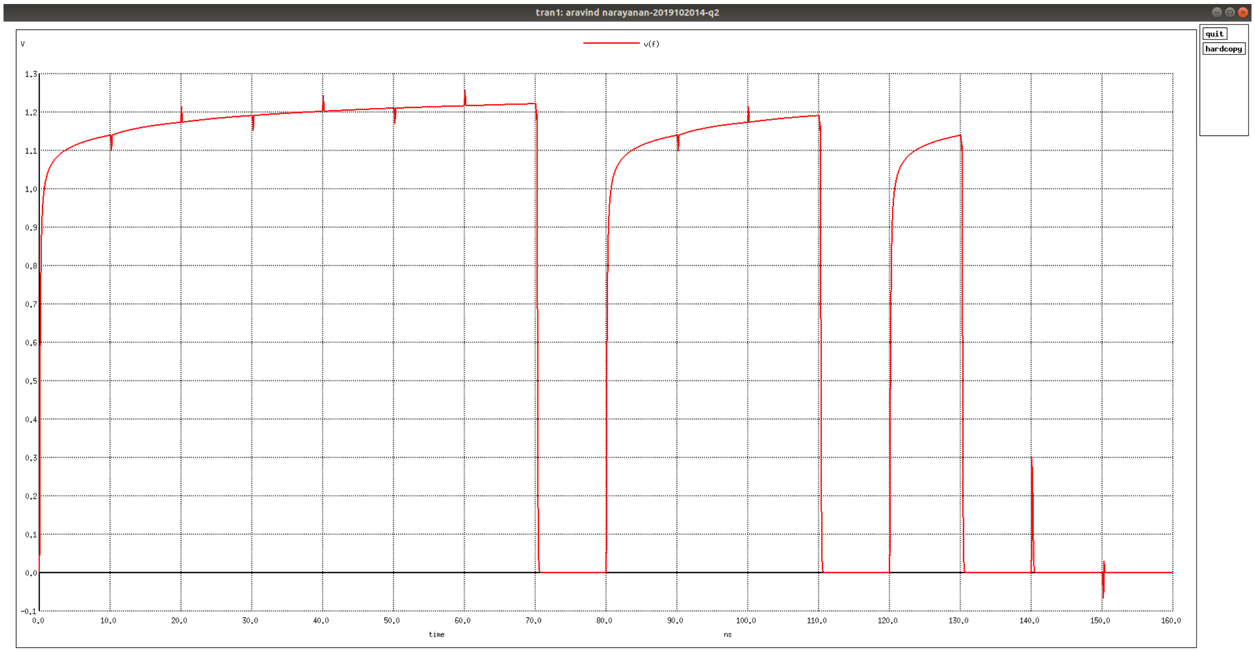
3) V(X3)


tran1: aravind narayanan-2019102014-q2
quit
hardcopy
V
v(x3)
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
100.0
110.0
120.0
130.0
140.0
150.0
160.0
time
ns

4) V(X4)


tran1: aravind narayanan-2019102014-q2
quit
hardcopy
V
v(x4)
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
100.0
110.0
120.0
130.0
140.0
150.0
160.0
time
ns

5) V(f)



Input-Output Table:

NOTE: Here 0 = LOW and 1 = HIGH

Time Range(ns)	X1	X2	X3	X4	Expected	f
0-10	1	1	1	1	1	1
10-20	0	1	1	1	1	1
20-30	1	0	1	1	1	1
30-40	0	0	1	1	1	1
40-50	1	1	0	1	1	1
50-60	0	1	0	1	1	1
60-70	1	0	0	1	1	1
70-80	0	0	0	1	0	0
80-90	1	1	1	1	1	1
90-100	0	1	1	0	1	1
100-110	1	0	1	0	1	1
110-120	0	0	1	0	0	0
120-130	1	1	0	0	1	1

130-140	0	1	0	0	0	0
140-150	1	0	0	0	0	0
150-160	0	0	0	0	0	0

We notice that all the expected values calculated manually match the f value obtained thereby verifying the working of the circuit.

Observations:

We notice that though the v(f) gives correct MUX outputs, it does not reach Vdd(HIGH) completely but reaches gnd(LOW) properly.

Why does NMOS pass weak 1(HIGH) while it passes strong 0(LOW)?

ANS: NMOS is a symmetric device where to pass a strong 1, gate should always be at 1. But when the output reaches Vdd-Vth, V_gate - V_out = Vth, which is the minimum voltage needed for the NMOS to be in ON state for current to flow. So, after this point, the NMOS switches off. Therefore, the maximum voltage the NMOS can reach is VDD-Vth. This issue does not arrive when reaching 0, therefore it is able to pass a strong 0. This is also the reason NMOS is used as a pull-down network.

C)

For finding minimum transition time, we need to find the path that requires passing through the lowest number of MUXes(NMOS) for both t_PLH and t_PHL.

Combination for faster possible charging and discharging:

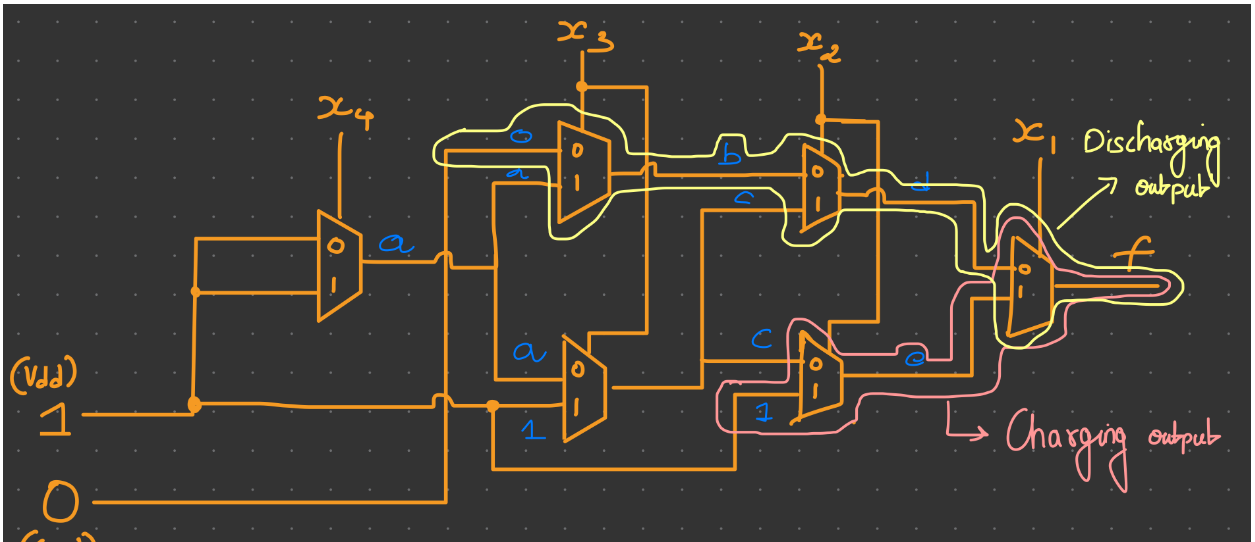


Fig: Diagram showing charging and discharging output

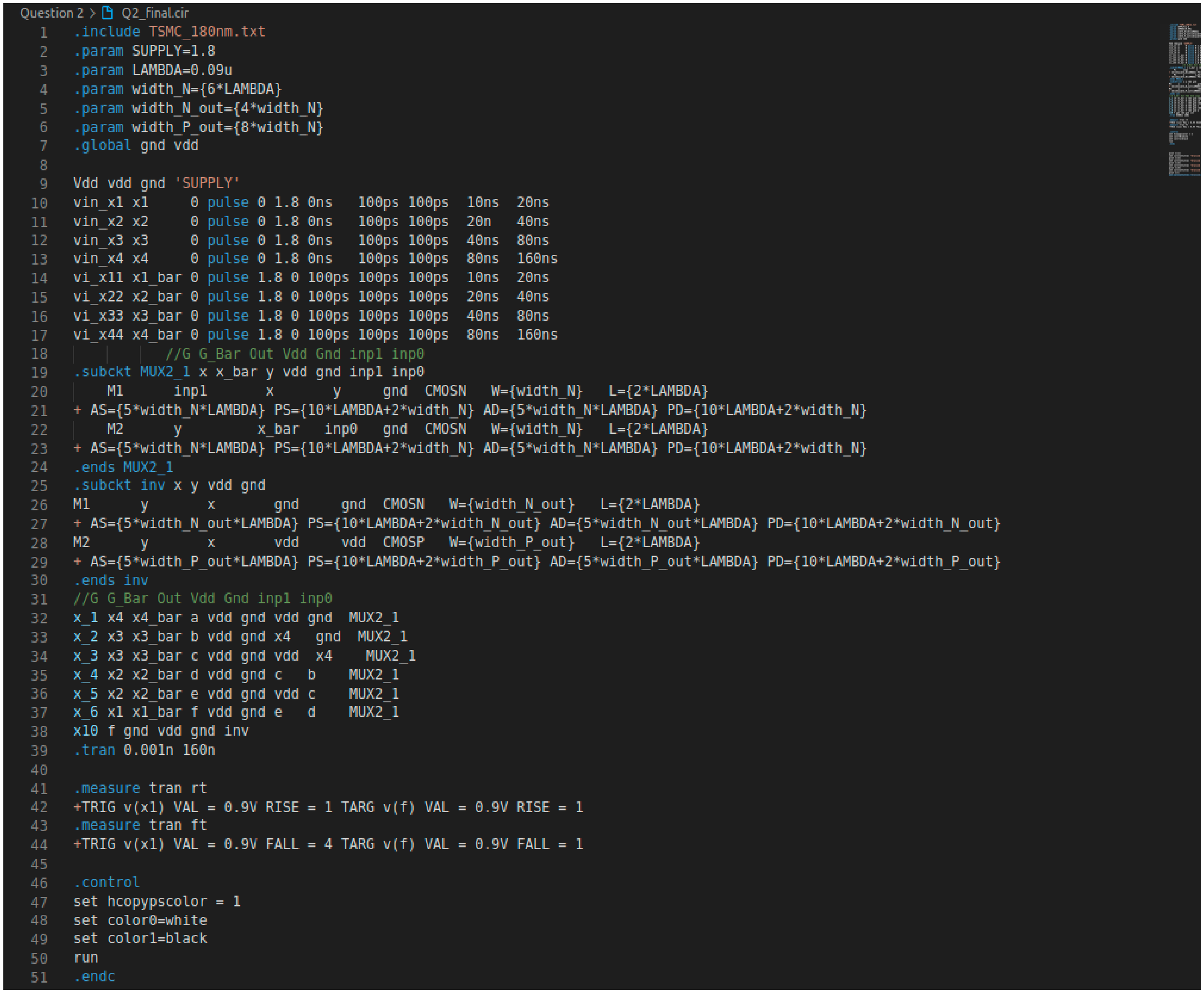
- For discharging the output, 3 MUXES are there.
- For charging, the output, 2 MUXES are there.

