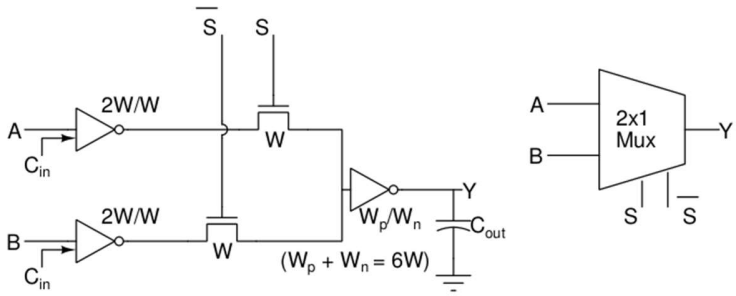
VLSI ASSIGNMENT-3 REPORT

Question-1



We need to implement the following circuit and report the delay. The constraint is to see that we get minimum delay such that Wp+Wn=6.

Netlist:

```
First question Suryasasanka_2019102036
.include TSMC_180nm.txt
.param SUPPLY=1.8
.param LAMBDA=0.09u
.param width=10*LAMBDA
.global gnd vdd

VDS vdd gnd 'SUPPLY'
vin_a A 0 pulse 0 1.8 0ns 100ps 100ps 4.9ns 10ns
vin_b B 0 pulse 0 1.8 0ns 100ps 100ps 9.9ns 20ns
ven s 0 pulse  1.8 0 0ns 100ps 100ps 49.9ns 100ns
vbs s_bar 0 pulse  0 1.8 0ns 100ps 100ps 49.9ns 100ns
.subckt invmos y x enable out1 vdd gnd
.param width_N=width
.param width_P=2*width

M1 y x 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 y x vdd vdd 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}

M3 y enable out1 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 invmos
                                                                    1,1
```

In the netlist, we initially define the parameters. Then we declare the input pulses A, B. The enable is S. So we define S and S_bar. We have written a subcircuit named invmos. The circuit is an inverter whose output is given as the drain to of an n-mos. The gate voltage of n-mos is given through ‘enable’.

```
.subckt invfinal y x vdd gnd
.param width_N=5*width
.param width_P=1*width

M1 y x 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 y x vdd vdd 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 invfinal


x1 dtop A s out1 vdd gnd invmos
x2 dbot B s_bar out1 vdd gnd invmos
x3 out out1 vdd gnd invfinal

Cin1 A 0 4.31f
Cin2 B 0 4.31f
Cout1 out1 0 8.62f
Cout out 0 8.62f

.tran 0.1n 100n

                                                                    31,0-1
```

We have written a subcircuit named invfinal. This is a normal inverter circuit but the only difference is we change the widths of n-mos and p-mos everytime we run to obtain the minimum delay.

We then combined the given subcircuits and made the final circuit. We have also added different capacitances where required so that we get a better estimate of delay. Note that to get even accurate values, we might need magic.

```
.measure tran TpdrA
+ TRIG v(A) VAL='SUPPLY/2' RISE=1
+ TARG v(out) VAL='SUPPLY/2' RISE=1

.measure tran TpdfA
+ TRIG v(A) VAL='SUPPLY/2' FALL=1
+ TARG v(out) VAL='SUPPLY/2' FALL=1

.measure tran tpdA param='(TpdrA+TpdfA)/2' goal=0

.measure tran TpdrB
+ TRIG v(B) VAL='SUPPLY/2' RISE=1
+ TARG v(out) VAL='SUPPLY/2' RISE=1

.measure tran TpdfB
+ TRIG v(B) VAL='SUPPLY/2' FALL=1
+ TARG v(out) VAL='SUPPLY/2' FALL=1

.measure tran tpdB param='(TpdrB+TpdfB)/2' goal=0

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

```
run
plot v(A)
plot v(B)
plot v(s)
plot v(s_bar)
set curplottitle= SuryaSasanka_2019102036
plot v(out)



.endc
```

We use the .measure function to find the Propagation delay. TpdrA is found by the thought process that we want to calculate the time elapsed between the rise of v(A) and the rise of v(out). Similarly, we do it for the second terminal of the mux B. Then we plot all the pulses.

We know that

$$C_{in} \propto C_g$$

$$C_g = WLC_{ox}$$

$$\Rightarrow C_{in} \propto WLC_{ox}$$

$$\Rightarrow C_{in} \propto (W_p + W_n)L$$

$$\Rightarrow \boxed{C_{in} = 3WLC_{ox}}$$

We know $C_{ox} = 8.85 \times 10^{-3}$

$L = 2 \times$ Lambda Lambda $= 0.09\,\mu$

We chose $W = 10 \times$ Lambda

$$\therefore C_{in} = 3 \times 10 \times 2 \times (0.09 \times 10^{-6})^2 \times 8.85 \times 10^{-3}$$

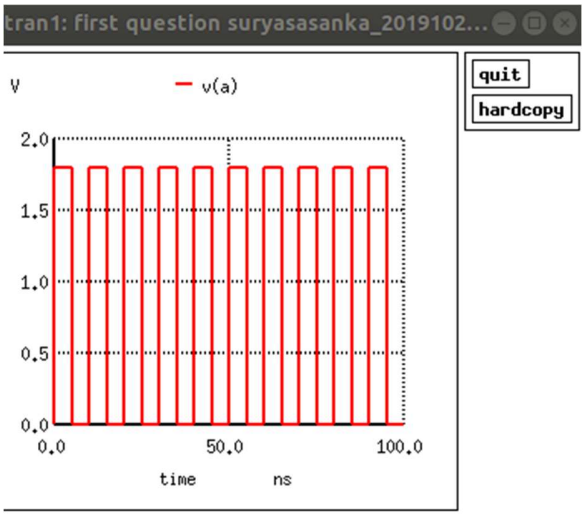
$$\boxed{C_{in} = 4.31\,fF}$$

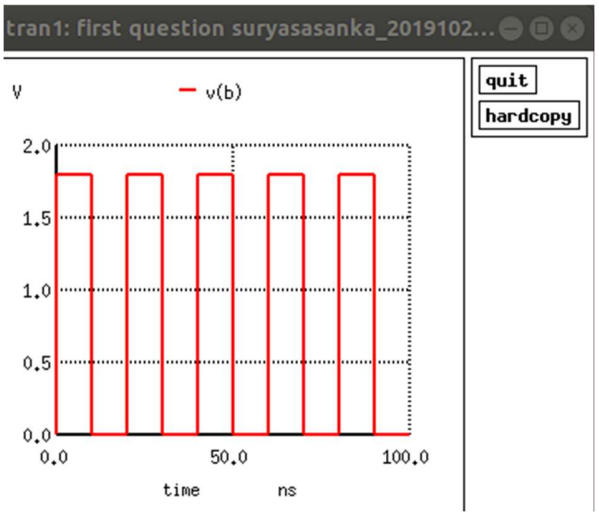
Given $\dfrac{C_{out}}{C_{in}} = 2$

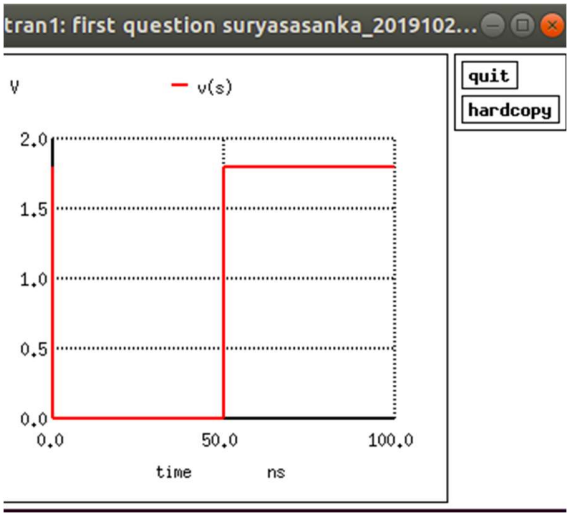
$$\Rightarrow \boxed{C_{out} = 8.62\,fF}$$

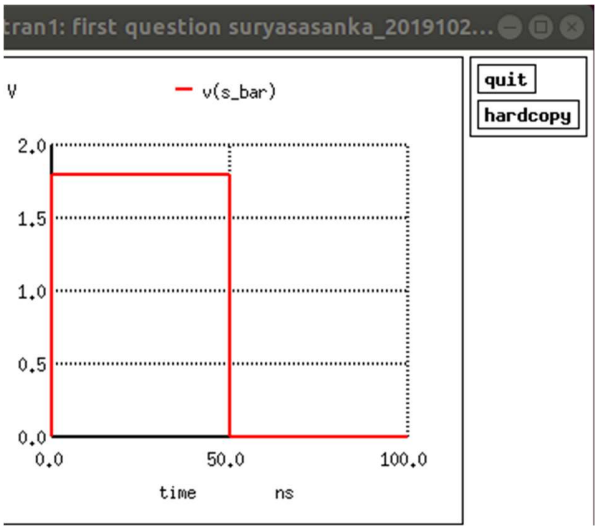
The values of capacitances have been added in the netlist at the required nodes

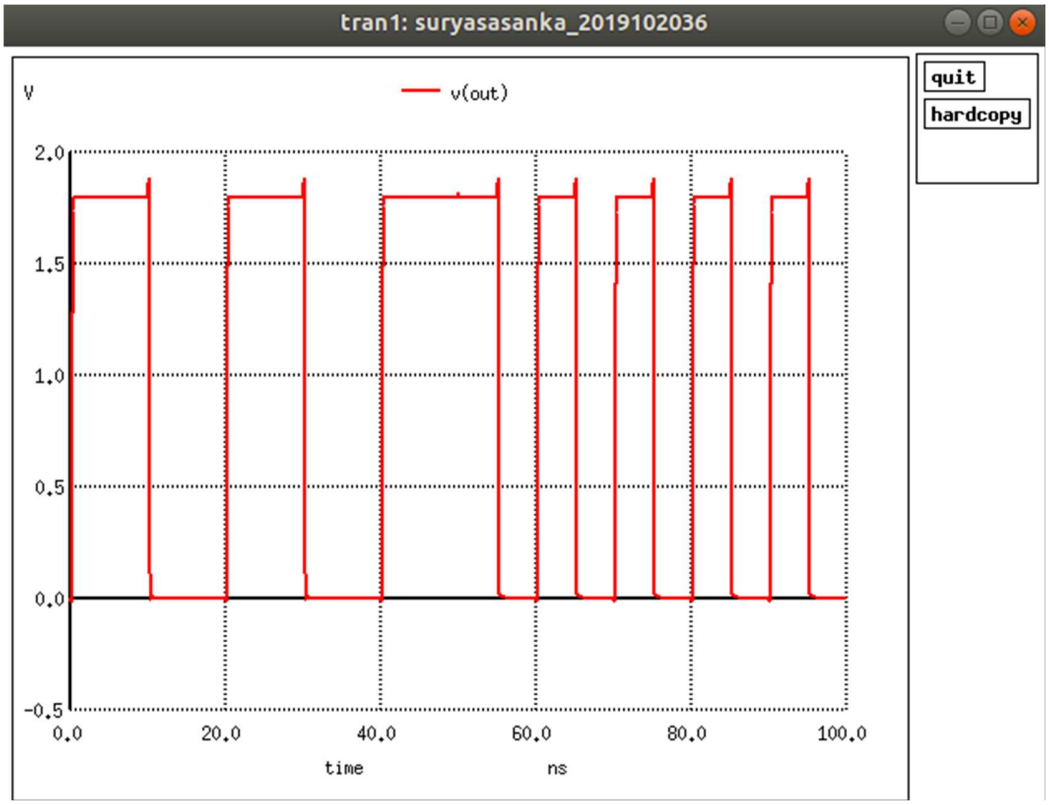
Plots:


tran1: first question suryasasanka_2019102...
V
v(a)
quit
hardcopy
2.0
1.5
1.0
0.5
0.0
0.0
50.0
100.0
time
ns


tran1: first question suryasasanka_2019102...
V
v(b)
quit
hardcopy
2.0
1.5
1.0
0.5
0.0
0.0
50.0
100.0
time
ns


tran1: first question suryasasanka_2019102...
V
v(s)
quit
hardcopy
2.0
1.5
1.0
0.5
0.0
0.0
50.0
100.0
time
ns


tran1: first question suryasasanka_2019102...
V
v(s_bar)
quit
hardcopy
2.0
1.5
1.0
0.5
0.0
0.0
50.0
100.0
time
ns



Explanations:

The output is as expected. S is 0 for the first 50ns. This means that B waveform must appear in the first 50ns.

At 50ns, S changes to 1. This means that A waveform must appear from 50ns to 100ns.

Delays:

Wp	Wn	Tpd(A)	Tpd(B)
1	5	2.679ns	0.179ns
2	4	2.666ns	0.166ns
3	3	2.671ns	0.171ns
4	2	2.697ns	0.197ns
5	1	2.801ns	0.301ns

Since we have noticed that Wp=2W, Wn=4W has least delay, we observe near that region. Wp=2.25W and Wn=3.75W has least delay. So in this interval

Wp=2-2.5W Wn=3.75-4W, we find the least delay.

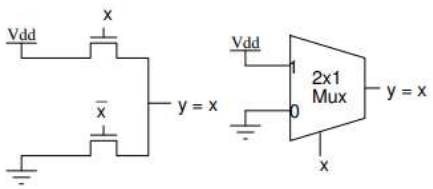
```
 Measurements for Transient Analysis

