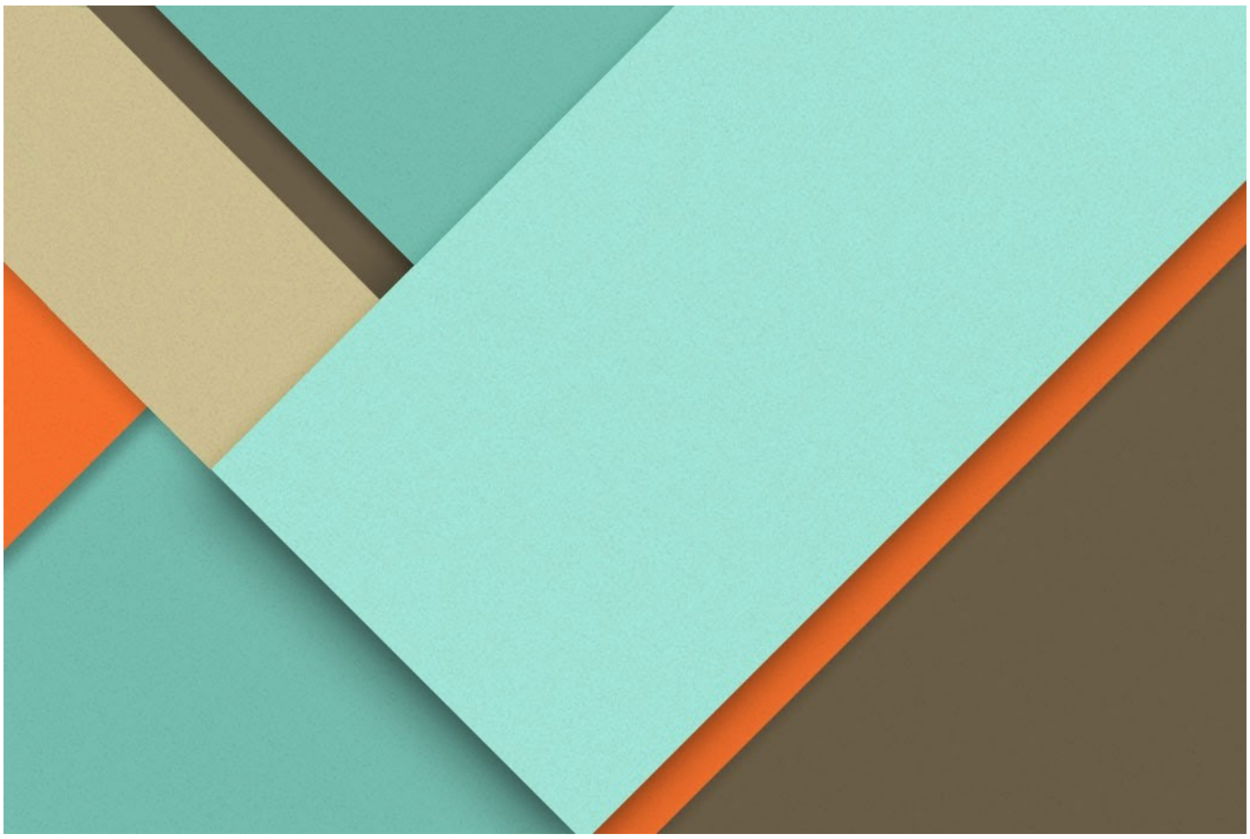


VLSI Course Project

Carry Look Ahead Adder

Shaantanu kulkarni
2019102031

Proposed Structure for the Adder

The main goal of building a carry look ahead adder is to minimize the delay as compared to the ripple carry adder where the following adders had to wait for the carry generations from the previous stages.

Consider 2 four-bit numbers that are to be added a4a3a2a1 and b4b3b2b1 and we are generating two signals pi (propagate signal) and gi (generate signal).

$$p_i = a_i \oplus b_i$$

$$g_i = a_i . b_i$$

Note: Some notations have been changed from what were given in the question to simplify the interpretability. However the overall circuit is the same.

And the carry bit from ith adder is ci and sum is sumi which is given as:

ci = (pi .c(i-1)) + gi where i = 1,2,3,4.

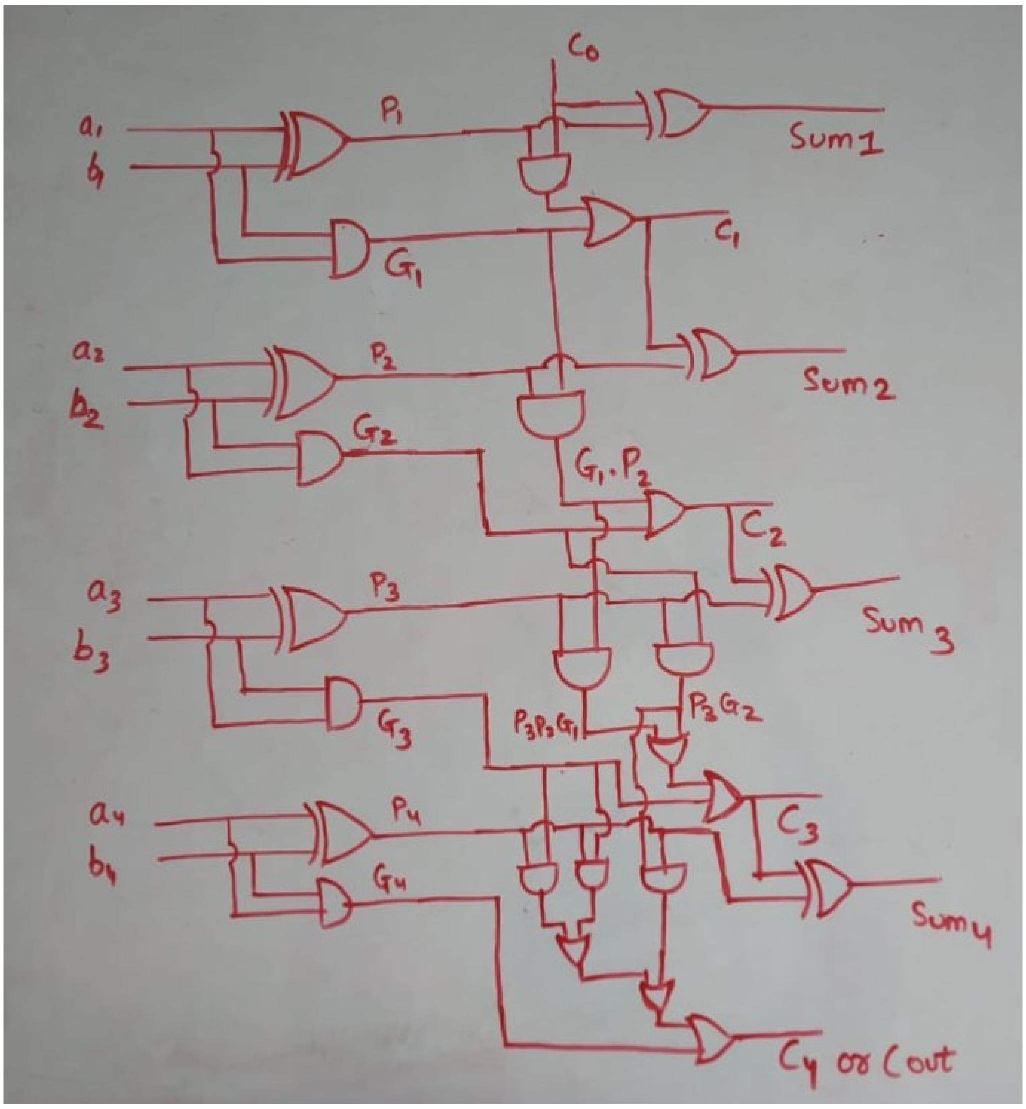
sumi = pi ⊕ ci

- C1 = C0.P0+G0.
- C2 = C1.P1+G1 = (C0.P0+G0).P1+G1 = C0.P0.P1 + G0.P1 + G1
- C3 = C2.P2+G2 = C1.P1.P2 + G1.P2 + G2 = C0.P0.P1.P2 + G0.P1.P2 + G1.P2 +G2
- C4 = C3.P3+G3 = C0.P0.P1.P2.P3 + P3.P2.P1.G0 + P3.P2.G1 + G2.P3 + G3

Some Specifications :

- We know that the C0 bit is always ‘0’ so we can eliminate the first term in every expression of Ci
- Instead of using multiple input gates, we can use 2 input gates multiple times and reuse the common terms from previous stages so that we reduce the overall number of gates. Specifically, for some three terms and in previous stage can be directly used as input to 2 input and gate which can effectively act as a four input and gate.
- Example: Consider the term G0.P1.P2 used to calculate C3 . Now this term is partially present in P3.P2.P1.G0 which is used to calculate C4 . This former output can be used to calculate latter directly by anding it with P3. Also we are not waiting for any input to set then calculate some value, so its now like ripple adder. We are just reusing some block from previous stage.

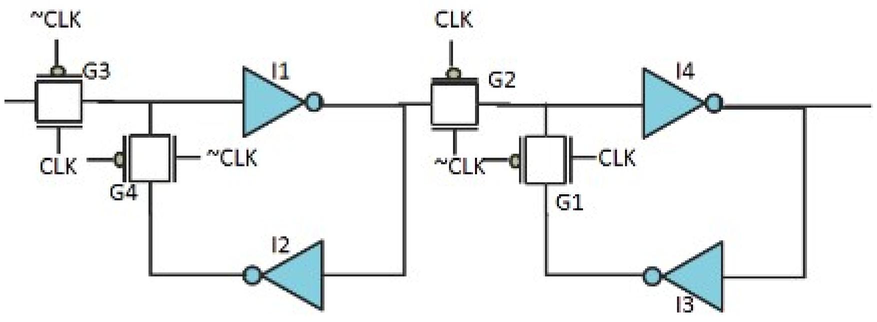
Gate Level Diagram of Proposed Adder



- We can see that C0 is always zero so we dont need in in later stages
- Reusing the outputs from previous stages reduces number of gates and we dont have to wait like ripple carry adder.

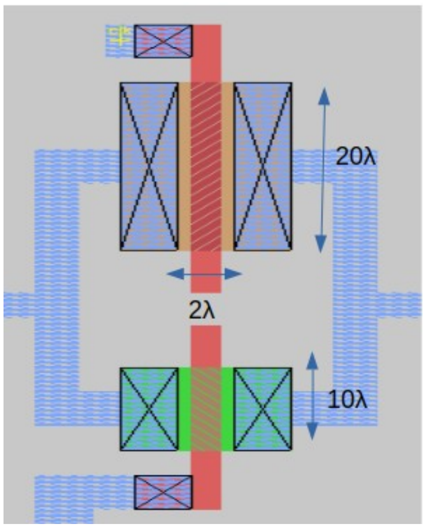
Topology And Sizing

D Flip-Flop



The above circuit was used to make D-FlipFlops using pass transistor logic.