INPUT COMBINATION:

	X1	X2	X3	X4	f
Discharging	0	0	0	X	0
Charging	1	1	X	X	1

X = Don’t Care State (can be 0 or 1) but doesn’t affect output state.

Netlist to compute t_PLH and t_PHL:



Question 2 > Q2_final.cir

```
.include TSMC_180nm.txt
.param SUPPLY=1.8
.param LAMBDA=0.09u
.param width_N={6*LAMBDA}
.param width_N_out={4*width_N}
.param width_P_out={8*width_N}
.global gnd vdd

Vdd vdd gnd 'SUPPLY'
vin_x1 x1     0 pulse 0 1.8 0ns   100ps 100ps  10ns  20ns
vin_x2 x2     0 pulse 0 1.8 0ns   100ps 100ps  20n   40ns
vin_x3 x3     0 pulse 0 1.8 0ns   100ps 100ps  40ns  80ns
vin_x4 x4     0 pulse 0 1.8 0ns   100ps 100ps  80ns  160ns
vi_x11 x1_bar 0 pulse 1.8 0 100ps 100ps 100ps  10ns  20ns
vi_x22 x2_bar 0 pulse 1.8 0 100ps 100ps 100ps  20ns  40ns
vi_x33 x3_bar 0 pulse 1.8 0 100ps 100ps 100ps  40ns  80ns
vi_x44 x4_bar 0 pulse 1.8 0 100ps 100ps 100ps  80ns  160ns
                      //G G_Bar Out Vdd Gnd inp1 inp0
.subckt MUX2_1 x x_bar y vdd gnd inp1 inp0
    M1      inp1       x       y     gnd  CMOSN   W={width_N}   L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}
    M2      y          x_bar   inp0   gnd  CMOSN   W={width_N}   L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}
.ends MUX2_1
.subckt inv x y vdd gnd
M1      y       x       gnd     gnd  CMOSN   W={width_N_out}   L={2*LAMBDA}
+ AS={5*width_N_out*LAMBDA} PS={10*LAMBDA+2*width_N_out} AD={5*width_N_out*LAMBDA} PD={10*LAMBDA+2*width_N_out}
M2      y       x       vdd     vdd  CMOSP   W={width_P_out}   L={2*LAMBDA}
+ AS={5*width_P_out*LAMBDA} PS={10*LAMBDA+2*width_P_out} AD={5*width_P_out*LAMBDA} PD={10*LAMBDA+2*width_P_out}
.ends inv
//G G_Bar Out Vdd Gnd inp1 inp0
x_1 x4 x4_bar a vdd gnd vdd gnd  MUX2_1
x_2 x3 x3_bar b vdd gnd x4   gnd  MUX2_1
x_3 x3 x3_bar c vdd gnd vdd  x4    MUX2_1
x_4 x2 x2_bar d vdd gnd c   b    MUX2_1
x_5 x2 x2_bar e vdd gnd vdd c    MUX2_1
x_6 x1 x1_bar f vdd gnd e   d    MUX2_1
x10 f gnd vdd gnd inv
.tran 0.001n 160n

.measure tran rt
+TRIG v(x1) VAL = 0.9V RISE = 1 TARG v(f) VAL = 0.9V RISE = 1
.measure tran ft
+TRIG v(x1) VAL = 0.9V FALL = 4 TARG v(f) VAL = 0.9V FALL = 1

.control
set hcopypscolor = 1
set color0=white
set color1=black
run
.endc
```

Observation:

	Minimum Transition Time(ns)
t_{PLH}	301.0487ns
t_{PHL}	28.02271ns

- The minimum transition for output to change from Low to High(t_{PLH}) is significantly larger than that of t_{PHL}.
- This is mostly due to the NMOS's ability to charge completely. Here the NMOS pass transistors in the MUX are cascaded due to which output is only Vdd-Vth per mosfet and this

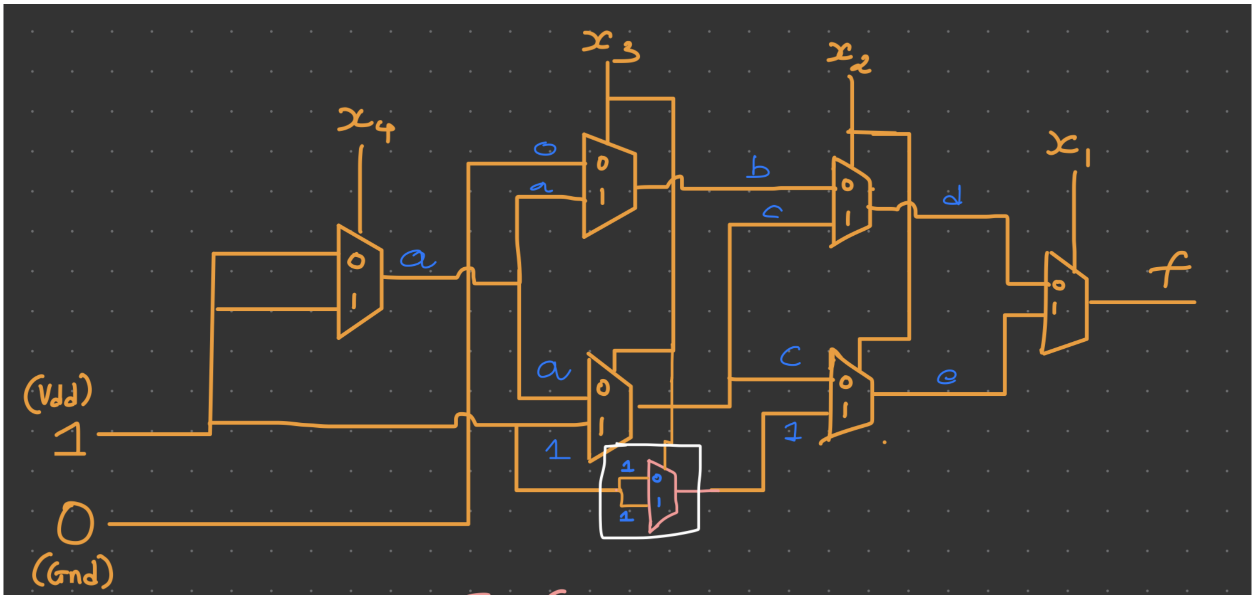
also leads to **static power consumption** + **slower transition.**

d)