tpdra                =  1.499801e-10 targ=  1.999801e-10 trig=  5.000000e-11
tpdfa                =  5.182338e-09 targ=  1.023234e-08 trig=  5.050000e-09
tpda                 =  2.66616e-09
tpdrb                =  1.499801e-10 targ=  1.999801e-10 trig=  5.000000e-11
tpdfb                =  1.823375e-10 targ=  1.023234e-08 trig=  1.005000e-08
tpdb                 =  1.66159e-10
```

Figure depicts least delay observed

Question-2

We have to implement f = x1x2 + x1x3 + x1x4 + x2x3 + x2x4 + x3x4 using 2X1 muxes shown below.

Given, input impedance is 4 times the minimum size inverter. Therefore the capacitance value is 4*4.31=17.24fF



2 a) Shannon's expansion

$$f(x_1, x_2, \cdots x_n) = x_1 f(1, x_2, \cdots x_n) + \bar{x}_1 f(0, x_2, \cdots x_n)$$

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

$$= \bar{x}_1(x_2x_3 + x_2x_4 + x_3x_4) + x_1(x_2 + x_3 + x_4 + x_2x_3 + x_2x_4 + x_3$$

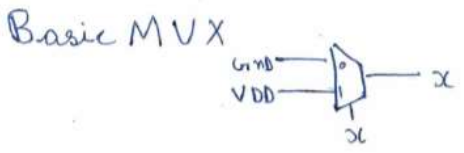
$$= \bar{x}_1(\bar{x}_2(x_3x_4) + x_2(x_3 + x_4 + x_3x_4))$$

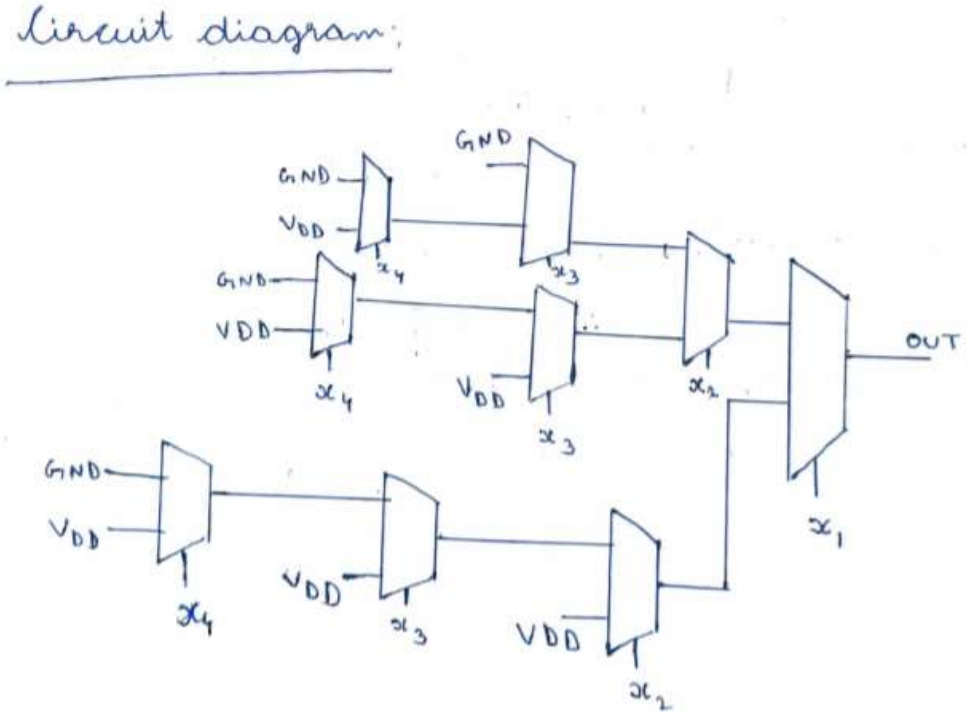
$$+ x_1(\bar{x}_2(x_3 + x_4 + x_3x_4) + x_2(1))$$

$$\boxed{f(x_1, x_2, x_3, x_4) = \bar{x}_1(\bar{x}_2(\bar{x}_3(0) + x_3(x_4)) + x_2(x_3(1) + \bar{x}_3(x_4)) + x_1(\bar{x}_2(\bar{x}_3(x_4) + x_3(1)) + x_2(1))}$$

Now, for this Shannon expansion, we need to draw the circuit diagram.

Important Note/convention: All the muxes used below are written such that select line 0 gives the output as the top input , and select line 1 gives the output as the bottom input. If A is the top input and B is the bottom input, then Ouput is Ax_bar+Bx. This is reverse to the general muxes we take.





This circuit is a practical realization of the mentioned equation because if A(I0), B(I1) are given as inputs to the mux with the select line as x, then the output is Ax_bar+Bx. So we have built the circuit for the Shannon’s expansion of the given equation.

Part B)

Ngspice netlist:

```
.include TSMC_180nm.txt
.param SUPPLY=1.8
.param LAMBDA=0.09u
.param width=10*LAMBDA
.global gnd vdd

VDS vdd gnd 'SUPPLY'
vin_a A 0 pulse 0 1.8 0ns 100ps 100ps 4.9ns 10ns
vin_b B 0 pulse 0 1.8 0ns 100ps 100ps 9.9ns 20ns
vin_c C 0 pulse 0 1.8 0ns 100ps 100ps 19.9ns 40ns
vin_d D 0 pulse 0 1.8 0ns 100ps 100ps 39.9ns 80ns
vin_abar A_bar 0 pulse 1.8 0 0ns 100ps 100ps 4.9ns 10ns
vin_bbar B_bar 0 pulse 1.8 0 0ns 100ps 100ps 9.9ns 20ns
vin_cbar C_bar 0 pulse 1.8 0 0ns 100ps 100ps 19.9ns 40ns
vin_dbar D_bar 0 pulse 1.8 0 0ns 100ps 100ps 39.9ns 80ns


.subckt mux a b enable enable_bar out vdd gnd
.param width_N=width


M1 a enable_bar out 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 b enable out 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 mux
```

We have given X1, X2, X3, X4 as four pulse A, B, C, D of timeperiod 10ns, 20ns, 40ns, 80ns respectively. We have then defined their inverted pulses A_bar, B_bar, C_bar, D_bar.

Then, we have written a subcircuit which is a realization of mux. If enable is 0, a is the output. If enable is 1, then b is the output.

```
x1 gnd vdd D D_bar four vdd gnd mux
x2 four vdd C C_bar stageb1 vdd gnd mux
x3 stageb1 vdd B B_bar stageb2 vdd gnd mux

x4 gnd vdd D D_bar four2 vdd gnd mux
x5 gnd four2 C C_bar stagea1 vdd gnd mux

x6 gnd vdd D D_bar four3 vdd gnd mux
x7 four3 vdd C C_bar stagea2 vdd gnd mux
x8 stagea1 stagea2  B B_bar stagea3 vdd gnd mux

x9 stagea3 stageb2 A A_bar out vdd gnd mux
```

In this part, we have used the mux subcircuit carefully by looking at the final result from the Shannon's expansion. We have connected the muxes properly so that we get x1x2+x1x3+x1x4+x2x3+x2x4+x3x4 at the output.