Pass Transistor Gate:

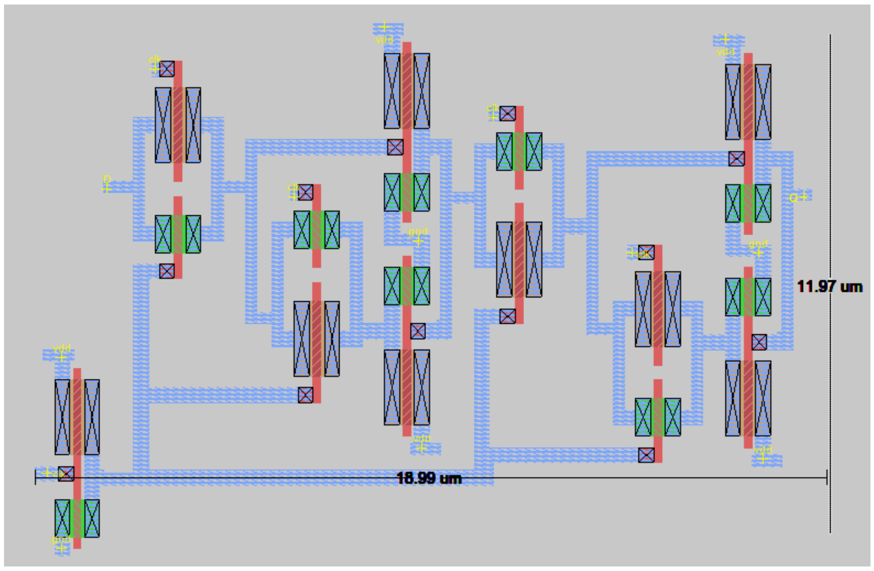


Dimensions of Pass Transistor :

Pmos width = 20 λ

Nmos width = 10 λ

D FlipFlop Dimensions:



Dimensions of D Flip-Flop :

18,99 um X 11.97um

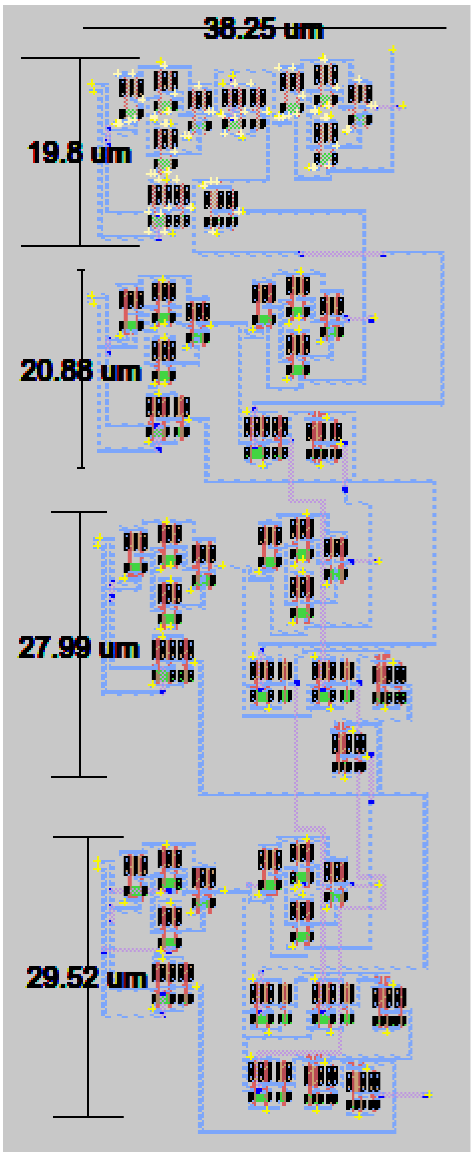
= 227.3103 um^2

~ 28063 λ^2

each Pmos has width of 20λ

each Nmos has width of 10λ

Adder Module:



Width of the total Adder circuit:

W = 38.25 um

Heights of individual bit adders:

Adder 1: 19.8um

Adder 2: 20.88um

Adder 3: 27.99um

Adder 4: 29.52um

Total height = 98.19 um

Total chip area covered by Adder:

= 38.25um X 98.19um

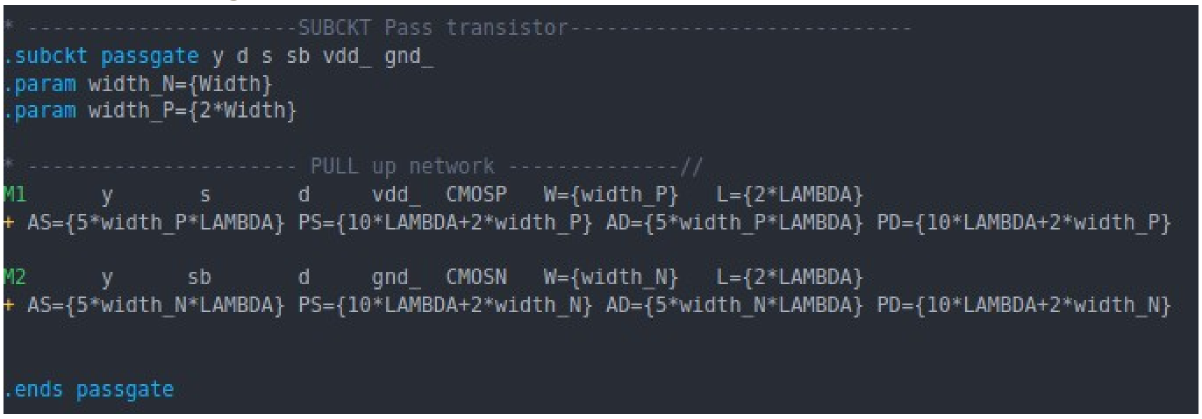
= 3755.7675 um^2

~ 463675 λ^2

Verification Of Functionality

1) D FlipFlop Functionality

Pass transistor gate Subcircuit:



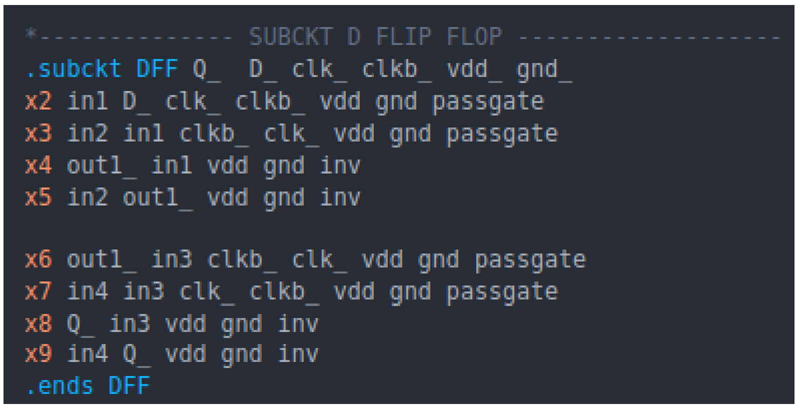
```
* ----------------------SUBCKT Pass transistor----------------------------
.subckt passgate y d s sb vdd_ gnd_
.param width_N={Width}
.param width_P={2*Width}

* ---------------------- PULL up network --------------//
M1      y       s       d      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}

M2      y      sb       d      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 passgate
```

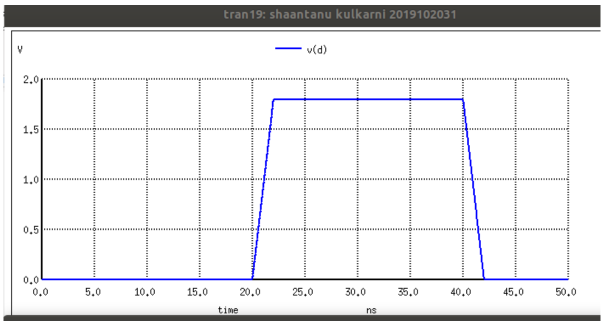
DFF Subcircuit using two D latches in master slave arrangement:

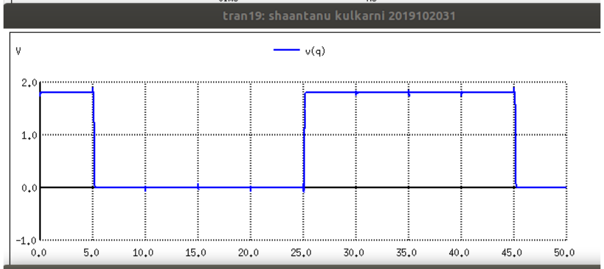


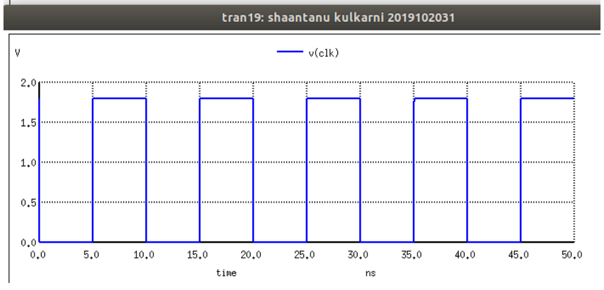
```
*-------------- SUBCKT D FLIP FLOP -------------------
.subckt DFF Q_  D_ clk_ clkb_ vdd_ gnd_
x2 in1 D_ clk_ clkb_ vdd gnd passgate
x3 in2 in1 clkb_ clk_ vdd gnd passgate
x4 out1_ in1 vdd gnd inv
x5 in2 out1_ vdd gnd inv

x6 out1_ in3 clkb_ clk_ vdd gnd passgate
x7 in4 in3 clk_ clkb_ vdd gnd passgate
x8 Q_ in3 vdd gnd inv
x9 in4 Q_ vdd gnd inv
.ends DFF
```

Simulation Results:



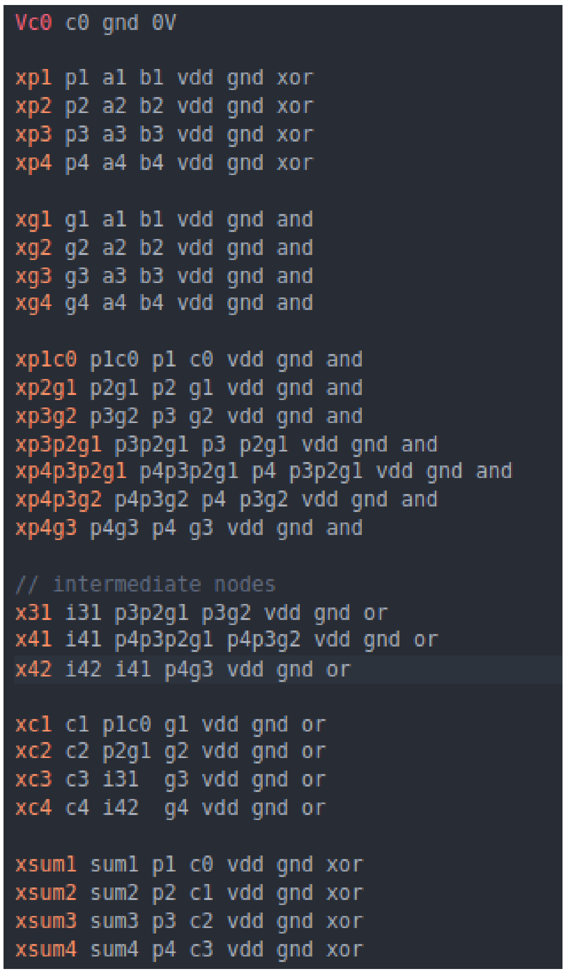




- The Topmost signal is the input signal to the D FlipFlop
- Middle one is the output from the D FlipFlop
- The lower plot shows he clock signal
- Clk Timeperiod = 10ns

1) CLA module Functionality

CLA module netlist



```
Vc0 c0 gnd 0V

xp1 p1 a1 b1 vdd gnd xor
xp2 p2 a2 b2 vdd gnd xor
xp3 p3 a3 b3 vdd gnd xor
xp4 p4 a4 b4 vdd gnd xor

xg1 g1 a1 b1 vdd gnd and
xg2 g2 a2 b2 vdd gnd and
xg3 g3 a3 b3 vdd gnd and
xg4 g4 a4 b4 vdd gnd and

xp1c0 p1c0 p1 c0 vdd gnd and
xp2g1 p2g1 p2 g1 vdd gnd and
xp3g2 p3g2 p3 g2 vdd gnd and
xp3p2g1 p3p2g1 p3 p2g1 vdd gnd and
xp4p3p2g1 p4p3p2g1 p4 p3p2g1 vdd gnd and
xp4p3g2 p4p3g2 p4 p3g2 vdd gnd and
xp4g3 p4g3 p4 g3 vdd gnd and

// intermediate nodes
x31 i31 p3p2g1 p3g2 vdd gnd or
x41 i41 p4p3p2g1 p4p3g2 vdd gnd or
x42 i42 i41 p4g3 vdd gnd or

xc1 c1 p1c0 g1 vdd gnd or
xc2 c2 p2g1 g2 vdd gnd or
xc3 c3 i31  g3 vdd gnd or
xc4 c4 i42  g4 vdd gnd or

xsum1 sum1 p1 c0 vdd gnd xor
xsum2 sum2 p2 c1 vdd gnd xor
xsum3 sum3 p3 c2 vdd gnd xor
xsum4 sum4 p4 c3 vdd gnd xor
```