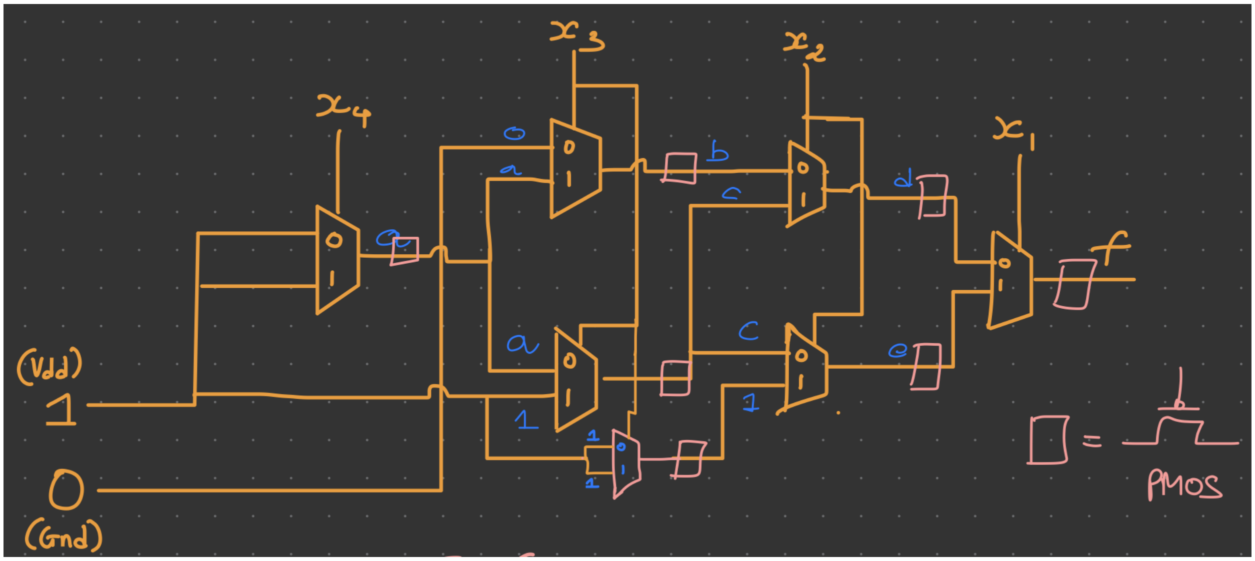
Yes, there is a difference between t_{PLH} and t_{PHL} whether transition time taken to go from low to 0. We also notice that the NMOS pass transistor doesn't reach the full amplitude, which directly contributes to the delay factor. Also more time is taken for NMOS to charge as we know that it is not a best candidate to produce strong 1s.

Methods:

- As seen in C), a lower number of muxes are used in the charging side, so we need to balance them out so that there are the same transistors on both sides. Modifications to be made:
    - Add 1 MUX to the charging side so that both need 3 MUXES(2 more transistors added) .



- Next issue is the time taken to charge the NMOS transistors. This can be neutralised by adding PMOS (gate attached to gnd so that it is always ON) to the output of each of the MUX as PMOS exhibits dual properties of the NMOS. By doing this we increase the fall time and decrease the rise time making the difference between each other significantly lesser.



x_4
x_3
x_2
x_1
0
1
a
b
c
d
e
f
(Vdd)
1
0
(Gnd)
= PMOS

Question 3

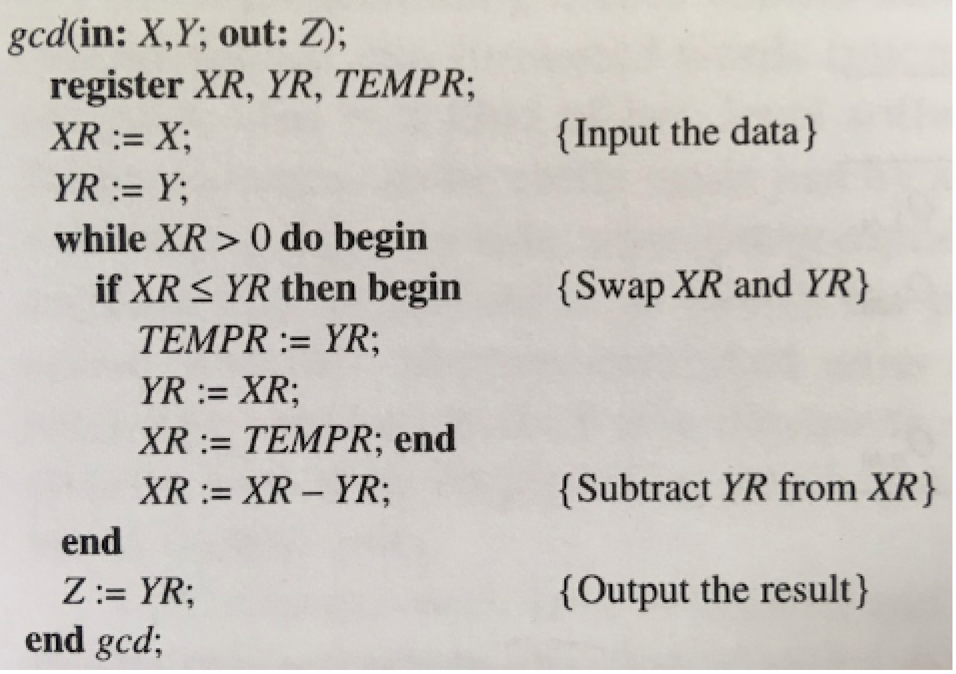
As it's a Moore type FSM, the output signal values depend only on the current state and are independent of the input combinations.

This is a special purpose processor that computes the Greatest Common Divisor(GCD) of two 5-bit positive integers X and Y.

gcd(X,Y) is defined as the largest integer that divides exactly into both A" and Y.

Main Procedure:

- Subtract the smaller number of the two numbers from the other repeatedly until we obtain a number that divides the other.



```
gcd(in: X,Y; out: Z);
    register XR, YR, TEMPR;
    XR := X;                            {Input the data}
    YR := Y;
    while $XR > 0$ do begin
        if $XR \le YR$ then begin       {Swap XR and YR}
            TEMPR := YR;
            YR := XR;
            XR := TEMPR; end
            $XR := XR - YR$;            {Subtract YR from XR}
    end
    Z := YR;                            {Output the result}
end gcd;
```

A) State Diagram for Control Unit

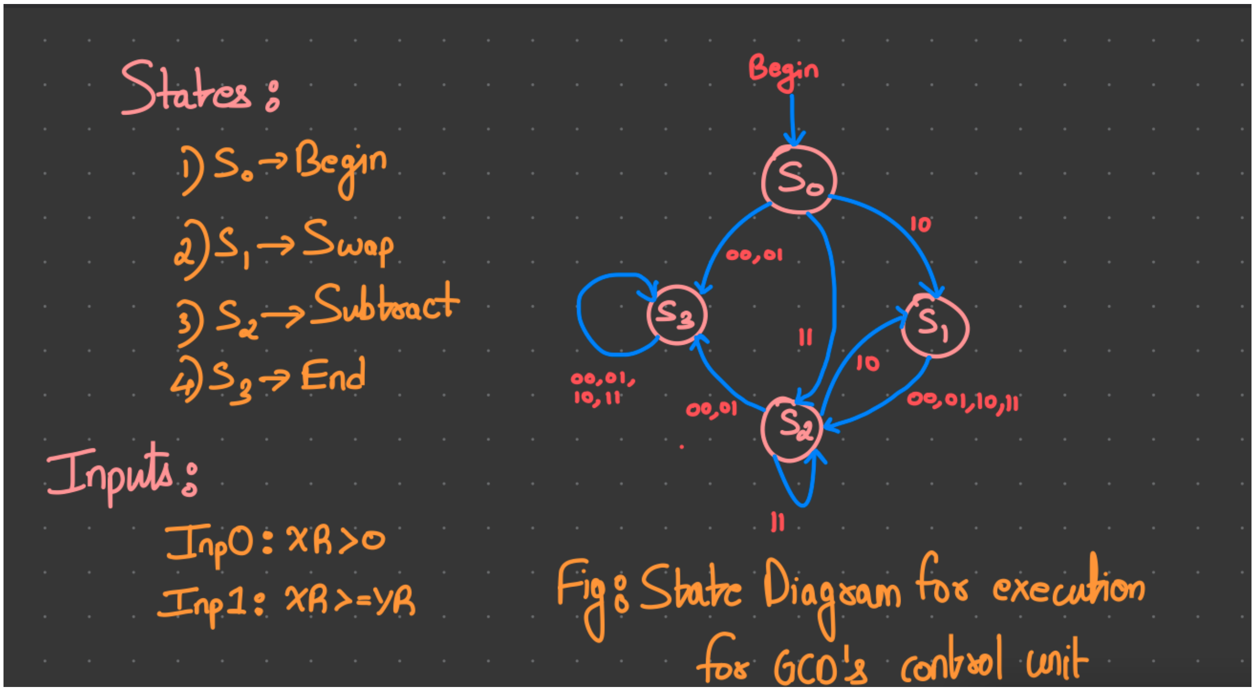


Fig: State Diagram for execution for GCO's control unit

B) Circuit Diagram for Control Unit

Signals Control Unit:

❏ **Outputs:**

- **Load XR -** Load input data X to the register XR independently.
- **Load YR -** Load input data Y to the register YR independently.
- **Swap -** Triggers swap operation to swap XR and YR values.
- **Subtract -** Controls subtraction by routing output of subtractor to XR.