```
.tran 0.1n 200n

.measure tran tplh
+TRIG v(out) val = '0' RISE=1
+TARG v(out) val = '1' RISE=1

.measure tran tphl
+TRIG v(out) val = '1' FALL=1
+TARG v(out) val = '0' FALL=1


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


run
plot v(A)
plot v(B)
plot v(C)
plot v(D)
set curplottitle= SuryaSasanka_2019102036
plot (out)
.endc
```

We are calculating the value of tplh and tphl.

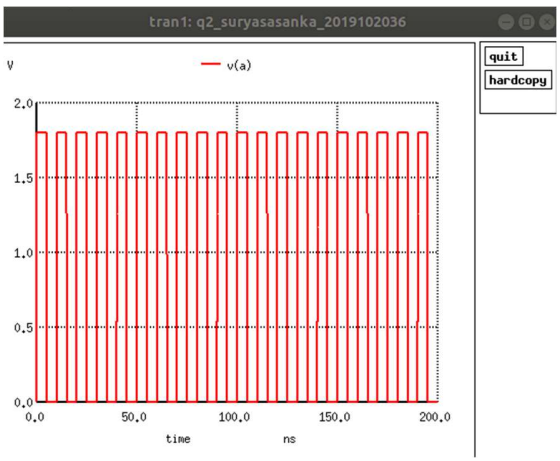
Tplh- It is the time taken for the value at a particular node to rise from 0 to 1

Tplh- It is the time taken for the value at a particular node to fall from 1 to 0

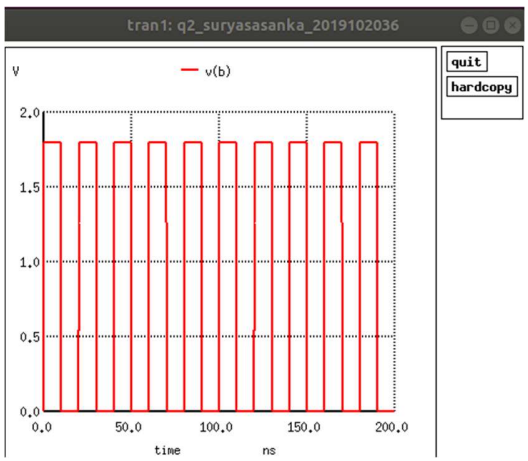
Note that we have been asked to take 1 because for the output to reach to 1.8, the pulse should be very long because though initially we get a high value quickly, from 1, the capacitor charges extremely slowly to 1.8. So for practical purposes we notice that ~1.2 is the maximum value that is possible in reasonable time and thus take 1 as reference for high.

We have then plotted all our plots.

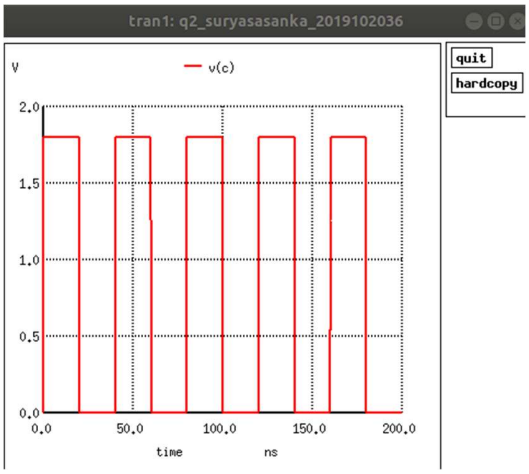
Plots:



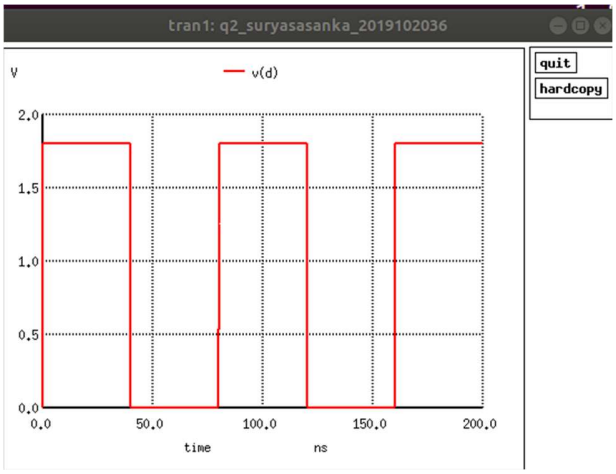
Time period-10ns



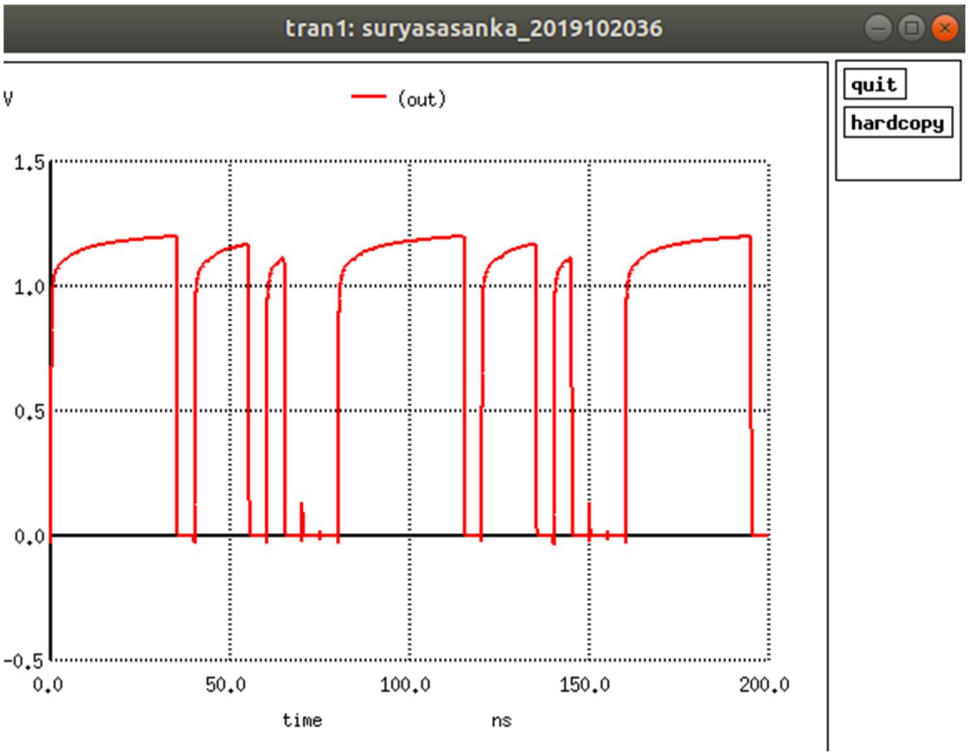
Time period-20ns

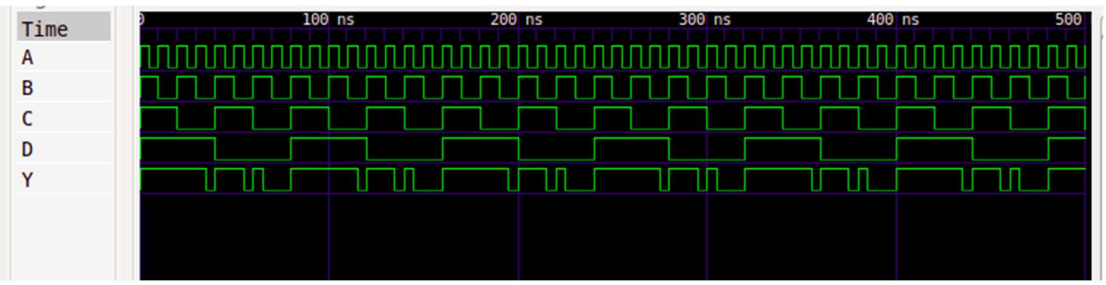


Time period-40ns



Time period-80ns





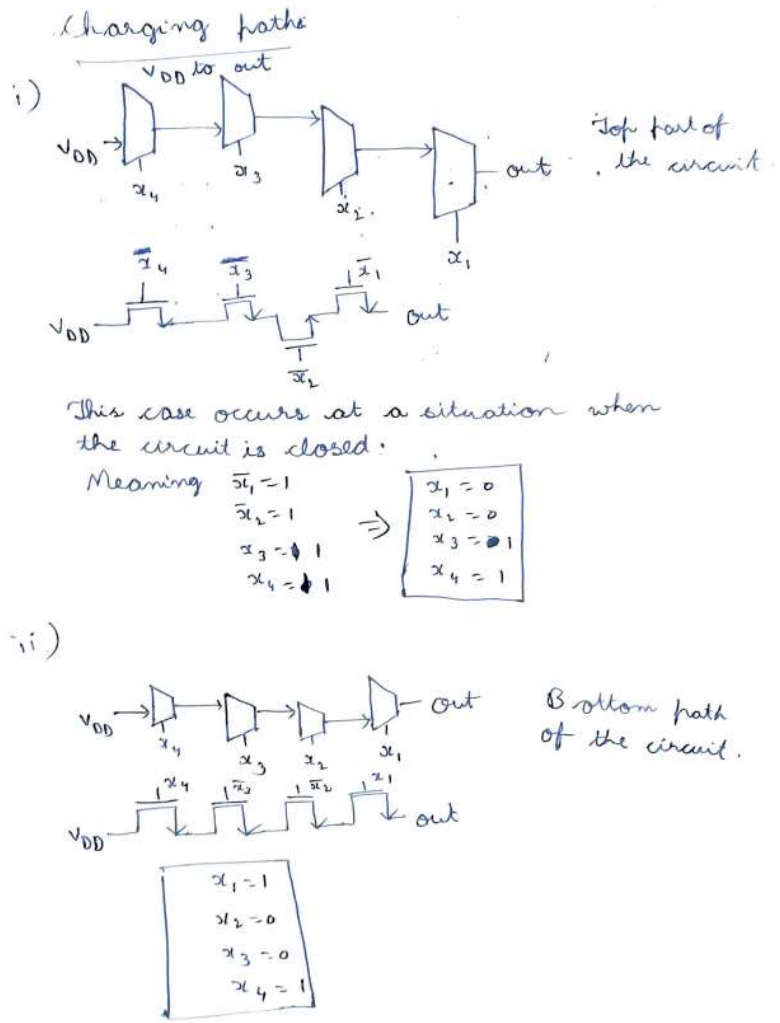
Expected output

Comments:

We have noticed that the actual output is a replica of the expected output. This ensures that our logic is successfully implemented by our netlists. However, in the output, we do not get exact pulses as we have a capacitor of capacitance Cout=17.24fF(4*minimum sized inverter capacitor) at the output. Since the capacitor takes time to charge, we do not exactly get rectangles.

Part C)

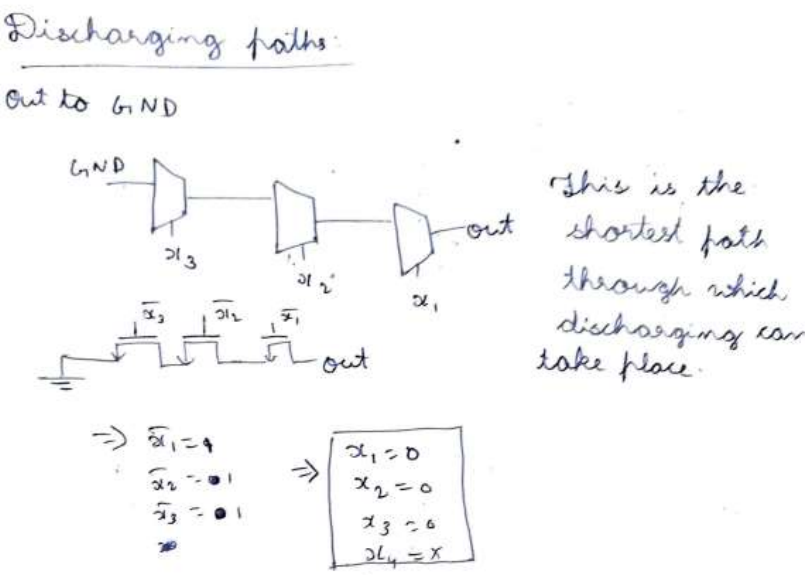
Charging path -Vdd to out



At t=15ns i) is realised

At t=30ns ii) is realised

Discharging path- Out to Gnd



At t=75ns this discharging path is realised