Total number of gates used:

XOR : 8 gates

AND : 11 gates

OR   : 7 gates

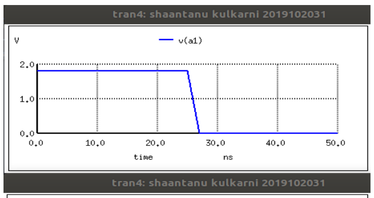
Total gates used: 25 gates:

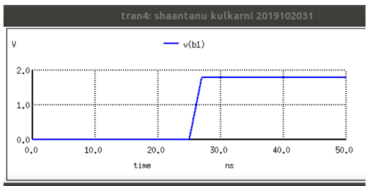
For testing the functionality two testcases are used:

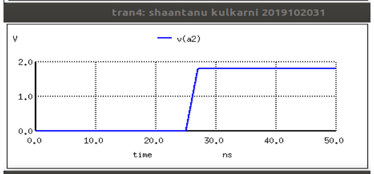
- input A = 0101 ; input B = 1000
- input A = 0110 ; input B = 0011

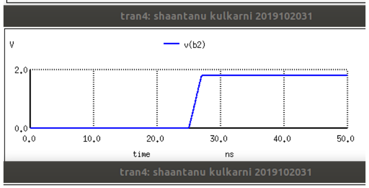
Input A

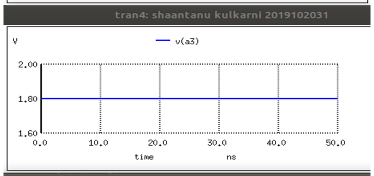
Input B

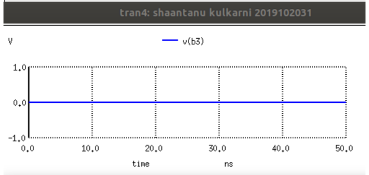


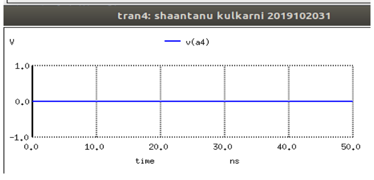


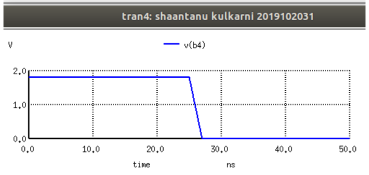




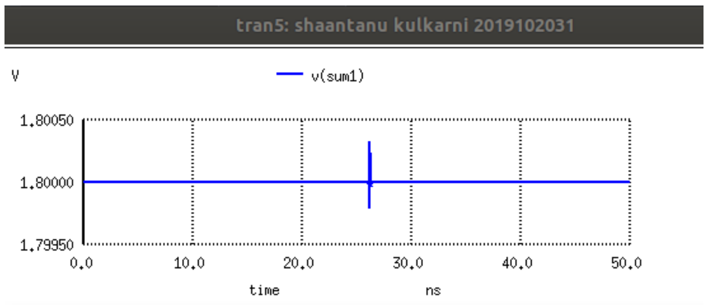


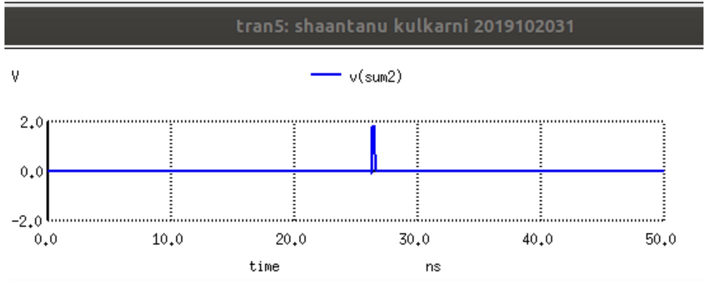


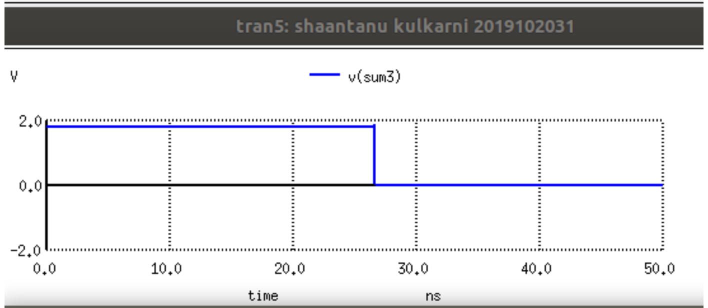


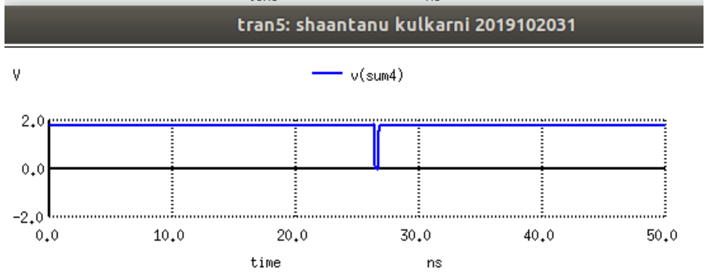


Sum bits:

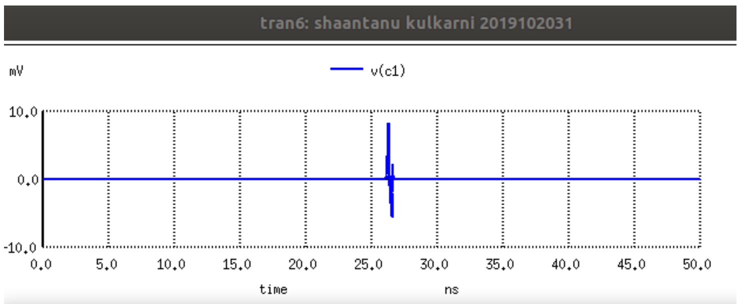


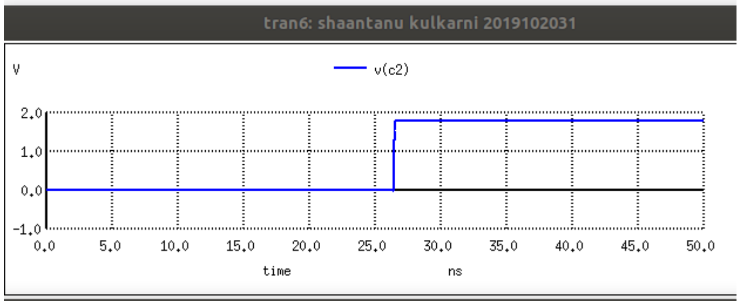


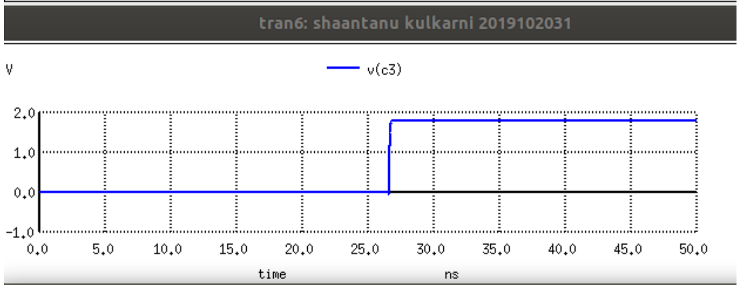


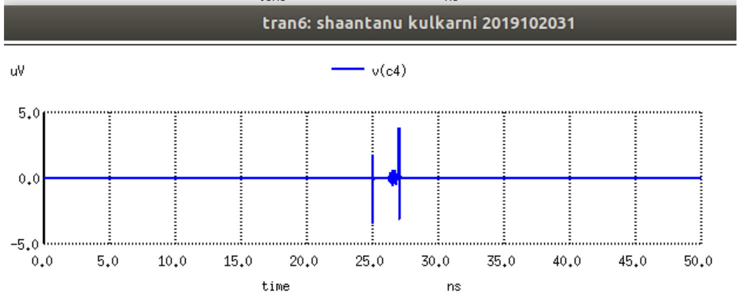


Carry bits:





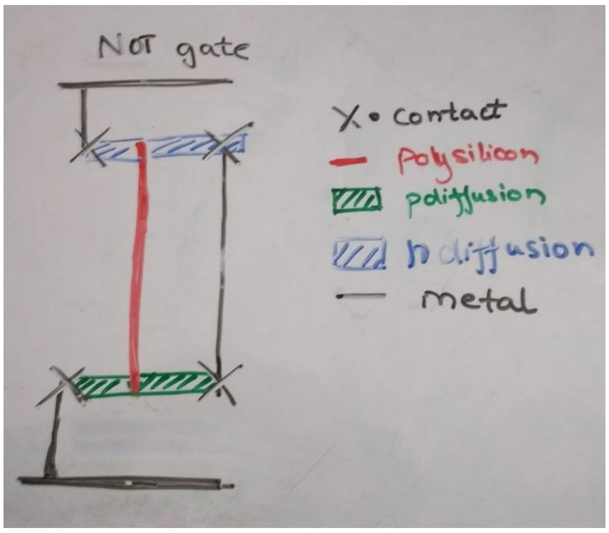


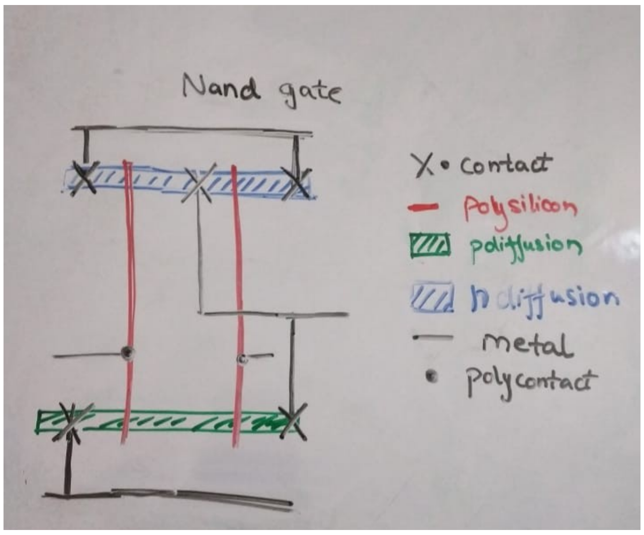


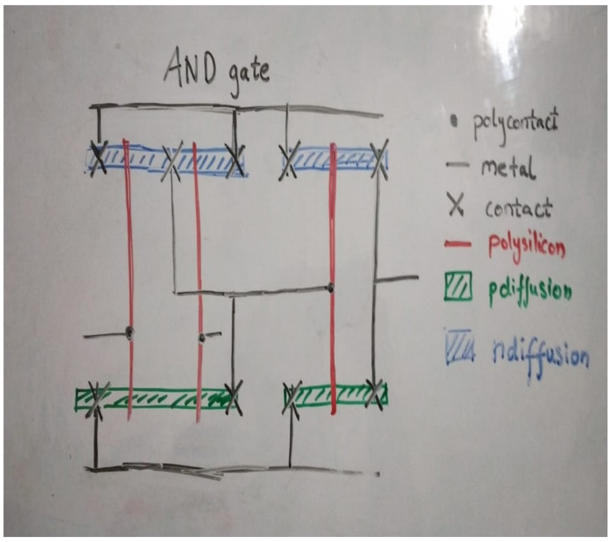
As expected the outputs of the two testcases are