❏ **Inputs:**

- **Asynchronous Reset Signal**
- **Comparison Signals:**
  - **(XR >= YR)**
  - **(XR>0)**

States:

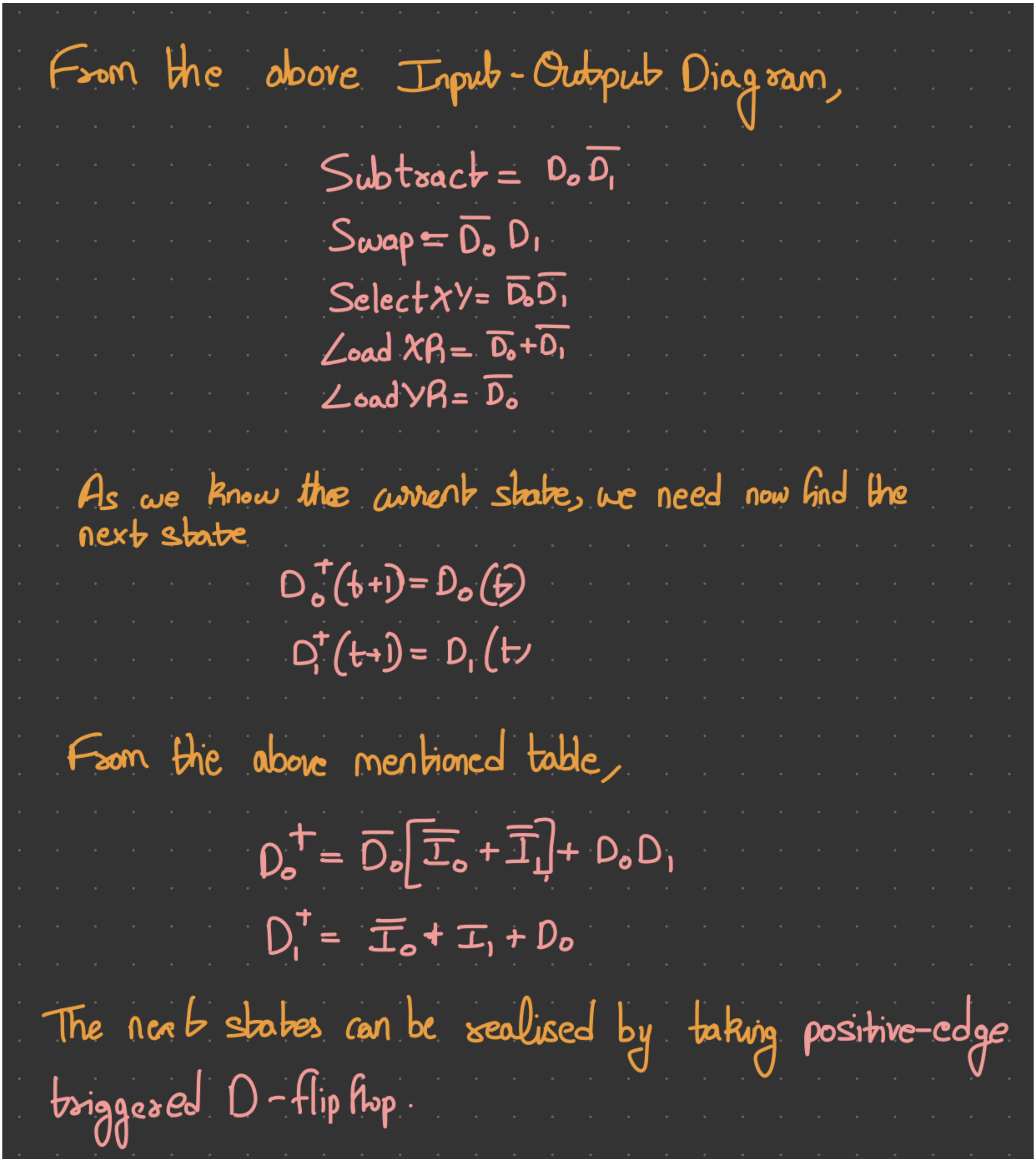
These states are defined by D0 D1 in the binary pattern for convenience.

- **S0(Begin):**
  - Start state is entered when Reset becomes 1
  - Triggers load XR and load YR
  - Can lead to a Swap or Subtract Operation

- **S1(Swap)**:
  - Triggers the swap operation
  - Leads to the subtract operation state
- **S2(Subtract)**
  - Triggers the subtract operation
  - Then loads the updated value to XR
  - Can lead to Swap, Subtract or End state based on conditions.
- **S3(End)**
  - Reaches this state if gcd(X,Y) has been found
  - The final value would reside in register YR

States	Binary Pattern
S0	00
S1	01
S2	10
S3	11

	Next State for Input	Next State for Input	Next State for Input	Next State for Input
Current State	00	01	10	11
S0(Begin)	S3	S3	S1	S2
S1(Swap)	S2	S2	S2	S2
S2(Subtract)	S3	S3	S1	S2
S3(End)	S3	S3	S3	S3