```
  Measurements for Transient Analysis

tplh                 =  5.688335e-10 targ=  6.166702e-10 trig=  4.783669e-11
tphl                 =  4.427068e-10 targ=  3.551818e-08 trig=  3.507548e-08
```

These are the minimum transition times for the output to go from high to low and vice versa.

We should first justify that this is the minimum Tplh and Tphl. We notice that the charging paths take more or less the same time because all charging paths we require the same number of transistors ie 4. So Tplh value is same for all the paths and the value is shown in the figure above.

However discharging from out to Gnd occurs with 3 transistors sometimes and 4 transistors. Since we want minimum delay, we take the case of discharging through 3 transistors and calculated that value to find the minimum delay which is the Tphl shown in the figure above.

Part D)

We have seen that tplh=5.6883*10^-10 secs

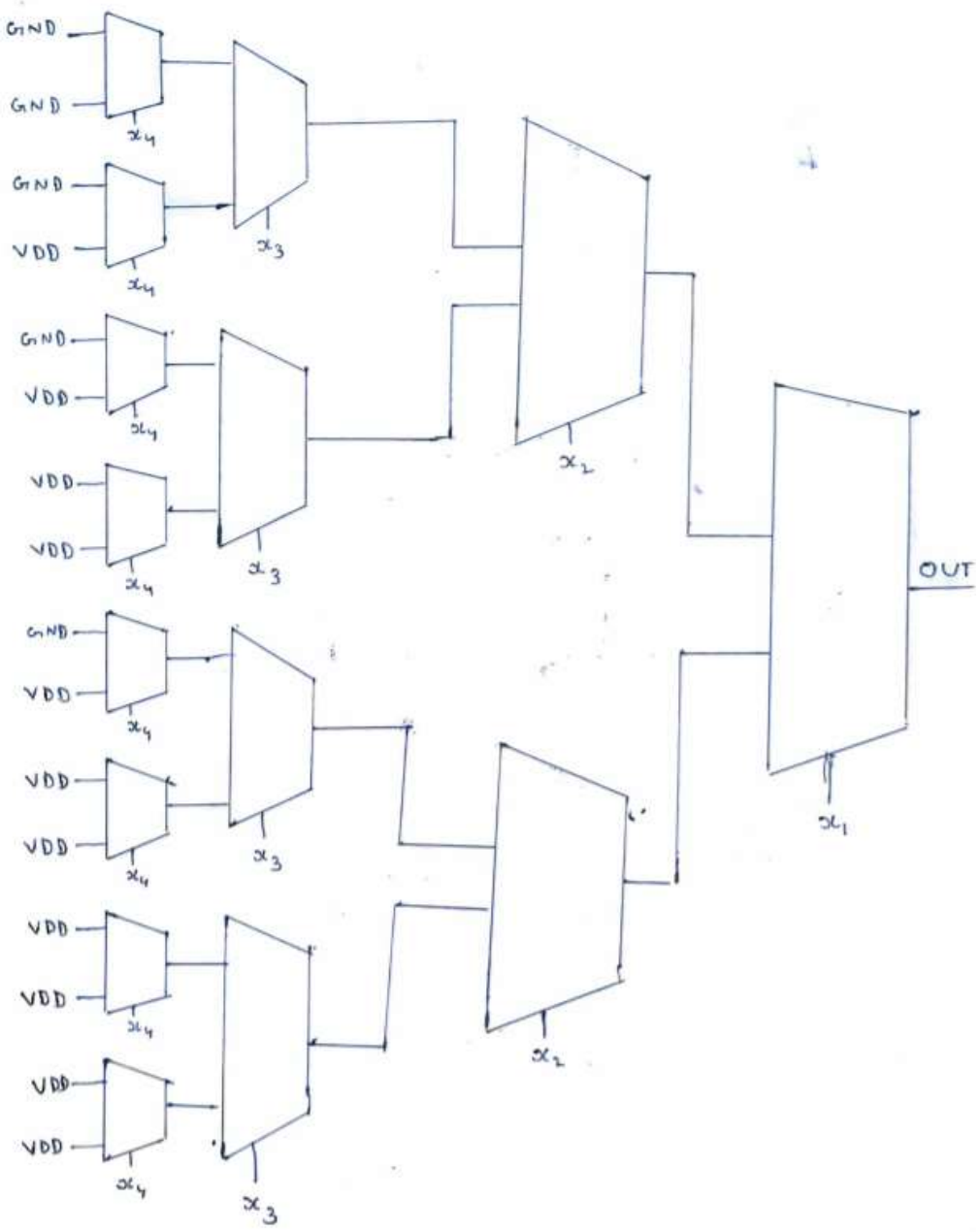
tphl=4.427*10^-10 secs.

There is a difference of 1.2*10^-10 which is actually a very small value. The value of Tplh >Tphl because discharging happens through 3 transistors(least) and charging happens through 4 transistors meaning that more time elapses for tplh .

We need to make this number equal. One possible way is to add inverters to see that the equal number of transistors are always present in charging path and discharing paths. However here, the better option will be adding muxes.

We simply increase the number of paths such that in any path we find equal number of transistors. This is done by taking a mux with a required select line and then taking both the inputs to the mux as GND or VDD.

Optimised circuit:



For this circuit, we notice that the charging paths and discharging paths all involve 4 transistors. And thus Tplh will be nearly equal to Tphl

Question-3

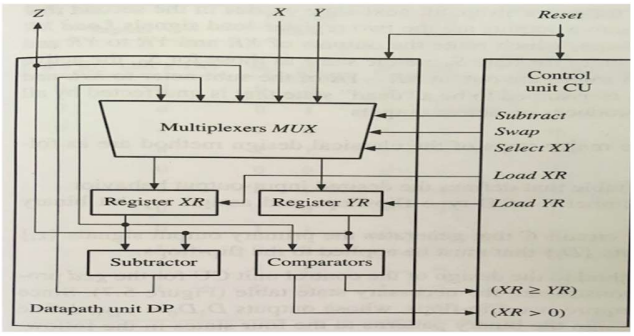
In this question we are required to calculate the GCD of two numbers using a variant of GCD algorithm given in the question.

Euclidean algorithm:

```
gcd(in: X,Y; out: Z);
  register XR, YR, TEMPR;
  XR := X;                          {Input the data}
  YR := Y;
  while XR > 0 do begin
    if XR ≤ 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;
```

- We have X and Y for which we need to calculate GCD.
- We load these values into registers Xr and Yr.
- We know that to calculate GCD, division is to be done. But division is subtraction done repeatedly. Therefore, while a particular number Xr>0, we take Xr and Yr. We compare whether Xr>=Yr. If the condition is false, then we swap the numbers to ensure that Xr has higher value and Yr has lower value.
- We then do the operation Xr=Xr-Yr. This process of swapping(if necessary) and subtracting(done everytime) keeps on repeating until the value of Xr>0. If that condition is false, the value in Yr is copied into Zr. Zr is the required GCD.

For this we add certain changes so that this process is done through modules which have a controller unit and data path.



Part-A

Q 3) (a) Let S_0 – begin

S_1 – swap

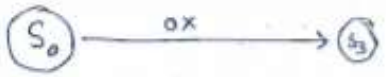
S_2 – subtract

S_3 – End.

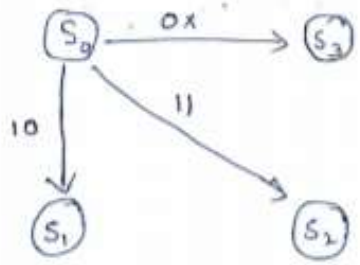
Conditions $(XR > 0)$ $(XR \geq YR)$

0 or 1 0 or 1

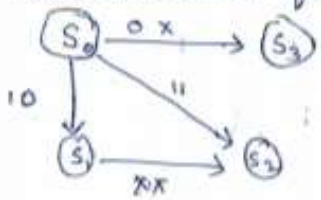
At S_0, if we encounter 0, we don't care what the second bit is, we directly end.



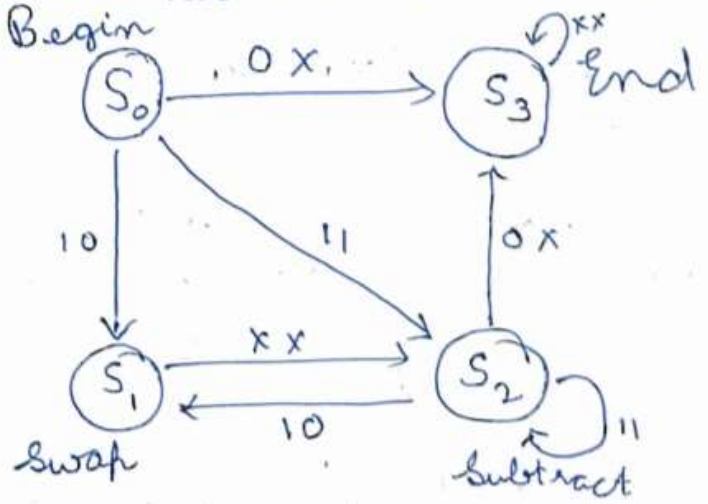
At S_0, if we encounter 1, we move to swap i.e S_1 when 10 and we can directly subtract S_2 when 11 is encountered.



At S_1, i.e swap no matter what your values are subtraction follows next.



At subtractor stage, we get many possibilities.
If 0 is encountered in first bit, we move to ends,
If 10 is encountered, we return again to swaps,
If 11 is encountered, subtraction must be repeated
Note that at end no matter what ~~state is~~ bit we give it remains there only.



Part-B

We already have the given state table

	Inputs ($XR > 0$) ($XR \geq YR$)			Outputs				
State	0–	10	11	*Subtract*	*Swap*	*Select XY*	*Load XR*	*Load YR*
S_0 (Begin)	S_3	S_1	S_2	0	0	1	1	1
S_1 (Swap)	S_2	S_2	S_2	0	1	0	1	1
S_2 (Subtract)	S_3	S_1	S_2	1	0	0	1	0
S_3 (End)	S_3	S_3	S_3	0	0	0	0	0

Now. We change this state table into an excitation table where for 0X, 10, 11 as inputs and depending on the current state, the next state is decided.

We then take S0=00, S1=01, S2=10, S3=11 to obtain the below excitation table where the next state values of these two bits(called D1, D0) are decided.

(b) We can write the given truth table in the form of the following Excitation table

$XR>0$	$XR \geq YR$	D_1	D_0	D_1^+	D_0^+	subtract	swap	select	XR	YR
0	x	0	0	1	1	0	0	1	1	1
0	x	0	1	1	0	0	1	0	1	1
0	x	1	0	1	1	1	0	0	1	0
0	x	1	1	1	1	0	0	0	0	0
1	0	0	0	0	1	0	0	1	1	1
1	0	0	1	1	0	0	1	0	1	1
1	0	1	0	0	1	1	0	0	1	0
1	0	1	1	1	1	0	0	0	0	0
1	1	0	0	1	0	0	0	1	1	1
1	1	0	1	1	0	0	1	0	1	1
1	1	1	0	1	0	1	0	0	1	0
1	1	1	1	1	1	0	0	0	0	0

We are using 2 flipflops over here because we have 4 states . Xr>0 and XR>=Yr are important quantities because they help us to decide the next stage into which we go.

We try and calculate the next state values i.e D1(t+1) and D0(t+1) using Xr>0, Xr>=Yr, D1(t) and D0(t).

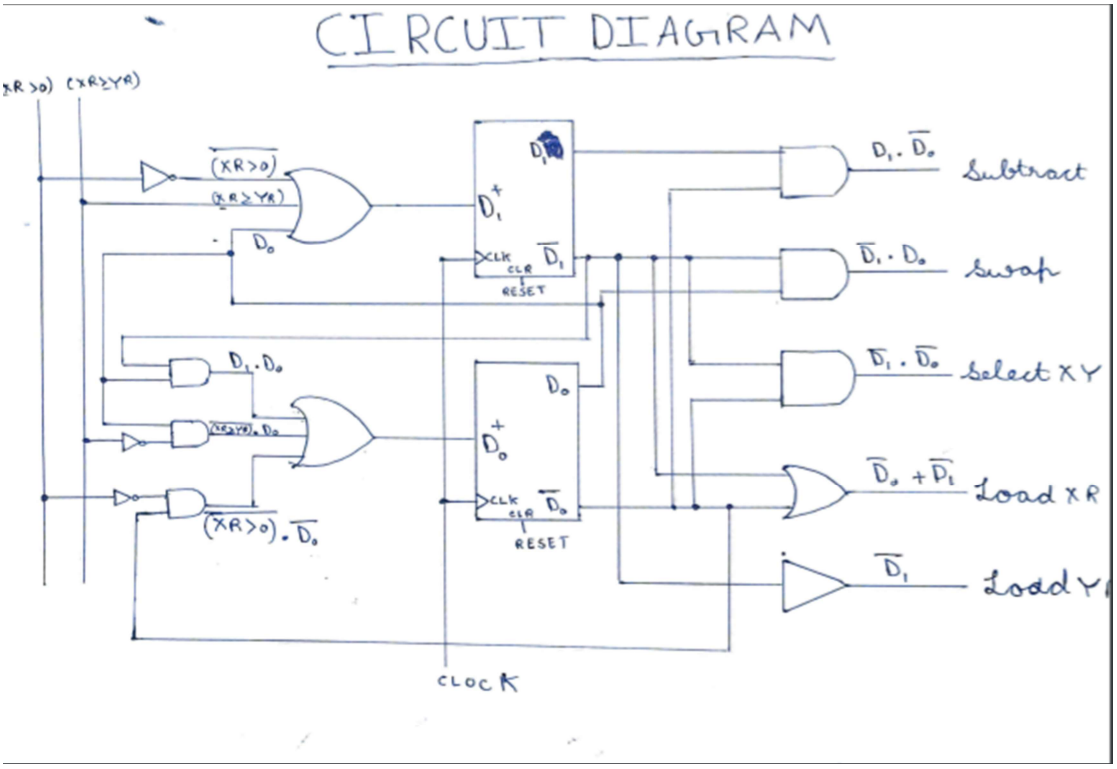
This is achieved through **K-maps**

Also the bits subtract, swap, select xy, load xr, load yr will be expressed in terms of D1(t) and D0(t). Calculating the values we get the expressions

$$\text{subtract} = D_1 . \overline{D_0}$$
$$\text{swap} = \overline{D_1} . D_0$$
$$\text{select } XY = \overline{D_1} . \overline{D_0}$$
$$\text{Load } XR = \overline{D_0} + \overline{D_1}$$
$$\text{Load } YR = \overline{D_1}$$

The next state equations are as follows

$$D_1^+ = \overline{(XR>0)} + (XR \geq YR) + D_0$$
$$D_0^+ = D_1 D_0 + \overline{(XR \geq YR)} . \overline{D_0} + \overline{(XR>0)} . D_0$$



Part-C, D(combined for better flow)

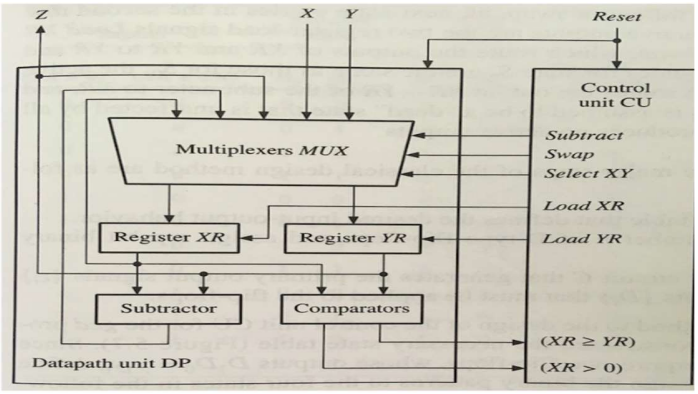
Verilog module for control unit:

```
surya@surya-HP-Laptop-15-da0xxx: ~/sem4/VLSIAssignment3
File  Edit  View  Search  Terminal  Help
module controlunit(endflag, swapflag, reset, clk, controlarr, a, b, flag);

input endflag, swapflag, reset, clk;
output reg a, b, flag;
output reg [4:0] controlarr;
reg temp;

initial begin
a=0;  // D-flipflop value D1
b=0;  // D-flipflop value D0
flag=1;

controlarr=5'b11100;
end

always @ (posedge clk) // Start at positive edge of the clock
begin
        $display("Entered cu a=%b b=%b controlarr=%b endflag=%b swapflag=%b\n", a, b, controlarr, endflag, swapflag);

        temp=b;
        b=(a*b)|(b*(!endflag))|((!b)*(!swapflag)); //D0 next= D1D0 + (!(Xr>=Yr)).(!D0) + (! Xr>=0).D0
        a = !(endflag* (!swapflag)*(!temp));       //D1 next= (! Xr>=0) + (Xr>=Yr) + D0

        controlarr[0]=a*(!b);// Setting up the subtractor
        controlarr[1]=b*(!a); // Setting up the swap
        controlarr[2]=(!a)*(!b); // Setting up load bit
        controlarr[3]=!(a*b); // Setting up load bit for X
        controlarr[4]=(!a); // Setting up load bit for Y

        $display("Completed cu Next stage: a=%b b=%b controlarr=%b \n", a, b, controlarr);
        flag=!flag;
                                                                                        17,0-1
```

- We initially defined the input, outputs and then used ’controlarr’ to store the five bits necessary for us in the datapath.
- ‘Controlarr’ stores the five important bits which help in controlling the flow of data. These values help us in making important decisions about the operation that needs to be done.
- Endflag tells us about (Xr>0) and swapflag tells us about (Xr>=Yr).
- We have also calculated the value of the next state which is implemented by flipflops. We simply write the expressions for the next states along with the subtract swap select xy, load xr, load yr.
- Necessary comments have been added in the module for easy understanding.

Data path:

The data path is the flow in which operations are performed on our data so that we end up with the GCD.



- We initially select XY and Load the values of the numbers for which we have to calculate the GCD. This loading of numbers is done from the Registers Xr and Yr.
- We send the values into the comparator block where two important operations i.e. (Xr>=Yr) and (Xr>0) are calculated.
- We send the number into the subtractor block. But before that we send the Xr and Yr into the mux so that swapping condition is checked(This information is obtained from control unit based on Xr>=Yr).
- The outputs are again sent back into the muxes if the value of Xr>0 bit is in the control block obtained from the comparator is 1.
- Otherwise, the value in Zr (which is gets updated as the value of Yr after every clock cycle) is the required GCD.

Verilog module for data path:

```
surya@surya-HP-Laptop-15-da0xxx: ~/sem4/VLSIAssignment3
File  Edit  View  Search  Terminal  Help
module datapath(x, y, controlarr, xr, yr, zr, reset, clk, flag, a, b );
input [4:0] controlarr, x, y;
input reset, clk, flag, a, b;
output reg [4: 0] xr, yr, zr;

reg[4:0] tempr;

always@(flag)// We used this flag to ensure the sequential behaviour and combining
             // the control unit and the data path properly
begin
        $display("Entered Datapath : controlarr=%b\n", controlarr);

        if(a==1 && b==1)// End condition
        begin
        $display("GCD=%d", zr); // This zr contains GCD
        $finish;
        end

        if(controlarr[2]==1)// Select XY
        begin
                if(controlarr[3]==1) //Load Xr
                xr=x;
                if(controlarr[4]==1)// Load Yr
                yr=y;
        end

        if(controlarr[1]==1)// Swap condition
        begin
                tempr=yr;
                yr=xr;
                xr=tempr;
        end

        if(controlarr[0]==1) // Subtract condition
        xr=xr-yr;

zr=yr;
```

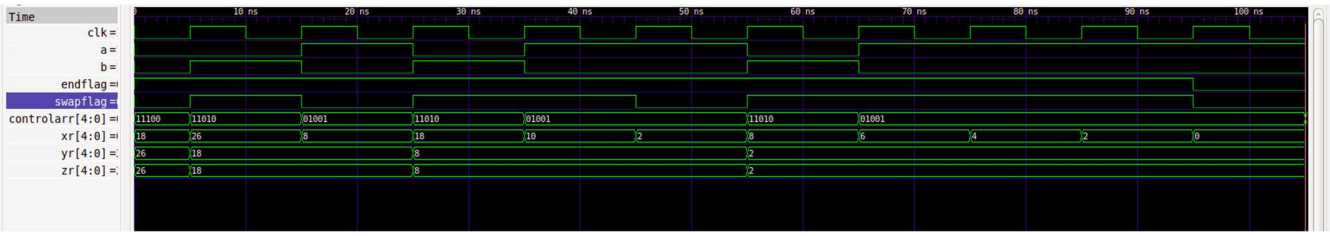
- After we initially defined the inputs and outputs, we simply analyse the bits in ‘controlarr’.
- If the value of the second bit is 1, then we load the values of the numbers x, y into registers Xr and Yr.
- Depending on whether the swap bit of controlarr is 0 or 1, we swap the data. If the subtract bit of controlarr is 1, then we subtract xr and yr and store it in xr.
- The value of Zr is updated as the Yr value. This process repeats as long as a!=1 and b!=1.
- As soon as a=1 and b=1(Indiciates S3-End) condition is seen, we display the value of Zr (which is the required gcd) and finish the process.

Verilog module for GCD(combining control unit and datapath):

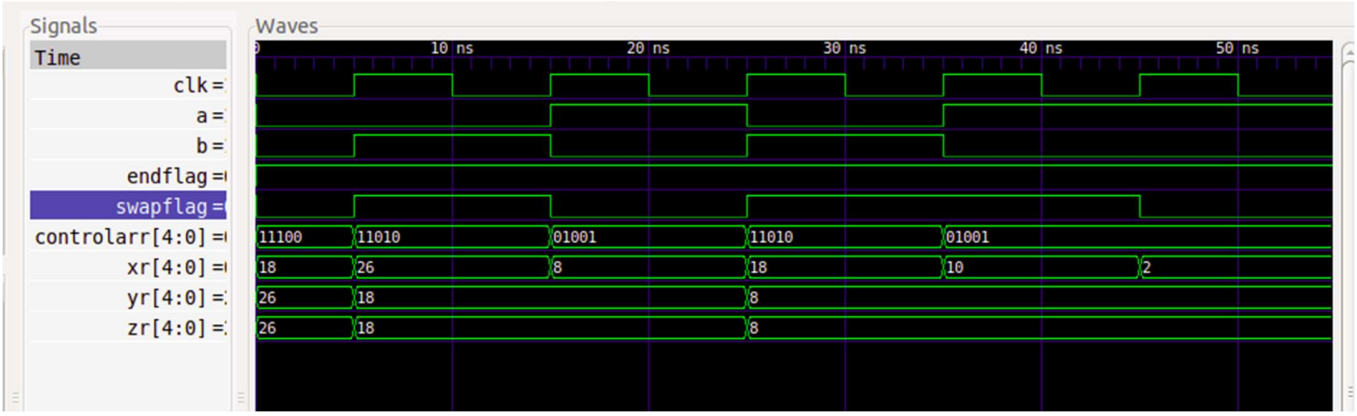
```
surya@surya-HP-Laptop-15-da0xxx: ~/sem4/VLSIAssignment3
File  Edit  View  Search  Terminal  Help
`include "controlunit.v"
`include "datapath.v"

module gcd ();

reg [4:0] x,y;
wire [4:0] controlarr,zr,xr,yr;
reg clk,reset;
wire a,b,flag;


initial begin
        $dumpfile("gcd.vcd");
        $dumpvars(0, gcd);

        clk=0;
        x=18;y=26;
end

always #5 clk=~clk;

datapath temp (x,y, controlarr, xr,yr,zr,reset,clk,flag,a,b);
controlunit dummy (xr>0, xr>=yr,reset,clk, controlarr,a,b,flag);