From the above Input - Output Diagram,

$$\text{Subtract} = D_0 \overline{D_1}$$

$$\text{Swap} = \overline{D_0} D_1$$

$$\text{Select } XY = \overline{D_0}\,\overline{D_1}$$

$$\text{Load } XR = \overline{D_0} + \overline{D_1}$$

$$\text{Load } YR = \overline{D_0}$$

As we know the current state, we need now find the next state

$$D_0^+(t+1) = D_0(t)$$

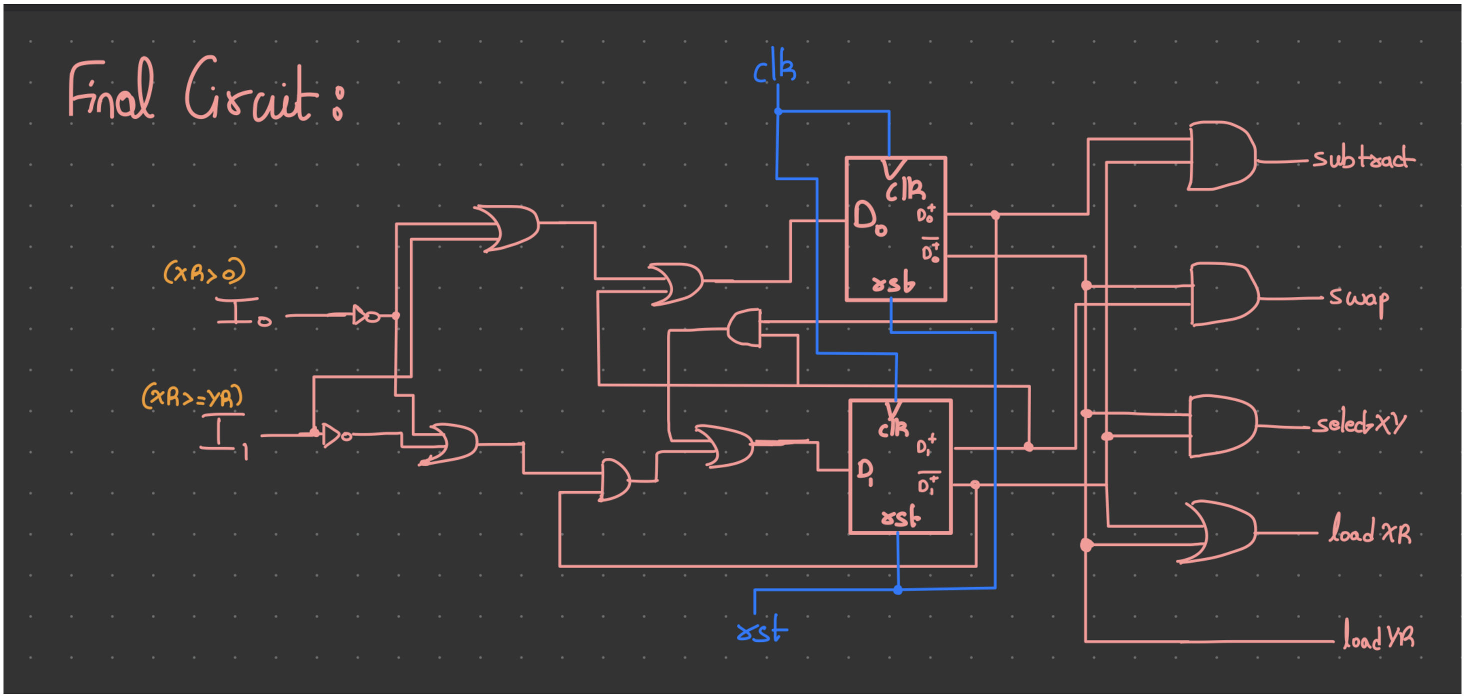
$$D_1^+(t+1) = D_1(t)$$

From the above mentioned table,

$$D_0^+ = \overline{D_0}\left[\overline{I_0} + \overline{I_1}\right] + D_0 D_1$$

$$D_1^+ = \overline{I_0} + I_1 + D_0$$

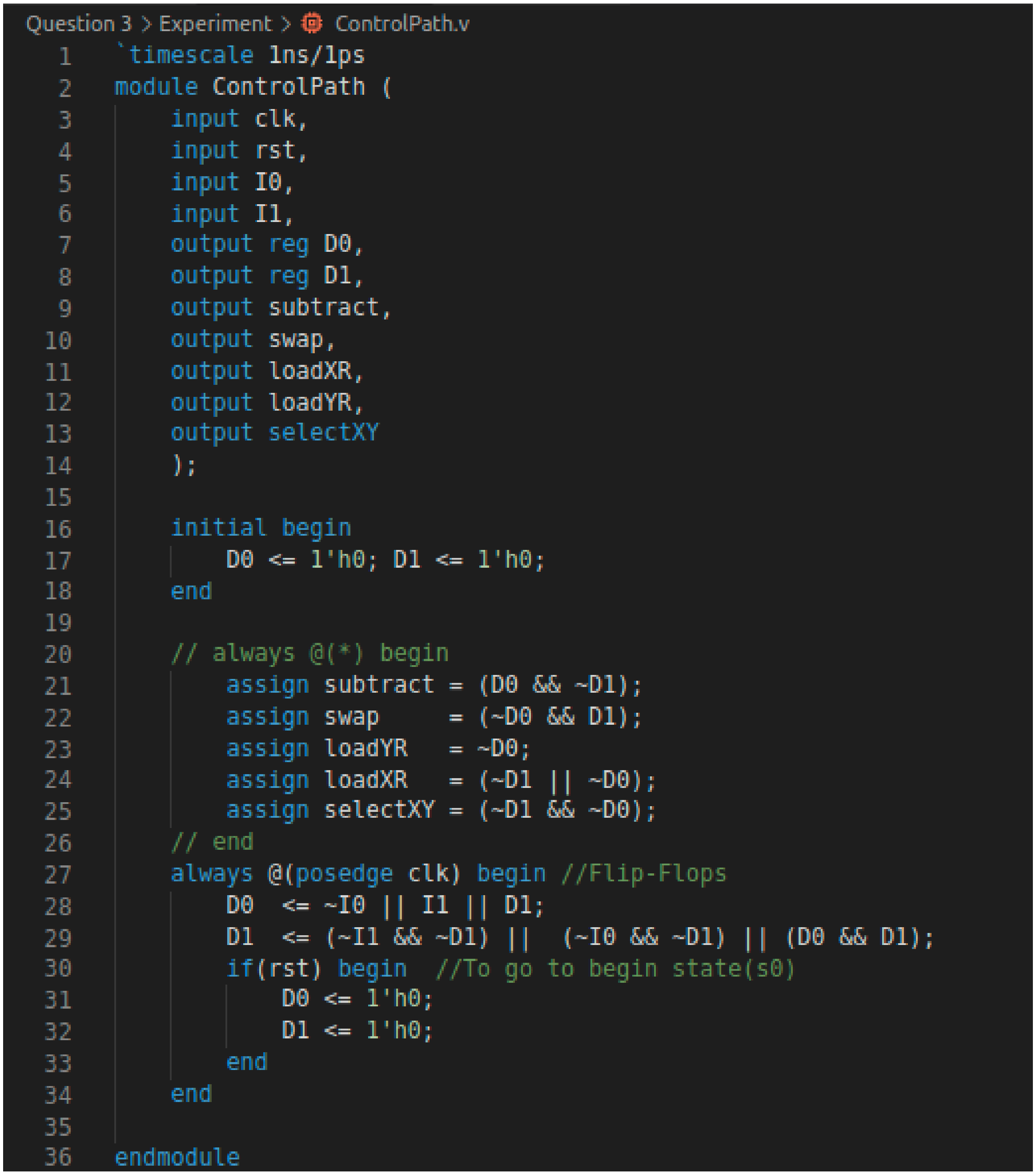
The next states can be realised by taking positive-edge triggered D-flip flop.



C) ControlPath Simulation

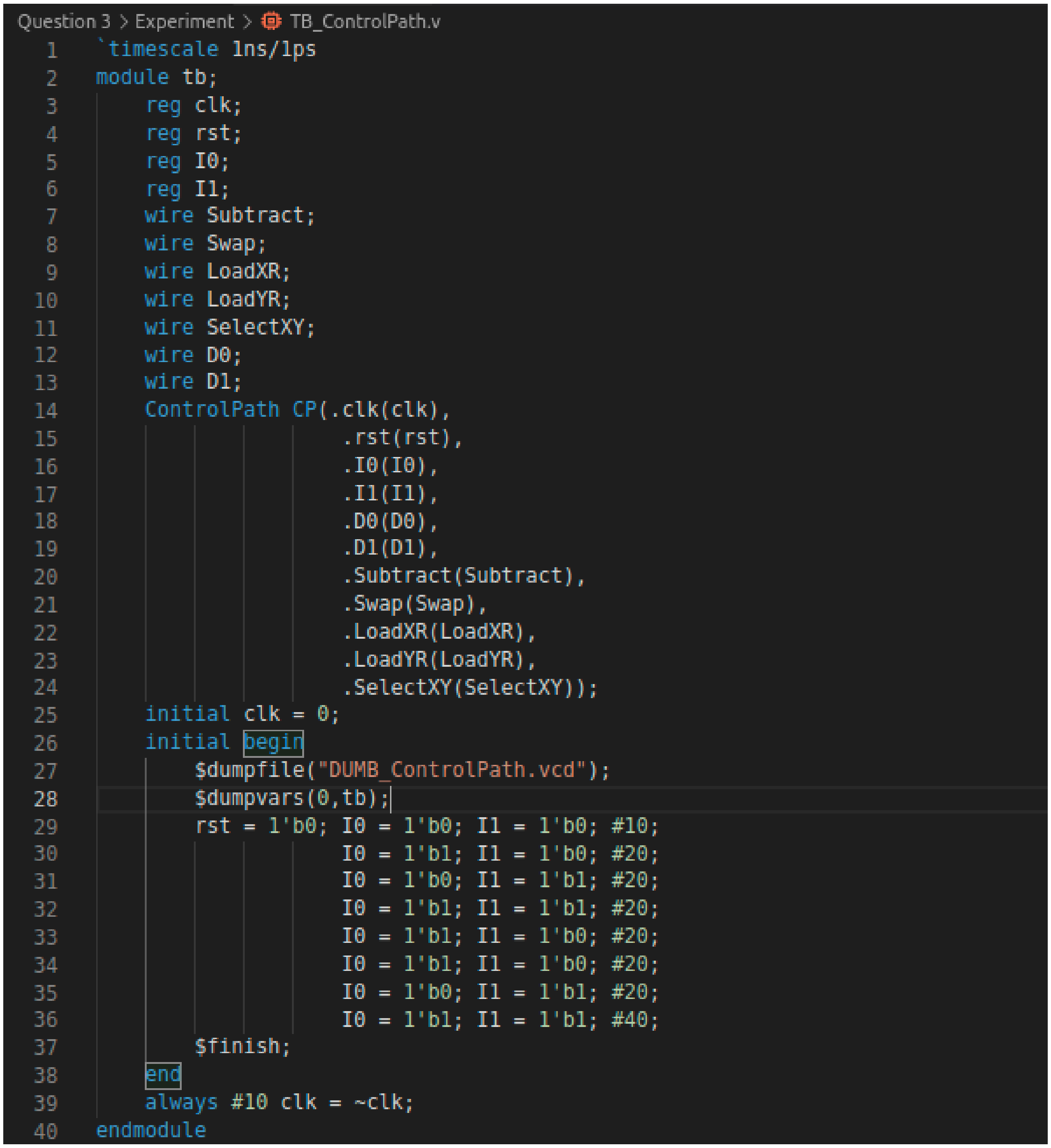
We show the working of the control unit by showing the change in state and assigning the 5 parameters when the inputs from the testbench is given(in actual, it will be given by the datapath module. We verify if this works correctly by the state diagram and table drawn in B) part.

ControlPath Module Code:



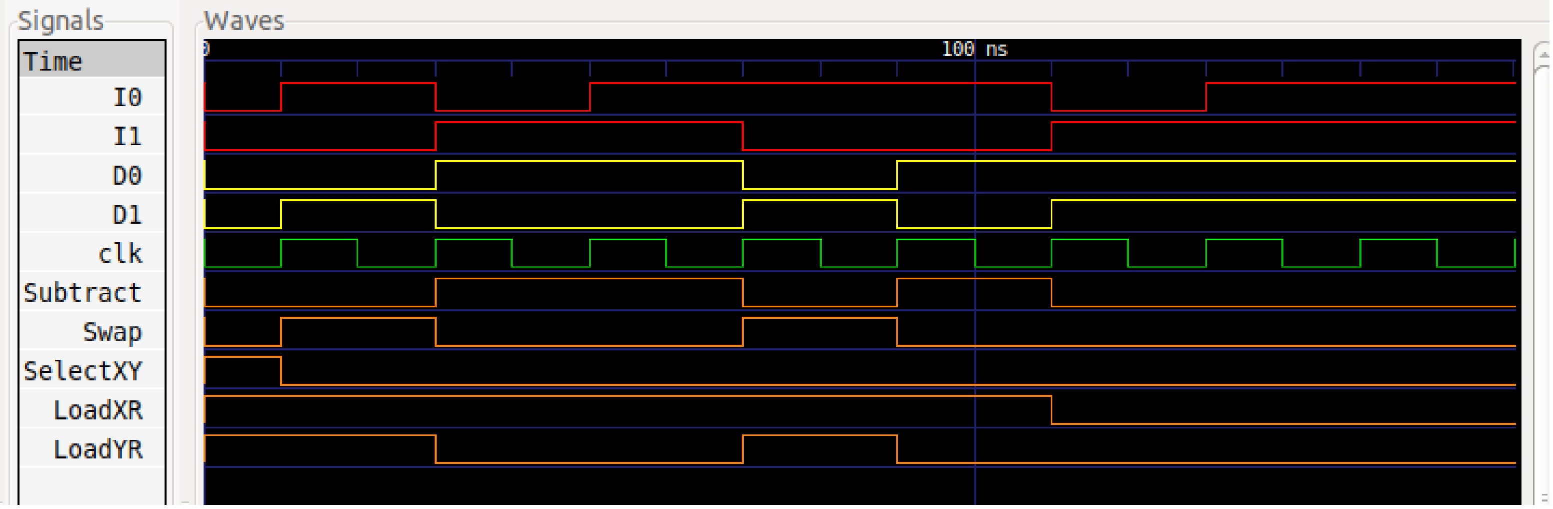
```
`timescale 1ns/1ps
module ControlPath (
    input clk,
    input rst,
    input I0,
    input I1,
    output reg D0,
    output reg D1,
    output subtract,
    output swap,
    output loadXR,
    output loadYR,
    output selectXY
    );

    initial begin
        D0 <= 1'h0; D1 <= 1'h0;
    end

    // always @(*) begin
        assign subtract = (D0 && ~D1);
        assign swap     = (~D0 && D1);
        assign loadYR   = ~D0;
        assign loadXR   = (~D1 || ~D0);
        assign selectXY = (~D1 && ~D0);
    // end
    always @(posedge clk) begin //Flip-Flops
        D0  <= ~I0 || I1 || D1;
        D1  <= (~I1 && ~D1) ||  (~I0 && ~D1) || (D0 && D1);
        if(rst) begin  //To go to begin state(s0)
            D0 <= 1'h0;
            D1 <= 1'h0;
        end
    end

endmodule
```

ControlPath TestBench Code:



```verilog
`timescale 1ns/1ps
module tb;
    reg clk;
    reg rst;
    reg I0;
    reg I1;
    wire Subtract;
    wire Swap;
    wire LoadXR;
    wire LoadYR;
    wire SelectXY;
    wire D0;
    wire D1;
    ControlPath CP(.clk(clk),
                    .rst(rst),
                    .I0(I0),
                    .I1(I1),
                    .D0(D0),
                    .D1(D1),
                    .Subtract(Subtract),
                    .Swap(Swap),
                    .LoadXR(LoadXR),
                    .LoadYR(LoadYR),
                    .SelectXY(SelectXY));
    initial clk = 0;
    initial begin
        $dumpfile("DUMB_ControlPath.vcd");
        $dumpvars(0,tb);
        rst = 1'b0; I0 = 1'b0; I1 = 1'b0; #10;
                    I0 = 1'b1; I1 = 1'b0; #20;
                    I0 = 1'b0; I1 = 1'b1; #20;
                    I0 = 1'b1; I1 = 1'b1; #20;
                    I0 = 1'b1; I1 = 1'b0; #20;
                    I0 = 1'b1; I1 = 1'b0; #20;
                    I0 = 1'b0; I1 = 1'b1; #20;
                    I0 = 1'b1; I1 = 1'b1; #40;
        $finish;
    end
    always #10 clk = ~clk;
endmodule
```

Simulation Result:



Observation Table;

S.No	INPUTS		Previous State	Present State	Outputs					Explanation
	I0	I1	(D0 D1)	(D0 D1)	subtract	swap	SelectXY	loadXR	loadYR	
1	0	0	00	00	0	0	1	1	1	As it is inp =00 we need to load values and select them for register. **State (00)**
2	1	0	00	01	1	0	0	1	0	As values are loaded now it goes to the subtract state along with a need to load the value into XR again.**State(S1)**
3	0	1	01	10	0	1	0	1	1	Now current state is **State(S2)** due to which we need to swap the value and the need to load the values arises
4	1	1	10	10	1	0	0	1	0	As values are loaded now it goes to the subtract state along with a need to load the value into XR again.**State(S1)**
5	1	0	10	01	0	1	0	1	1	Now current state is **State(S2)** due to

										which we need to swap the value and the need to load the values arises
6	1	0	01	10	1	0	0	1	0	As values are loaded now it goes to the subtract state along with a need to load the value into XR again.**State(S1)**
7	0	1	10	11	0	0	0	0	0	It has finally reached **State(S3)** after which any combination of input would always result in same D0 D1 value
8	1	1	11	11	0	0	0	0	0	As it is in State(S3) no change is there and the flags are all zero.

From the above table and explanation we see that the functioning of the Control Unit works properly where the outputs are given based on the positive-edge triggered states.

This control unit works along with the datapath unit and calculates the GCD on it's own when only the input terms along with reset option is given.

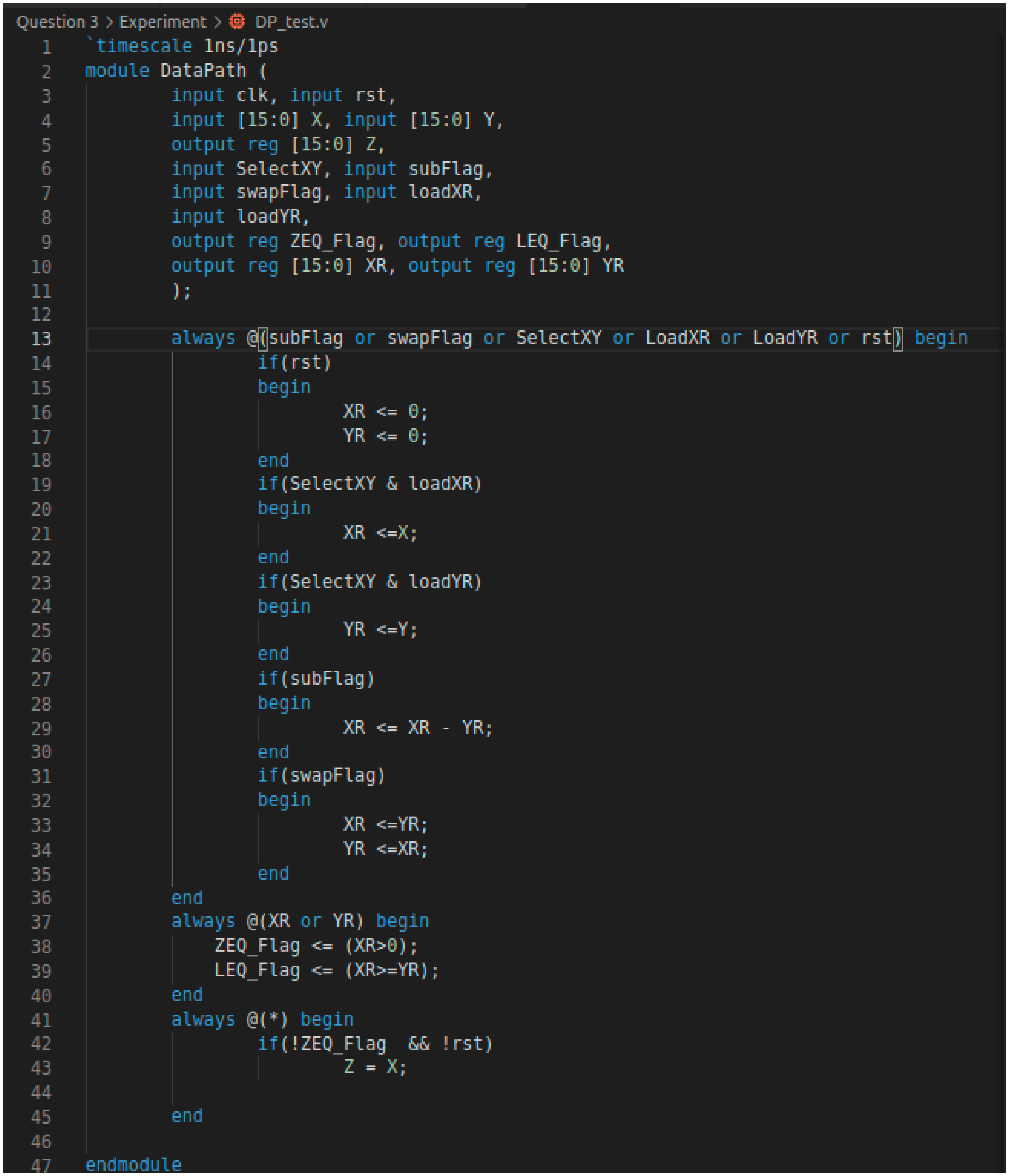
D) Datapath Unit

Components of DataPath:

- **Registers(5 bit length):**
  - XR
  - YR
- **Subtractor**
- **Comparators:**
  - **ZEQ_Flag(I0)**: Check for XR>0
  - **LEQ_Flag(I1)**: Check for XR>=YR