endmodule
```

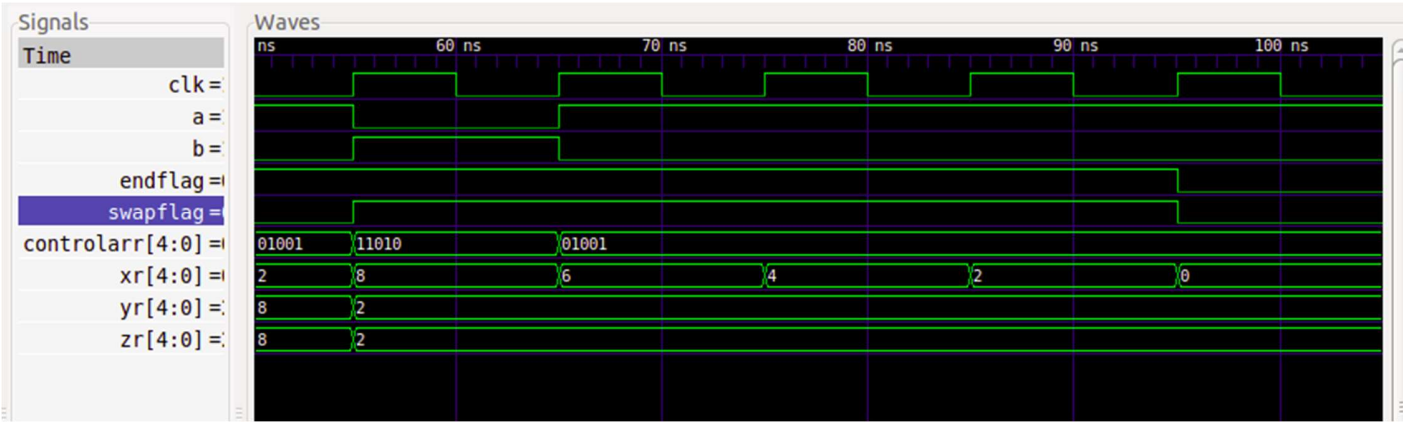
- In this module we include the controlunit and datapath modules after which we defined the required variables. We dumped the values into a .vcd file to obtain the GTKwave outputs.
- We defined the clock.
- The x and y are the values for which we need to calculate GCD. You can change them while running the module for obtaining GCD of different numbers
- We have then 'called' the datapath and controlunit modules.
- This module is just to combine the control unit and data path.

We have been asked to show simulations and working of the control unit and the datapath in parts C) and D) of the question. We will do it using the example GCD(18, 26).



Zoomed images:





Here,

clk- clock	Controlarr[4:0]----
a-D1	Controlarr[4](leftmost)-load yr
b-D0	Controlarr[3] -load xr
endflag (Xr>0)	Controlarr[2] -select xy
swapflag(Xr>=Yr)	Controlarr[1] -swap
Xr, Yr- registers	Controlarr[0](rightmost)-subtract

Explaining the functionality of the control unit:

For observing the functionality of the control unit, we must notice the values of clock, a, b, endflag(Xr>0), swapflag(Xr>=Yr), controlarr.

Notice that at the start, both a=0, b=0, controlarr =11100 (read bits from right to left) , (Xr>0)=1 and (Xr>=Yr)=0.

We notice that the next value a=0 b=1 conrolarr=11010. This value is in accordance with our excitation table written above. Actually what happened in the control unit is that we are at the begin stage where we have encountered 10. The state now moves to swap and the controlarr bits are updated correspondingly. Next subtraction takes place after swap. Then based on the value of endflag and swapflag, we move to another state, everytime calculate the future value of a, b and then calculate the controlarr bits which tell us the expected operation.

We must also notice that at 95 ns, the value of Xr>0 became 0. This means that we need to END the process after the completion of the clock cycle.

Explaining the functionality of datapath:

For observing the functionality of the datapath, we must notice the values of clock, controlarr, xr, yr, zr keeping an eye on the value of controlunit values.

We have 18, 26. They are loaded as Xr=18, Yr=26.

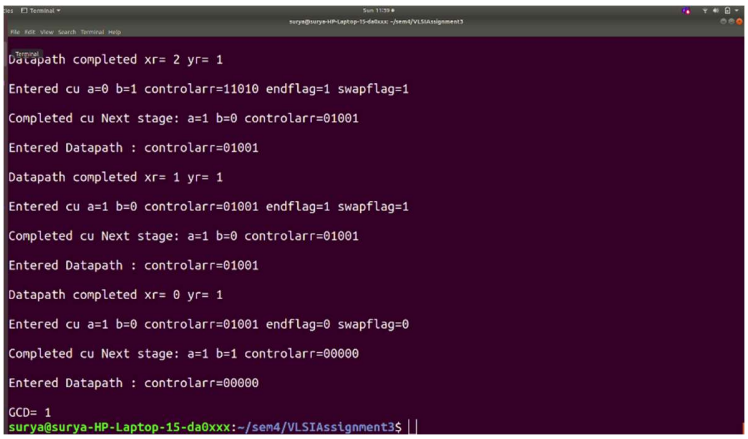
Since the swap bit in controlarr is 1, we see that the values are swapped to get Xr=26, Yr=18.

Now we notice that the subtractor bit is 1 in controlarr. So Xr=8 Yr=18. Zr is updated as 18

This swapping and subtracting process continues until we get in the final stage we got Xr=0 and Yr=2. Since Xr>0 is violated, we get Zr=Yr=2 as the GCD which is the right answer.

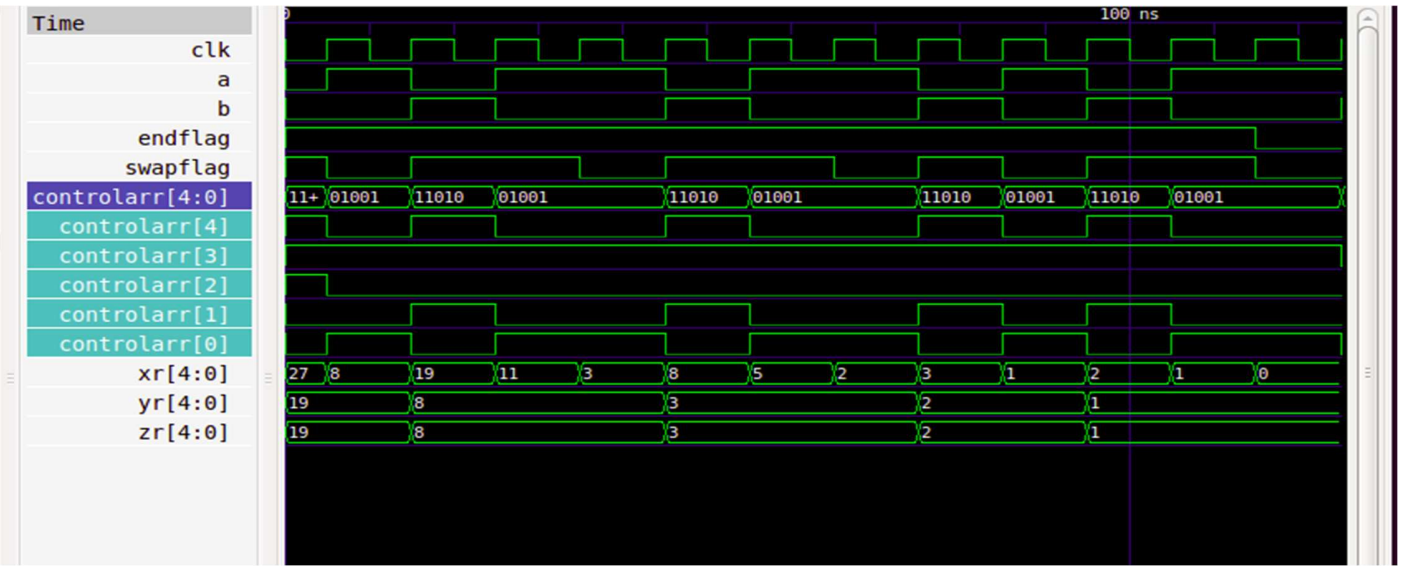
Part E)

GCD(27, 19)



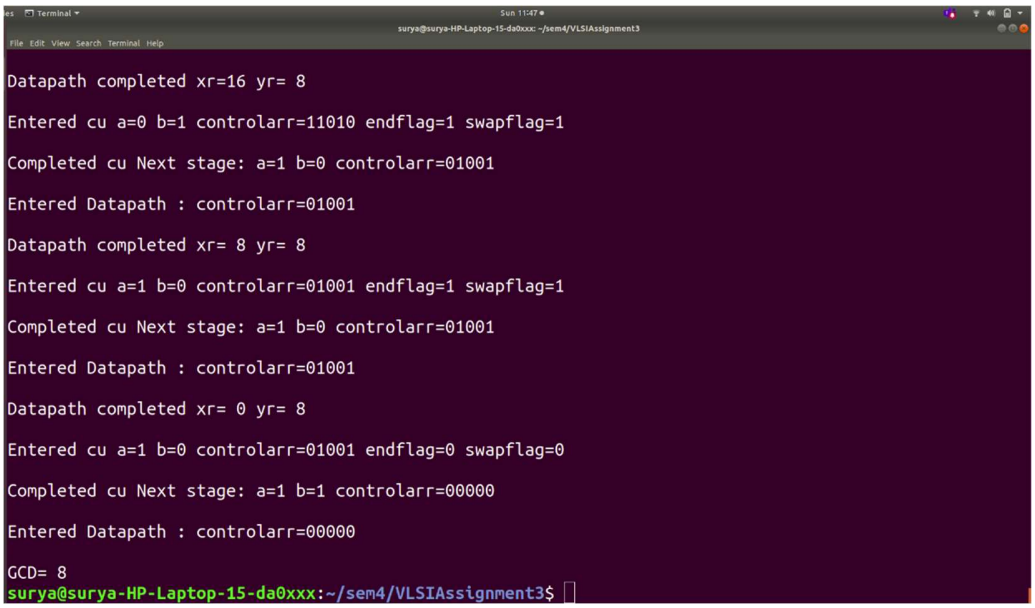
GCD=1.

GTKwave plot:



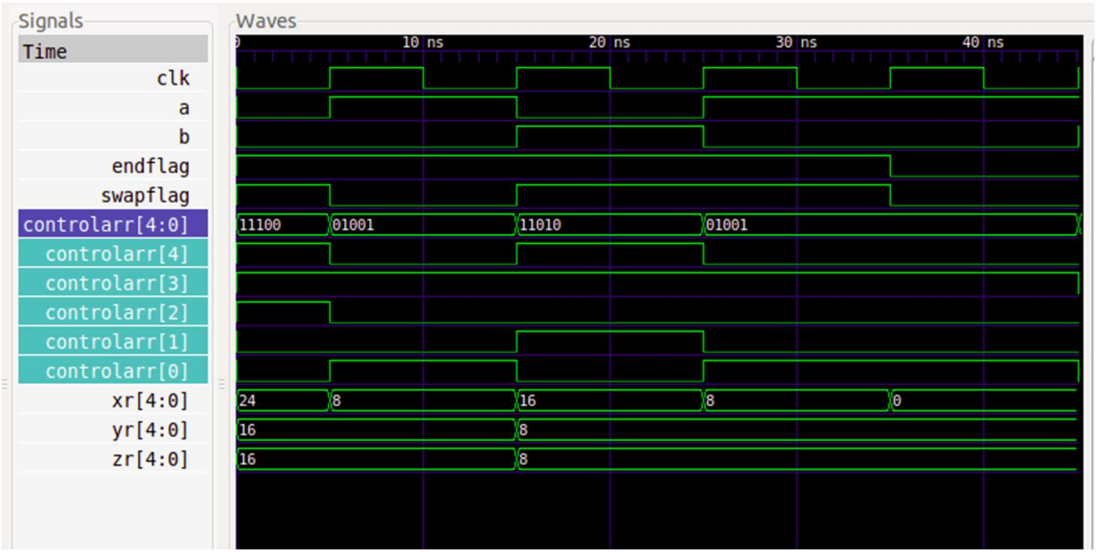
We can clearly observe the value of Zr when the process ends to be 1.

GCD(24, 16)



GCD=8

GTKwave plot:



We can clearly observe the value of Zr when the process ends to be 8.

You can find the netlists and modules in the following link:

https://iiitaphyd-my.sharepoint.com/:f:/g/personal/sri_surya_students_iiit_ac_in/EvdbFx99ePZJo6RmGSrLG3cBHf9Qj8HNM-ntBSAx19DElQ?e=JOT7m5