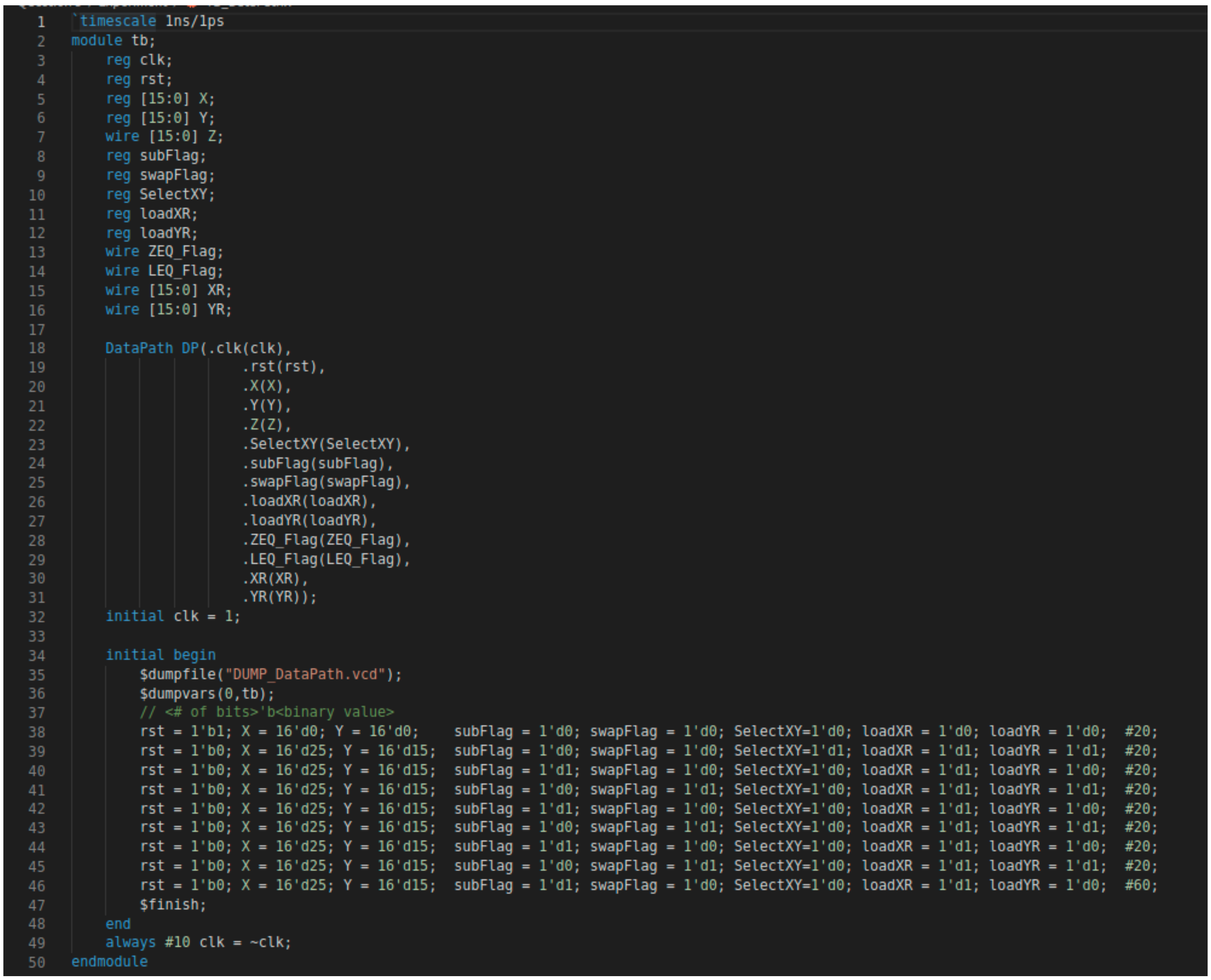
DataPath Module Code:

NOTE: The datapath does not call the control path but rather inputs given by testbench only for test purposes. For actual final combined testing, the control path will be called by the Datapath instead.



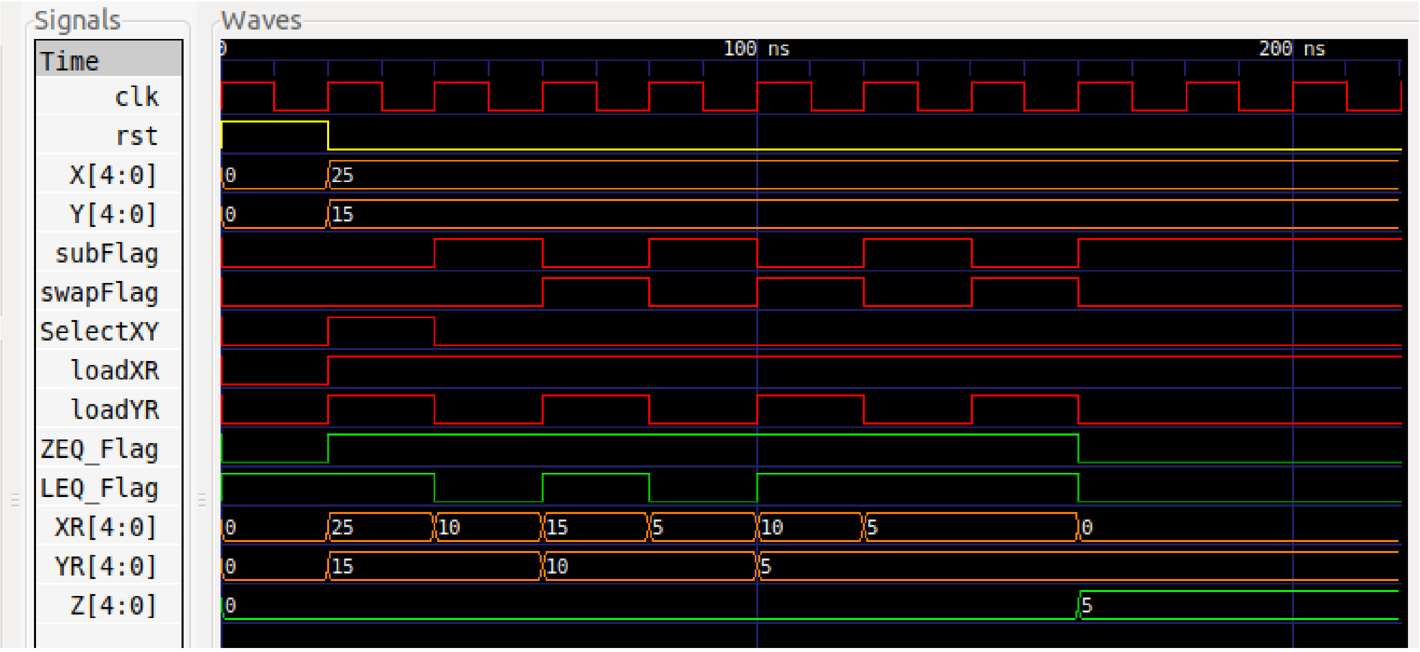
```
`timescale 1ns/1ps
module DataPath (
        input clk, input rst,
        input [15:0] X, input [15:0] Y,
        output reg [15:0] Z,
        input SelectXY, input subFlag,
        input swapFlag, input loadXR,
        input loadYR,
        output reg ZEQ_Flag, output reg LEQ_Flag,
        output reg [15:0] XR, output reg [15:0] YR
        );

        always @(subFlag or swapFlag or SelectXY or LoadXR or LoadYR or rst) begin
                if(rst)
                begin
                        XR <= 0;
                        YR <= 0;
                end
                if(SelectXY & loadXR)
                begin
                        XR <=X;
                end
                if(SelectXY & loadYR)
                begin
                        YR <=Y;
                end
                if(subFlag)
                begin
                        XR <= XR - YR;
                end
                if(swapFlag)
                begin
                        XR <=YR;
                        YR <=XR;
                end
        end
        always @(XR or YR) begin
            ZEQ_Flag <= (XR>0);
            LEQ_Flag <= (XR>=YR);
        end
        always @(*) begin
                if(!ZEQ_Flag  && !rst)
                        Z = X;

        end

endmodule
```

DataPath TestBench Code:



```verilog
`timescale 1ns/1ps
module tb;
    reg clk;
    reg rst;
    reg [15:0] X;
    reg [15:0] Y;
    wire [15:0] Z;
    reg subFlag;
    reg swapFlag;
    reg SelectXY;
    reg loadXR;
    reg loadYR;
    wire ZEQ_Flag;
    wire LEQ_Flag;
    wire [15:0] XR;
    wire [15:0] YR;

    DataPath DP(.clk(clk),
                    .rst(rst),
                    .X(X),
                    .Y(Y),
                    .Z(Z),
                    .SelectXY(SelectXY),
                    .subFlag(subFlag),
                    .swapFlag(swapFlag),
                    .loadXR(loadXR),
                    .loadYR(loadYR),
                    .ZEQ_Flag(ZEQ_Flag),
                    .LEQ_Flag(LEQ_Flag),
                    .XR(XR),
                    .YR(YR));
    initial clk = 1;

    initial begin
        $dumpfile("DUMP_DataPath.vcd");
        $dumpvars(0,tb);
        // <# of bits>'b<binary value>
        rst = 1'b1; X = 16'd0; Y = 16'd0;    subFlag = 1'd0; swapFlag = 1'd0; SelectXY=1'd0; loadXR = 1'd0; loadYR = 1'd0;  #20;
        rst = 1'b0; X = 16'd25; Y = 16'd15;  subFlag = 1'd0; swapFlag = 1'd0; SelectXY=1'd1; loadXR = 1'd1; loadYR = 1'd1;  #20;
        rst = 1'b0; X = 16'd25; Y = 16'd15;  subFlag = 1'd1; swapFlag = 1'd0; SelectXY=1'd0; loadXR = 1'd1; loadYR = 1'd0;  #20;
        rst = 1'b0; X = 16'd25; Y = 16'd15;  subFlag = 1'd0; swapFlag = 1'd1; SelectXY=1'd0; loadXR = 1'd1; loadYR = 1'd1;  #20;
        rst = 1'b0; X = 16'd25; Y = 16'd15;  subFlag = 1'd1; swapFlag = 1'd0; SelectXY=1'd0; loadXR = 1'd1; loadYR = 1'd0;  #20;
        rst = 1'b0; X = 16'd25; Y = 16'd15;  subFlag = 1'd0; swapFlag = 1'd1; SelectXY=1'd0; loadXR = 1'd1; loadYR = 1'd1;  #20;
        rst = 1'b0; X = 16'd25; Y = 16'd15;  subFlag = 1'd1; swapFlag = 1'd0; SelectXY=1'd0; loadXR = 1'd1; loadYR = 1'd0;  #20;
        rst = 1'b0; X = 16'd25; Y = 16'd15;  subFlag = 1'd0; swapFlag = 1'd1; SelectXY=1'd0; loadXR = 1'd1; loadYR = 1'd1;  #20;
        rst = 1'b0; X = 16'd25; Y = 16'd15;  subFlag = 1'd1; swapFlag = 1'd0; SelectXY=1'd0; loadXR = 1'd1; loadYR = 1'd0;  #60;
        $finish;
    end
    always #10 clk = ~clk;
endmodule
```

Datapath Unit Simulation:



Parameters involved in simulation:

S.No	Inputs to DataPath Module					Comparators		Registers		GCD Value
	subtract	swap	SelectXY	loadXR	loadYR	XR>0 (ZEQ_Flag)	XR>=YR (LEQ_FLAG)	XR	YR	Z
1	0	0	1	1	1	1	1	25	15	0
2	1	0	0	1	0	1	0	10	15	0
3	0	1	0	1	1	1	1	15	10	0
4	1	0	0	1	0	1	0	5	10	0
5	0	1	0	1	1	1	1	10	5	0
6	1	0	0	1	0	1	1	5	5	0
7	1	0	0	1	0	0	0	0	5	5

Explanation of Simulation and Parameters:

The functioning of the datapath unit has been shown with an example by giving custom inputs that would be supplied by the control unit module to show GCD(25,15).

The steps were computed manually according to the main procedure state above.

1. Initially we load the X, Y values into the registers XR, YR due to which the registers get 25, 15 and both the XR>0 and XR>=R are enabled.
2. Now as we have XR = 25 and YR = 15 we need to subtract and load load that value into register XR again, so XR obtains the value XR(10) = XR(25) - YR(15) then accordingly ZEQ_Flag=1 while LEQ_Flag=0 as XR<YR.
3. Then we have to swap the values in the registers XR and YR as XR<YR. So XR = 15 and YR = 10 then accordingly ZEQ_Flag=1 while LEQ_Flag=1 as XR>=YR and XR>0.
4. Next cycle, subtraction is done as XR(5) = XR(15) - YR(10) so ZEQ_Flag=1 while LEQ_Flag=0 as 5<10.
5. After this clock cycle, swapping is done again so XR = 10 and YR = 5.
6. As per the GCD algorithm, subtraction is done XR(5) = XR(10) - YR(5) due to which ZEQ_Flag=1 while LEQ_Flag=1.
7. Finally again subtraction is done after which XR=0 and ZEQ_Flag = 1 due to which Z gets the value of YR.

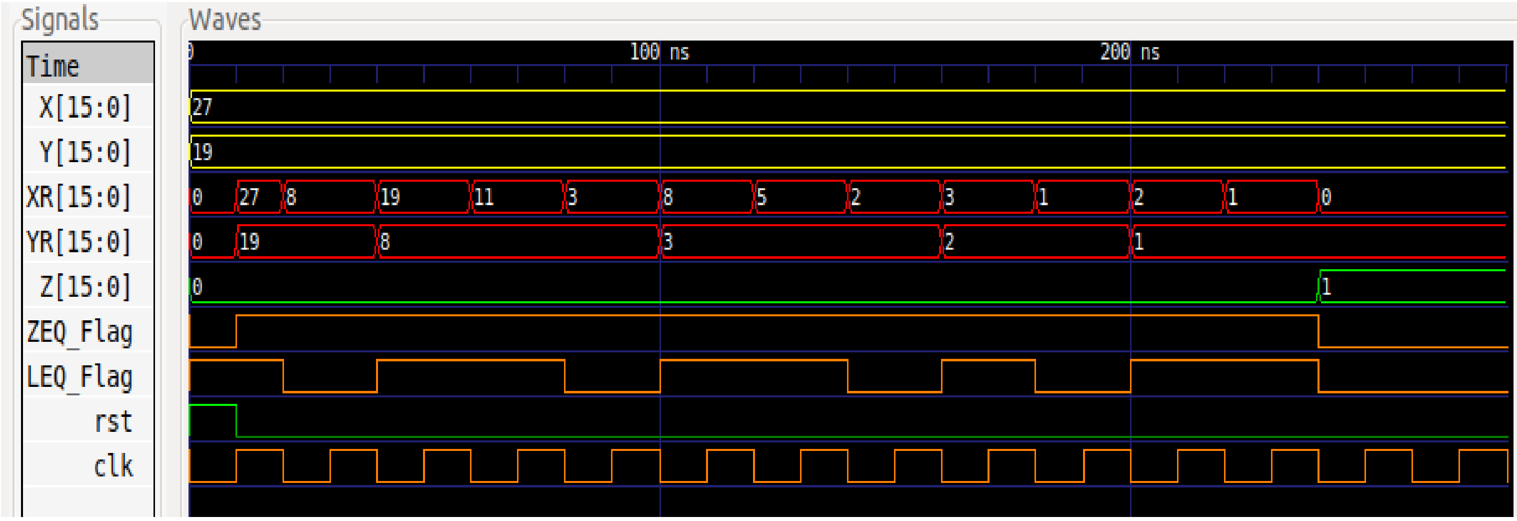
Conclusion:

The instructions sent from the testbench modifies the register values and finally gives the **GCD(25,15)** = **Z** = **5**. Therefore, we can conclude

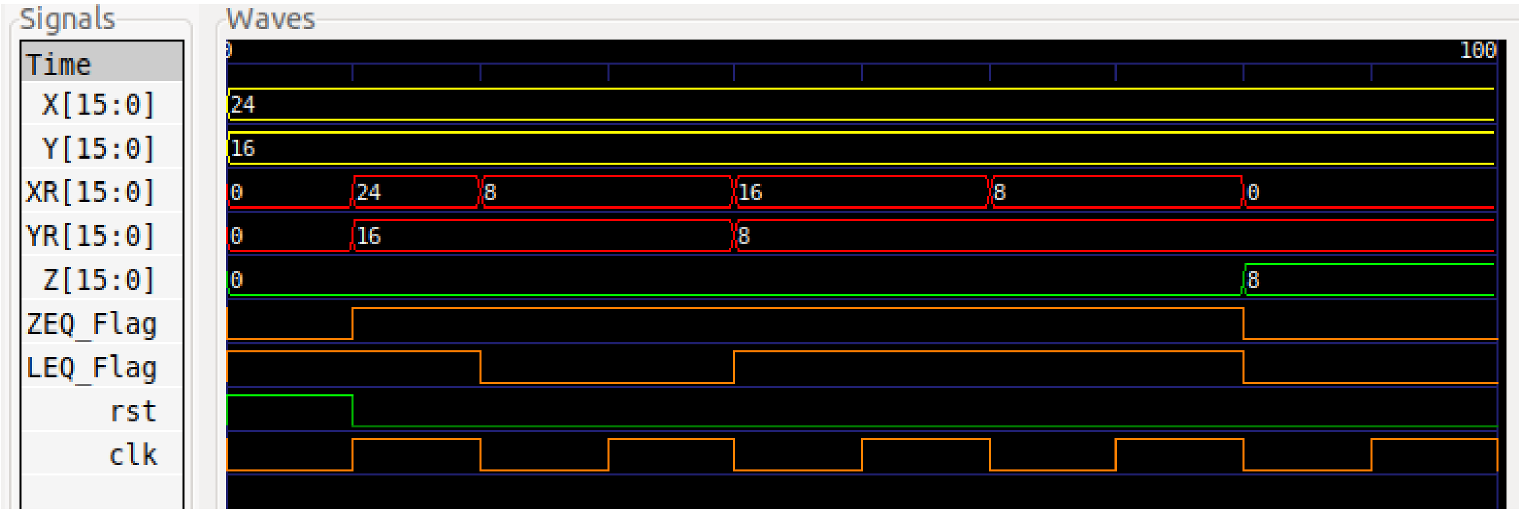
E)

NOTE: rst functionality has been shown in the first cycle.

(I) Computation of GCD(27,19)



(II) Computation of GCD(24,16)



Observation:

The GCD algorithm is executed till the state becomes S3 after which Z gets the GCD value(from YR) on the positive edge of the clock.

The GC D values obtained are:

(I) GCD(27,19) = 1;

(II) GCD(24,16) = 8;