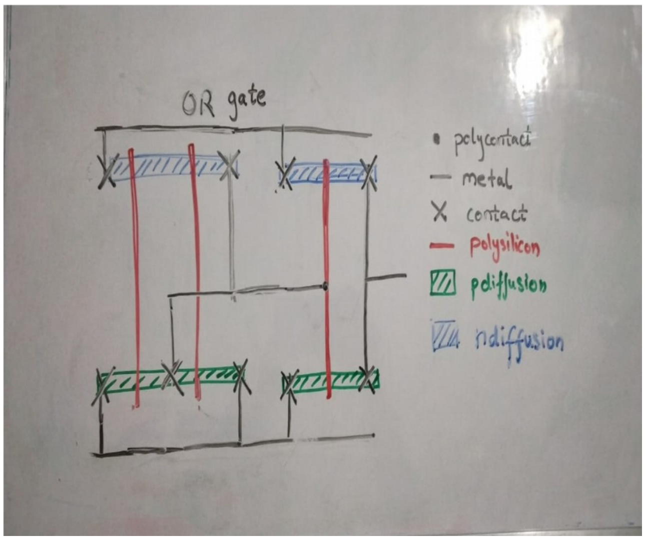
- sum = 1101 & c1 = 0,c2=0,c3= 0,c4=0
- sum = 1001 & c1=0,c2=1,c3=1,c4=0

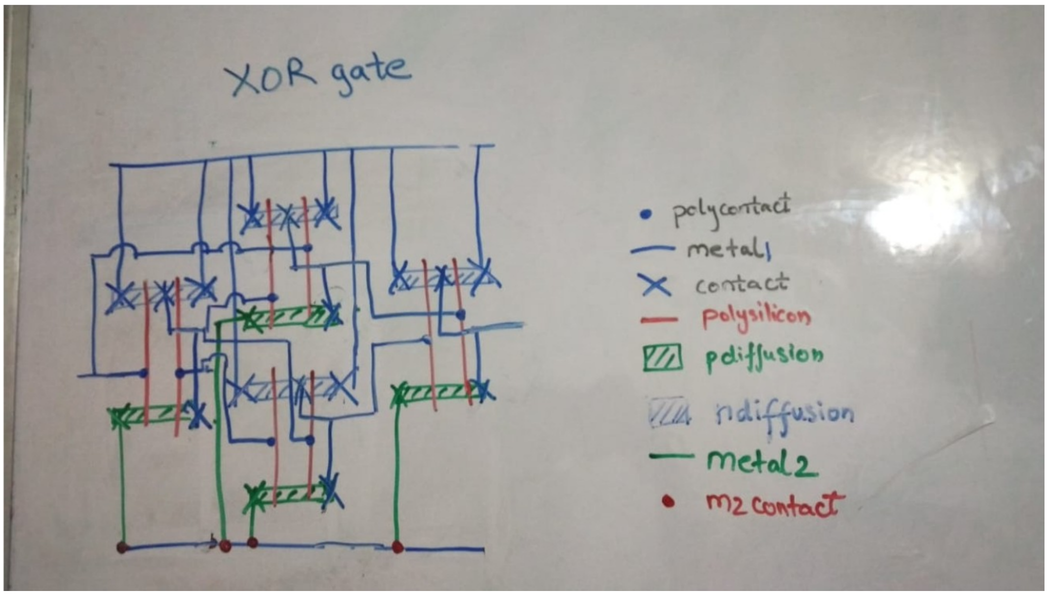
Stick diagrams of unique gates



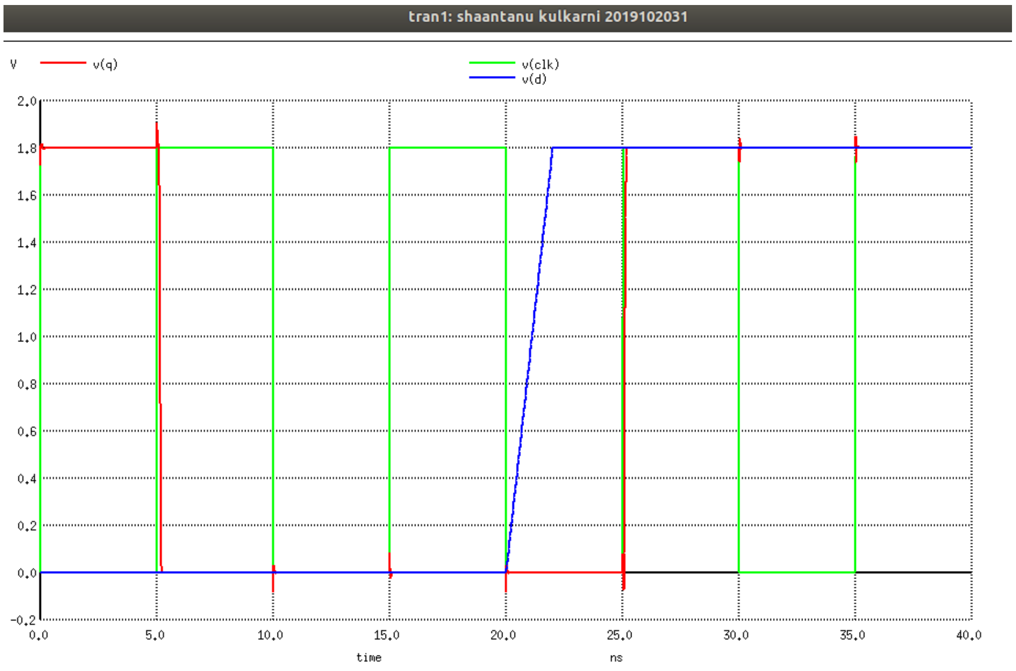








D Flip Flop Delays



Delays:

Tdhl = 0.11ns Tdhl = 0.33ns

Tpd = (Tdhl+tdlh)/2 = (0.33 + 0.11)/2 = 0.22 ns

For T-setup, the rise time of the input is increased slowly unless we can see distorted outputs. It was found that

T-setup = 0.044ns

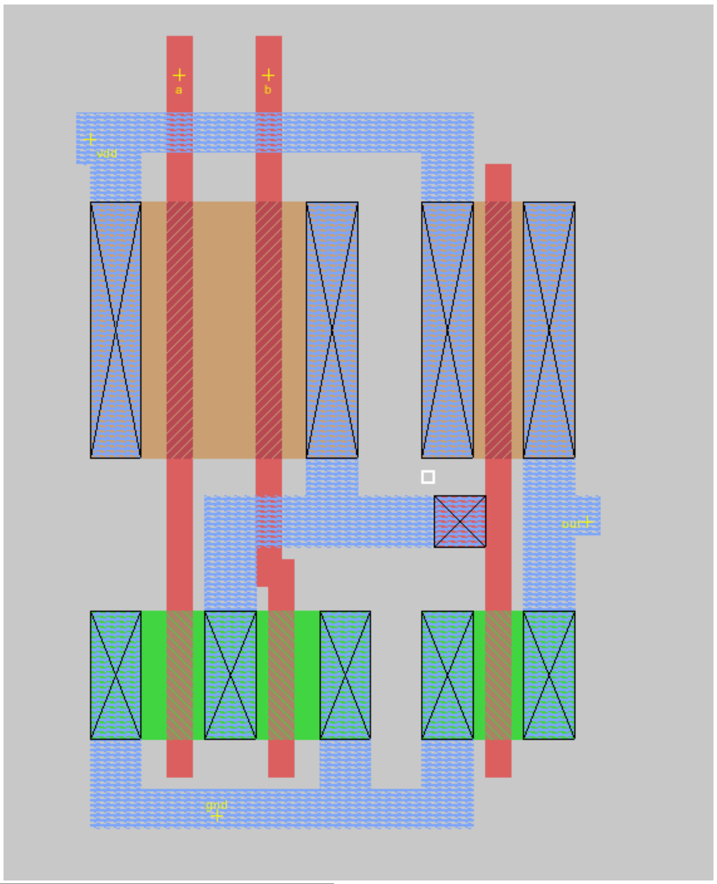
For T-hold, after the input was reflected at output of Dff, the input was suddenly changed and this time interval was decreased until distorted output was seen.

T-hold = 0.05ns

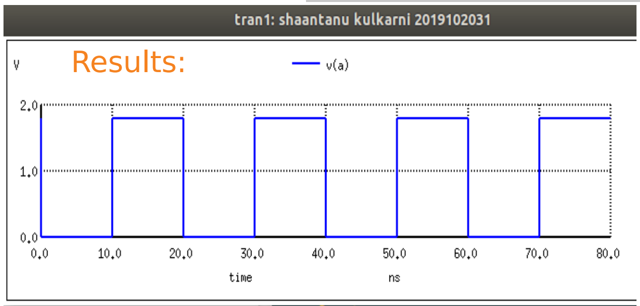
Layouts

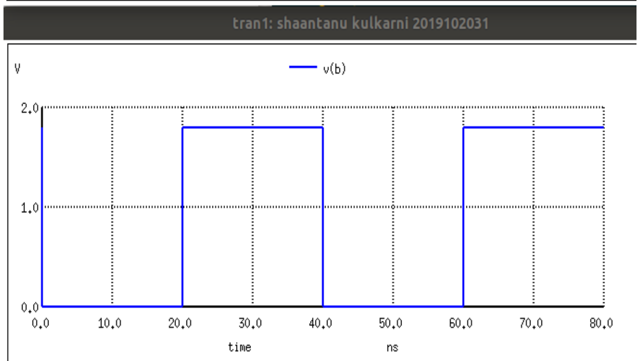
Note: For simulation of all the gates , all possible 4 inputs are provided and the output is check

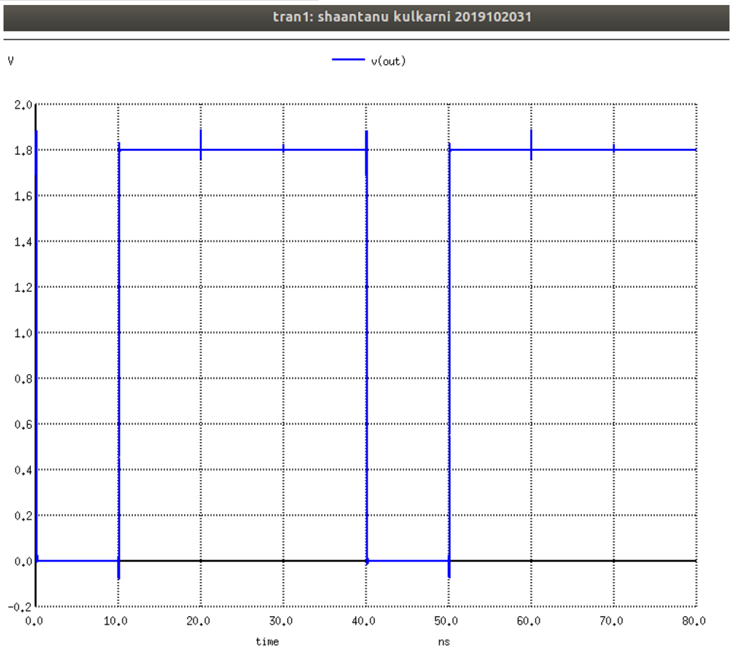
OR gate:



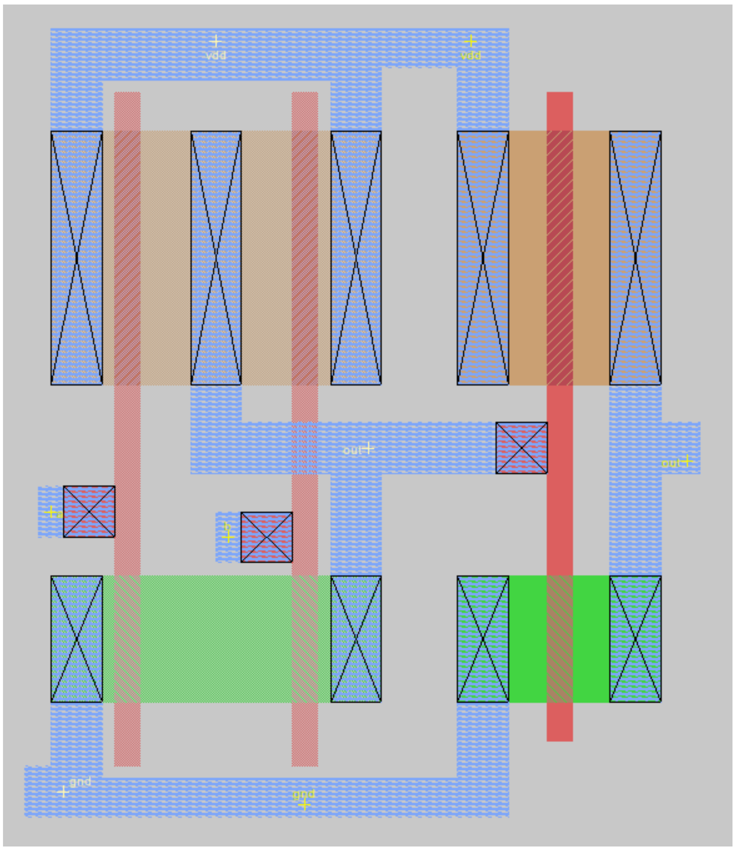
Results:



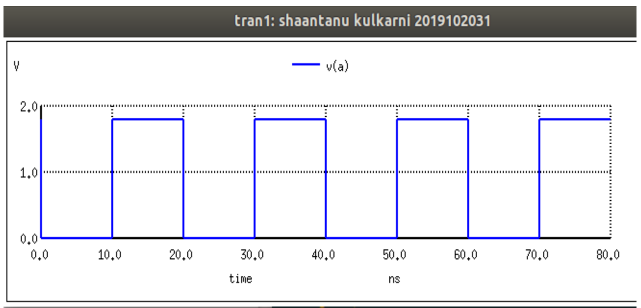


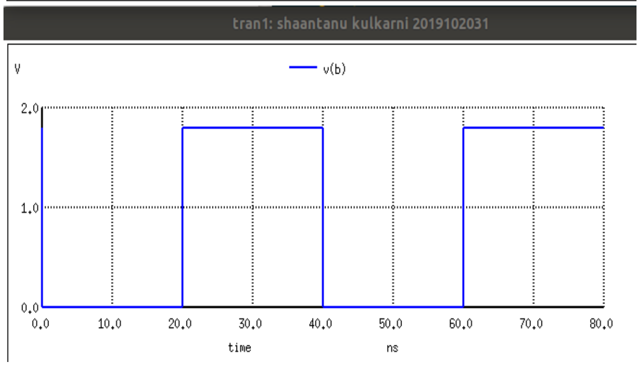


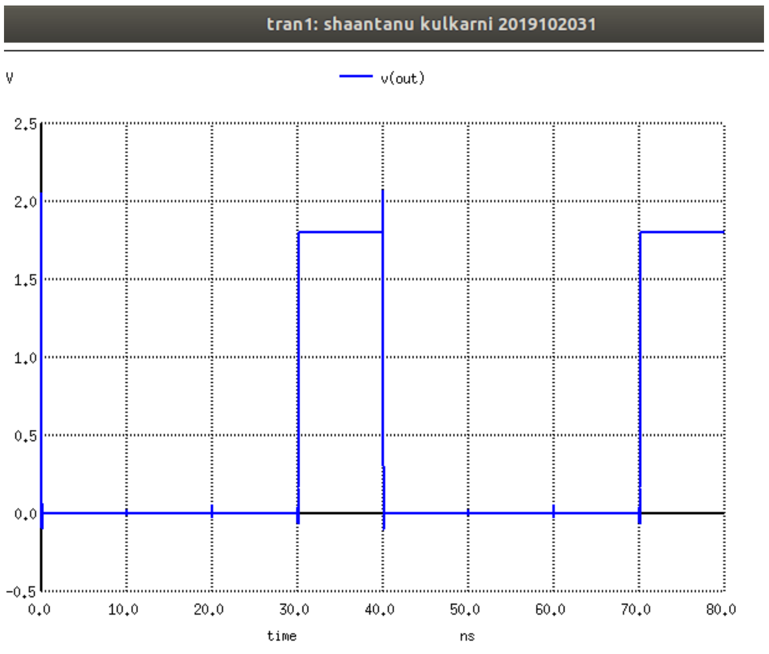
AND gate:



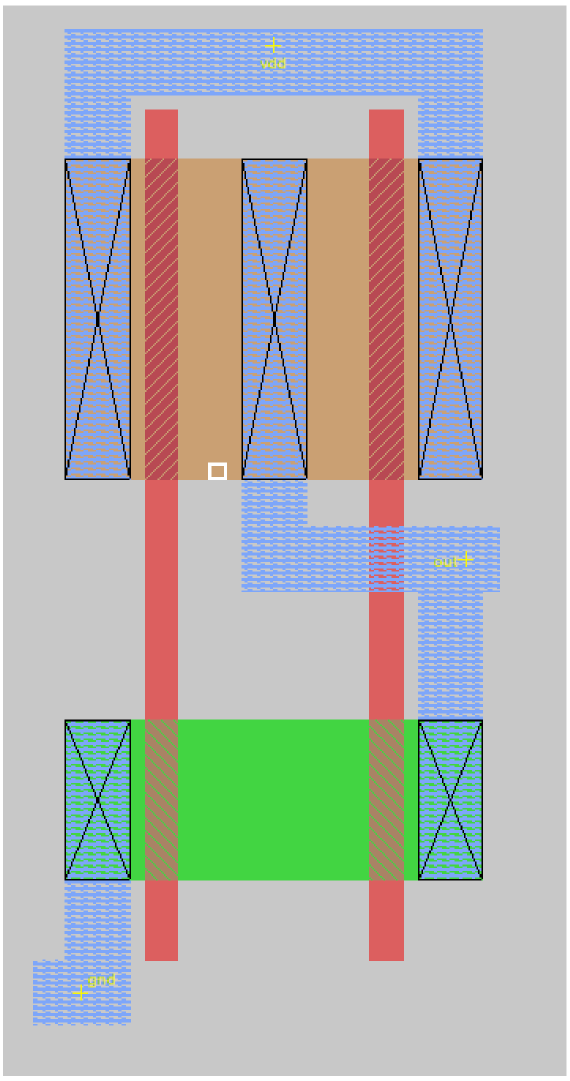
Results:



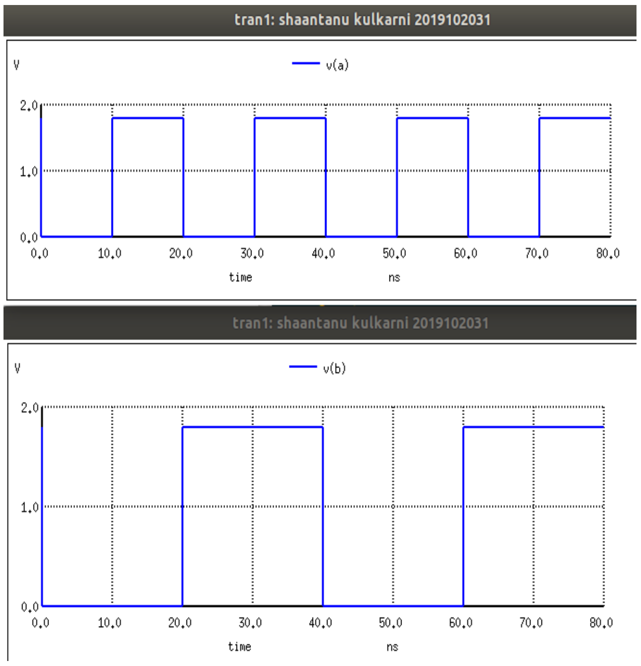


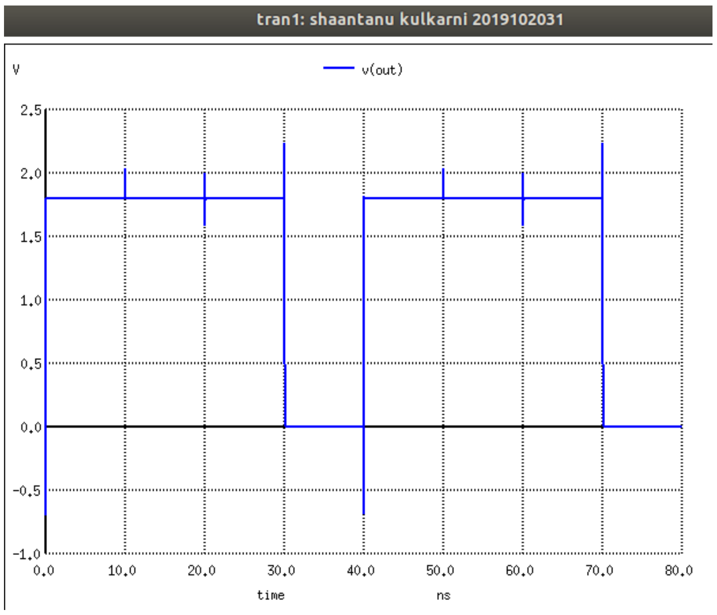


NAND gate:

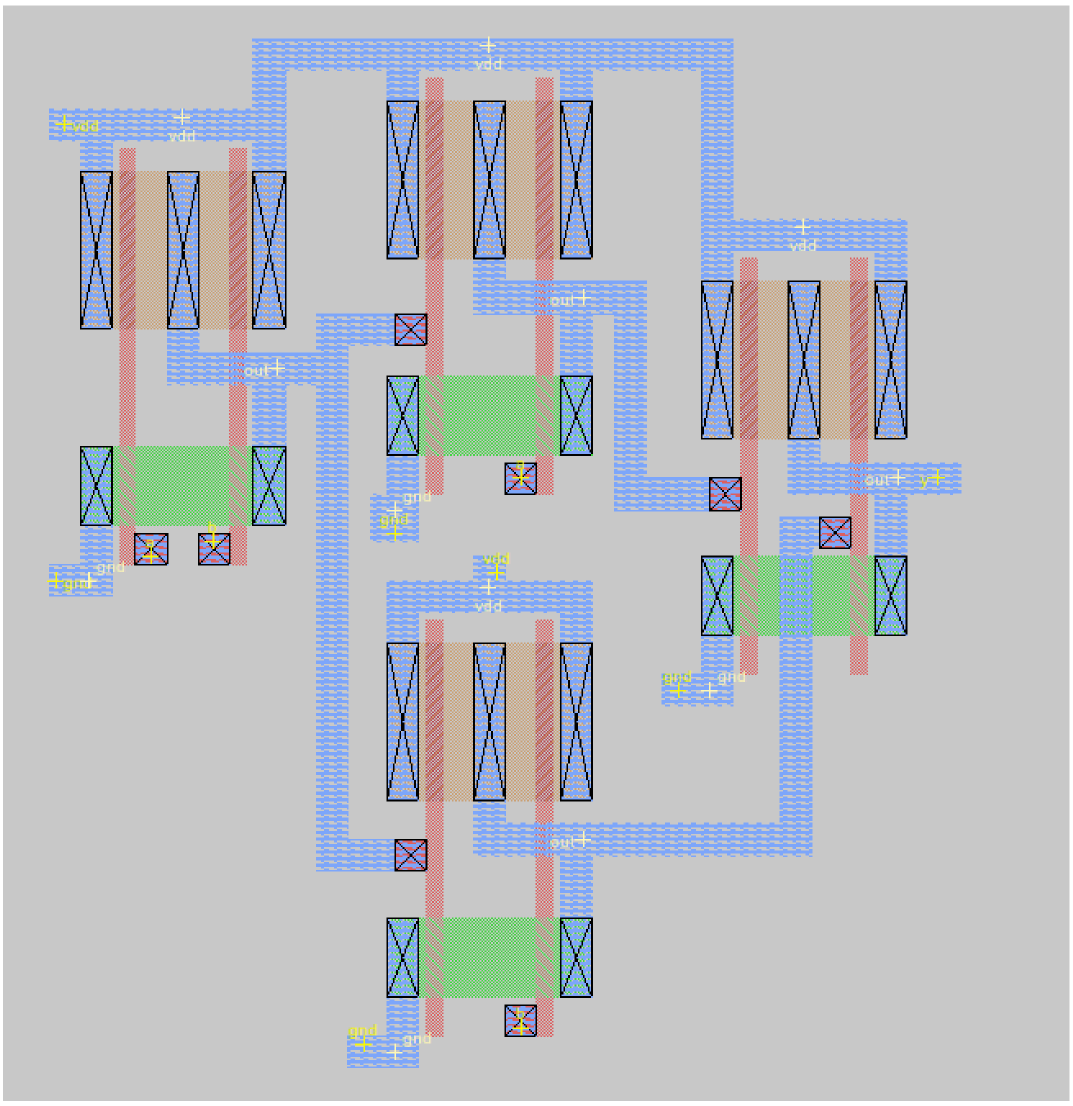


Results:

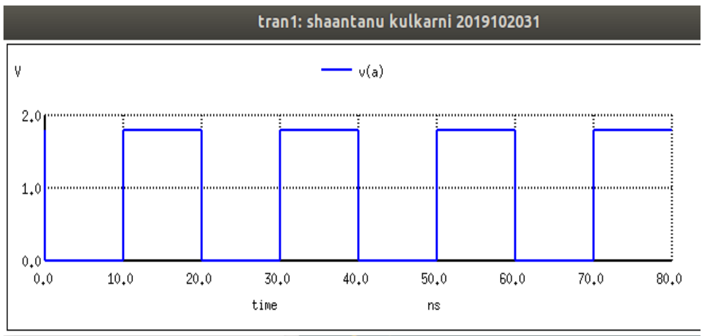


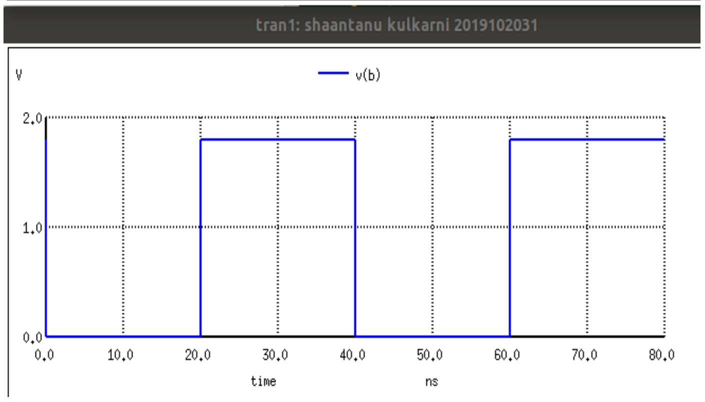


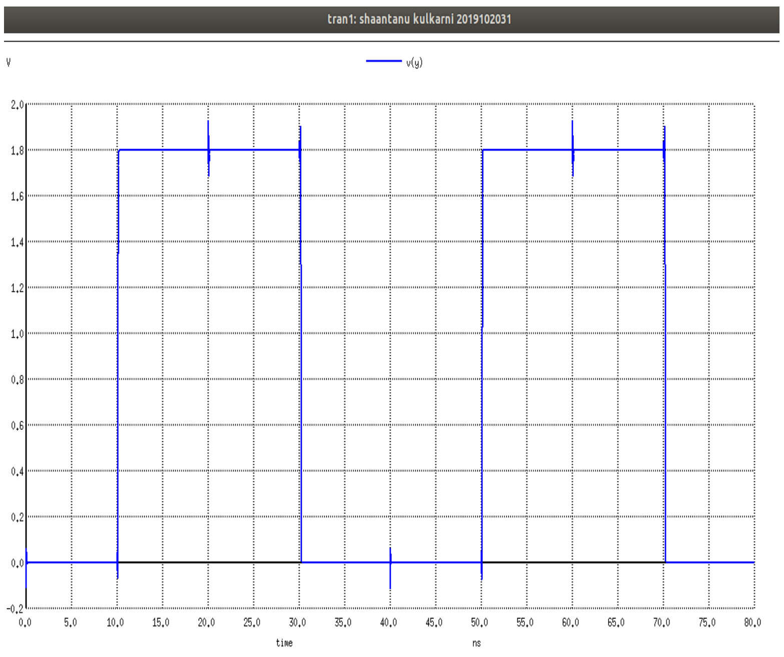
XOR gate:



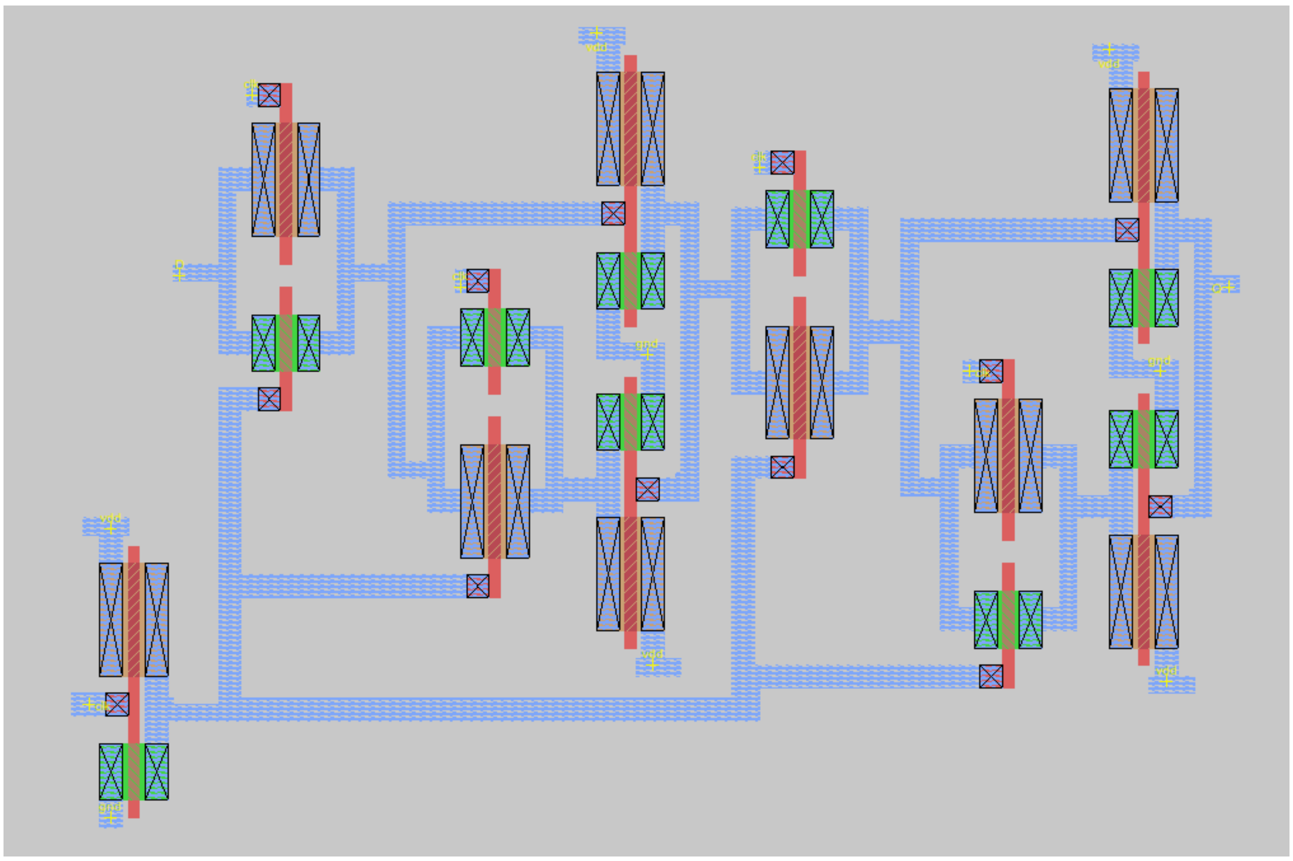
Results:





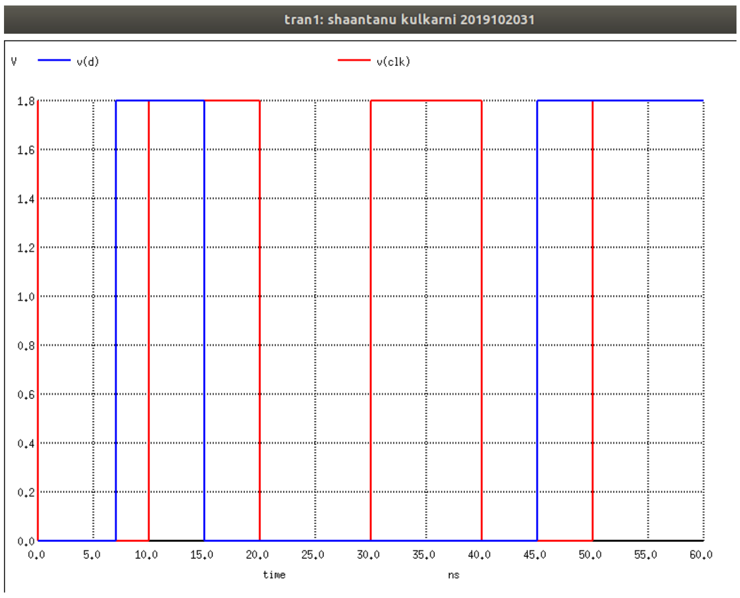


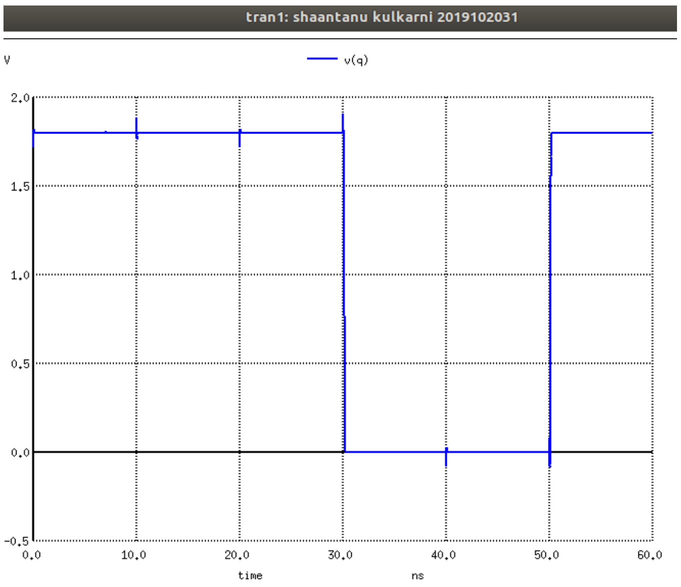
D FlipFlop:

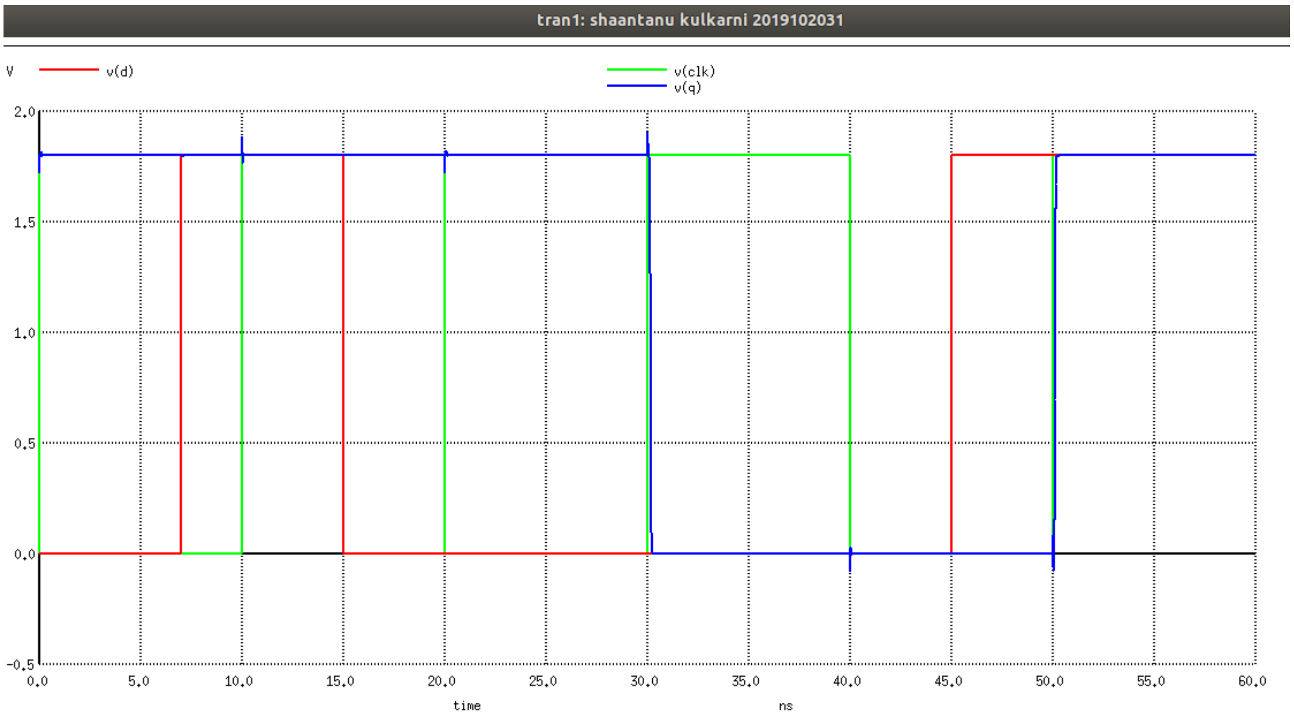


Results:

- The red plot is Clock signal and blue is the input to D flipflop
- The signal Q is the output of D flipflop which is updated at positive edge of the clock signal







Delays:

Tdhl = 0.13ns Tdlh = 0.089ns

Tpd = (Tdhl+tdlh)/2 = (0.13 + 0.089)/2 = 0.1095 ns

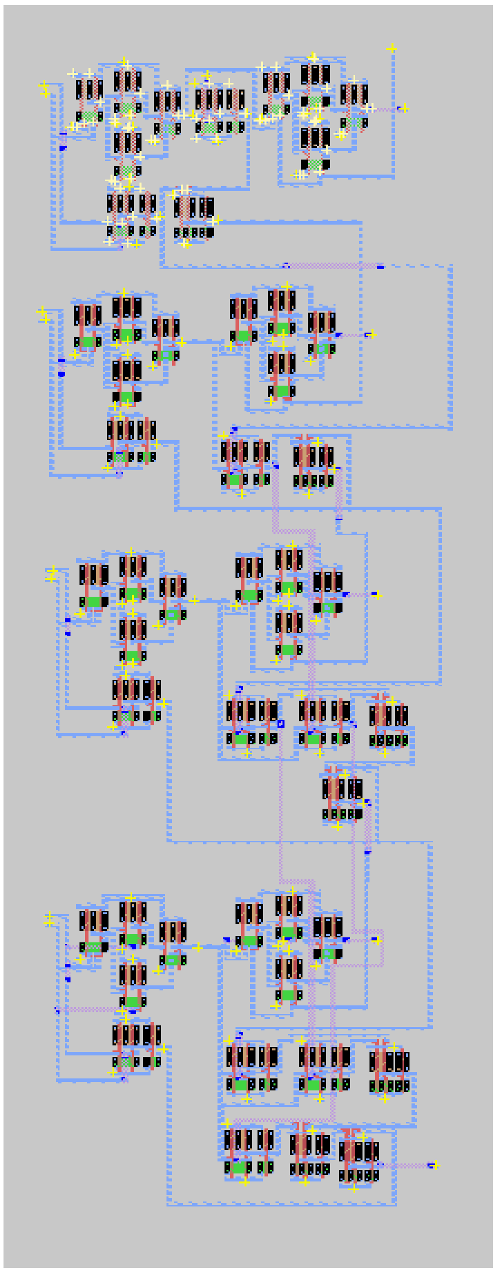
For T-setup, the rise time of the input is increased slowly unless we can see distorted outputs. It was found that

T-setup = 0.038ns

For T-hold, after the input was reflected at output of Dff, the input was suddenly changed and this time interval was decreased until distorted output was seen.

T-hold = 0.035ns

CLA module:

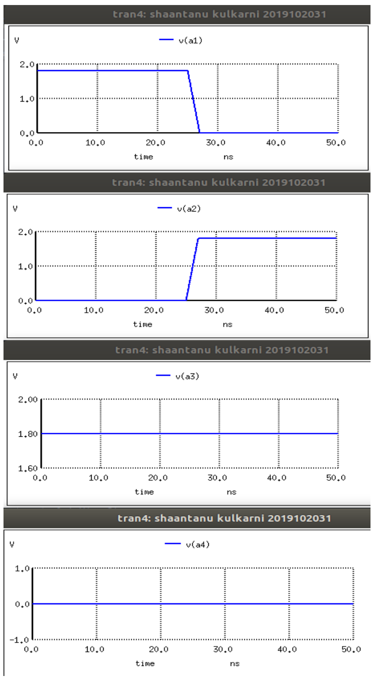


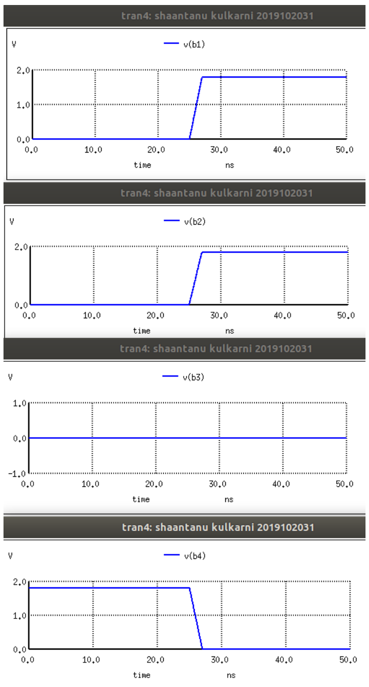
For testing the functionality two testcases are used:

- input A = 0101 ; input B = 1000
  - input A = 0110 ; input B = 0011

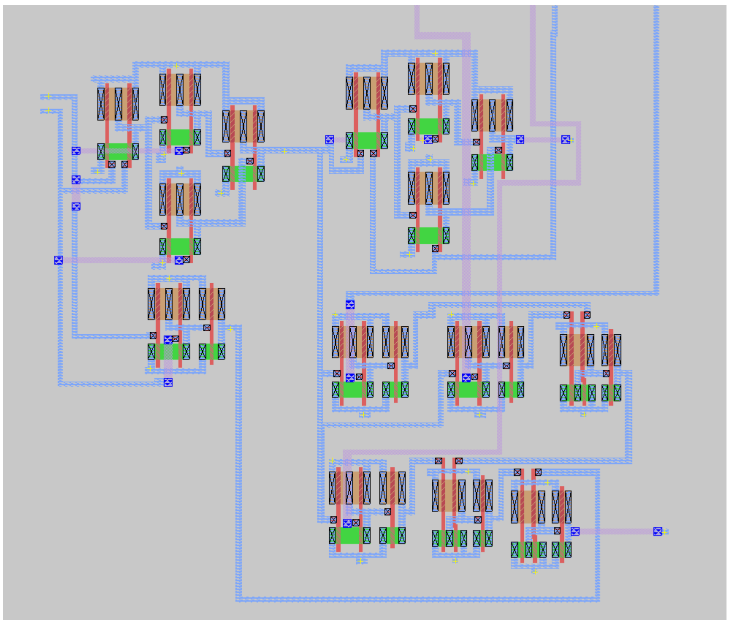
Input A

Input B



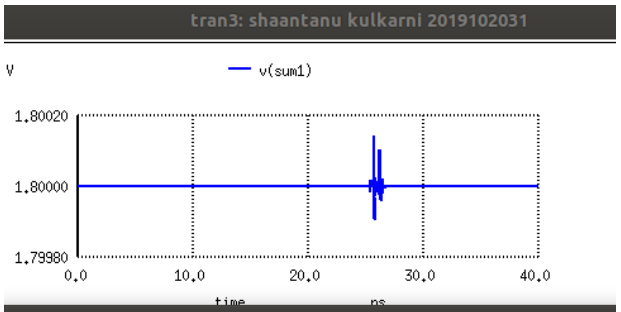


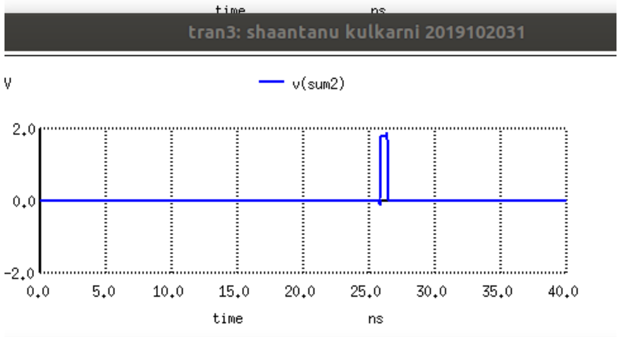
A screen shot of last bit adder to show the layout clearly

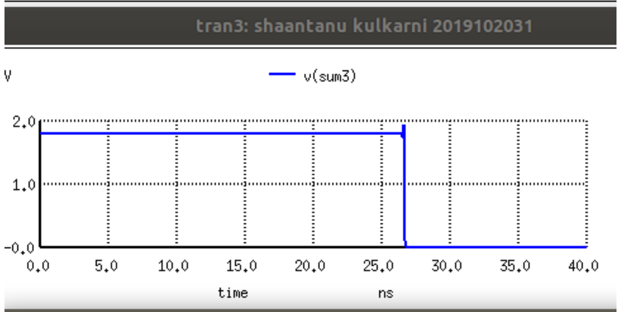


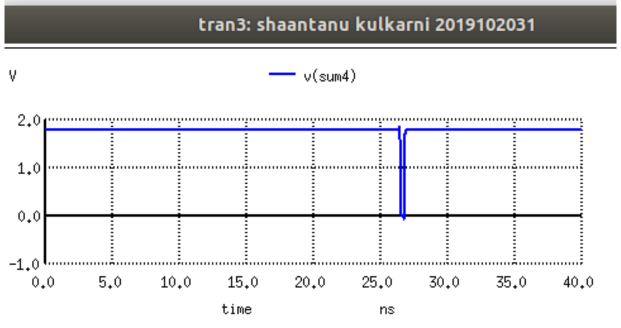
Results of post layout CLA module

Sum Bits:

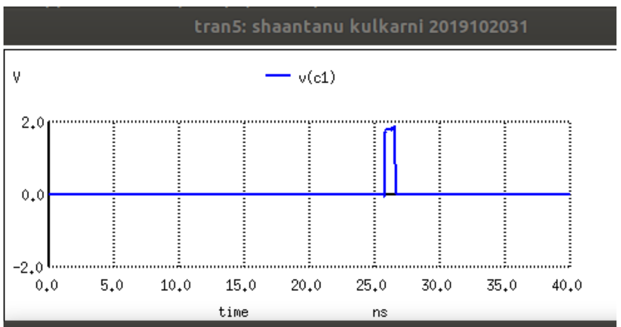


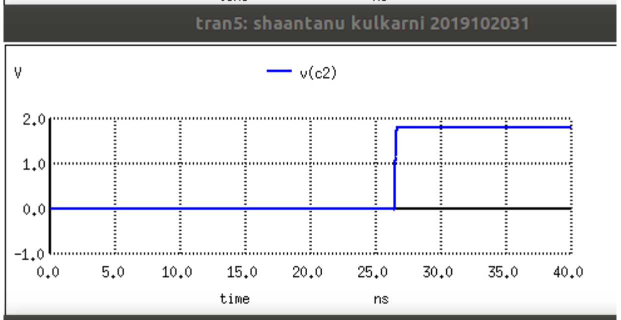


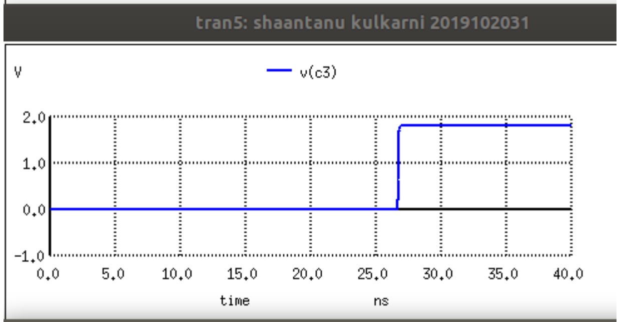


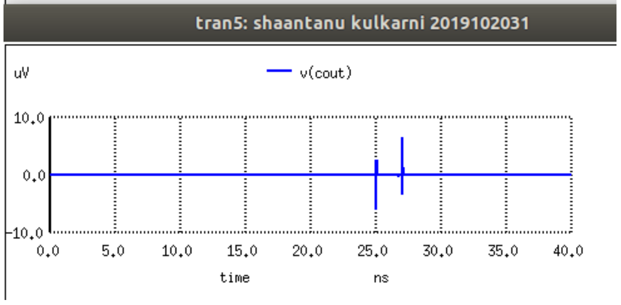


Carry bits:









As expected the outputs of the two testcases are

- sum = 1101 & c1 = 0,c2=0,c3= 0,c4=0
- sum = 1001 & c1=0,c2=1,c3=1,c4=0

Complete integrated simulation in Ngspice

For testing the complete integrated circuit , the test cases chosen are such that the worst case delay is expected on this test case.

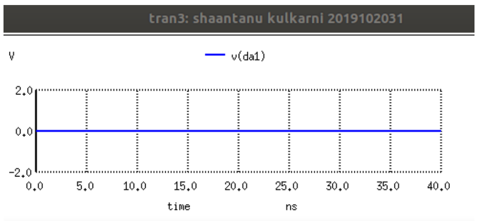
Testcase:

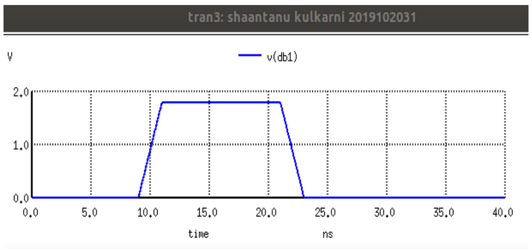
We want a testcase such that all the sum bits are changed and most of the voltages at the internal nodes change .

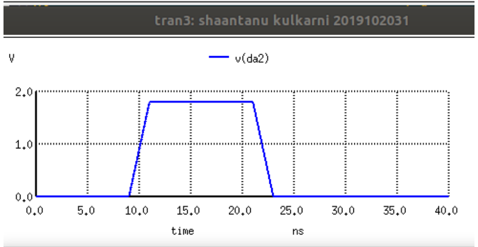
Say that the initial values of inputs are both (0000)2 so , the sum will also be (0000)2. Now we want the sum to change to (1111)2 and we also want maximum intermediate node values to change so that the delay is maximized.

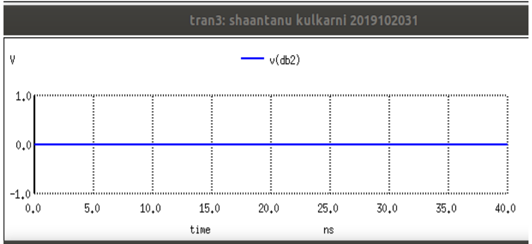
It turns out that if we choose input like A= (1010) and B = (0101), the sum will be (1111) and most of the intermediate tends to change . The intuition is that since Ai != Bi for ith bit, there are many xor gates in circuit. Initially these are at ‘0’. Now in the next clock cycle all these xors will change. Xor gates will have maximum delay because of 4 internal nand gates used in it.

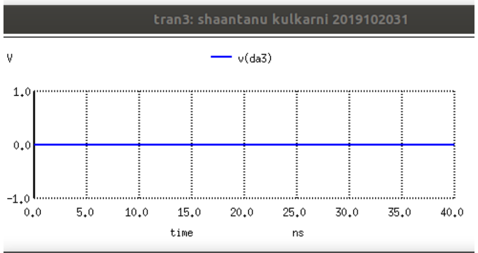
Inputs:

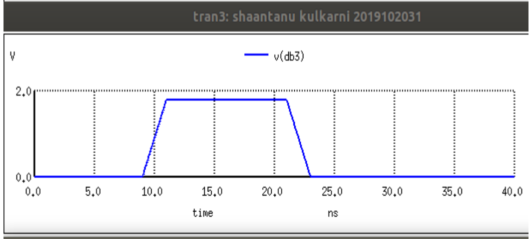


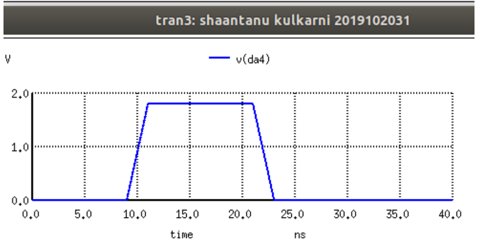


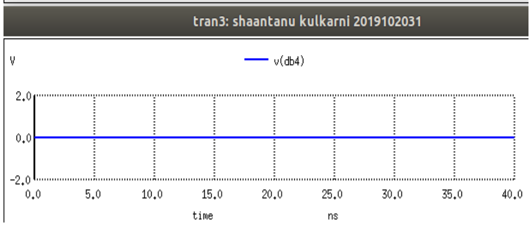




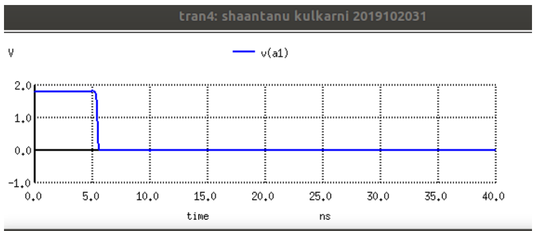


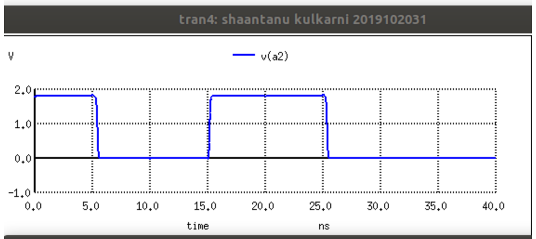


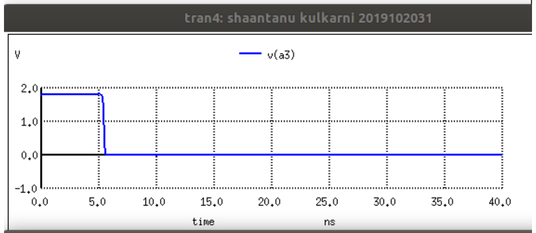


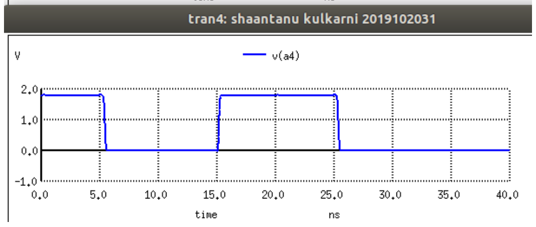


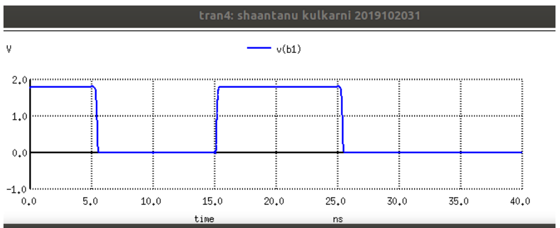
D flipflops (Inputs)

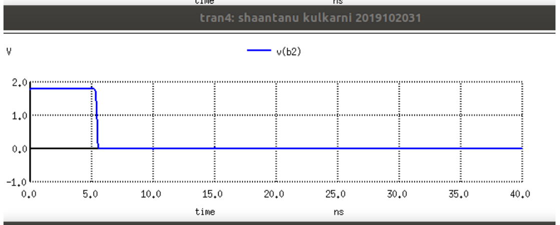


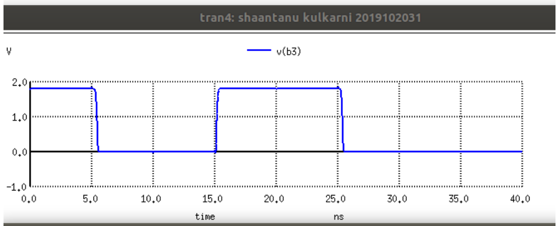


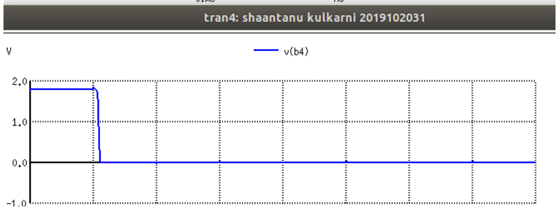








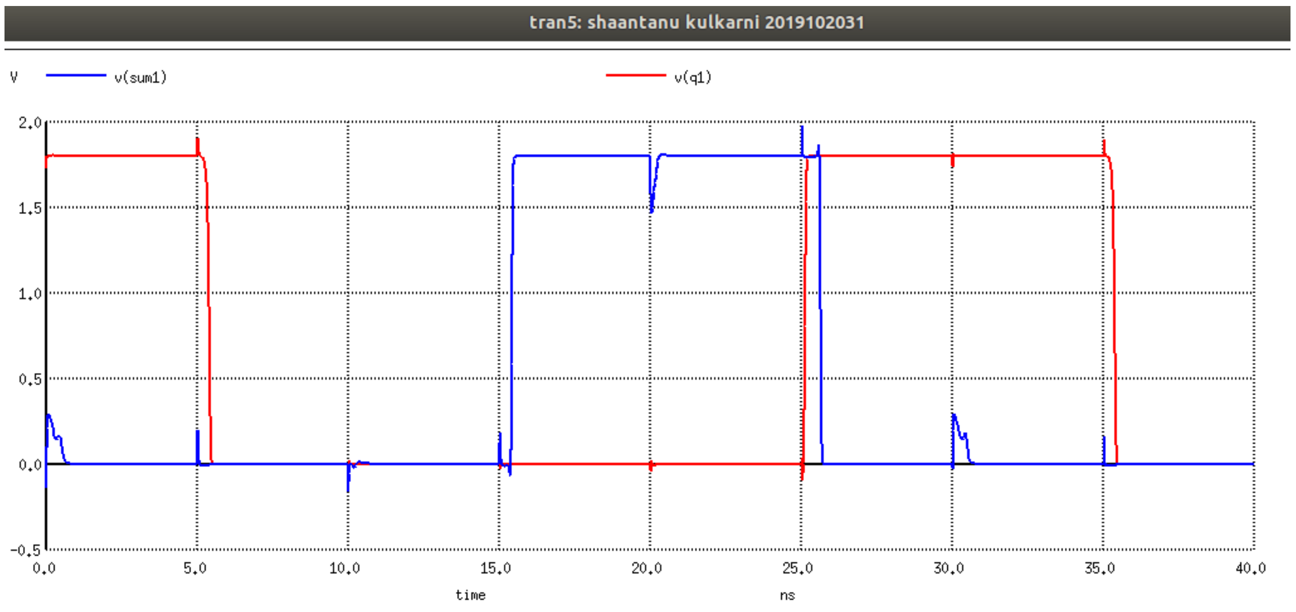




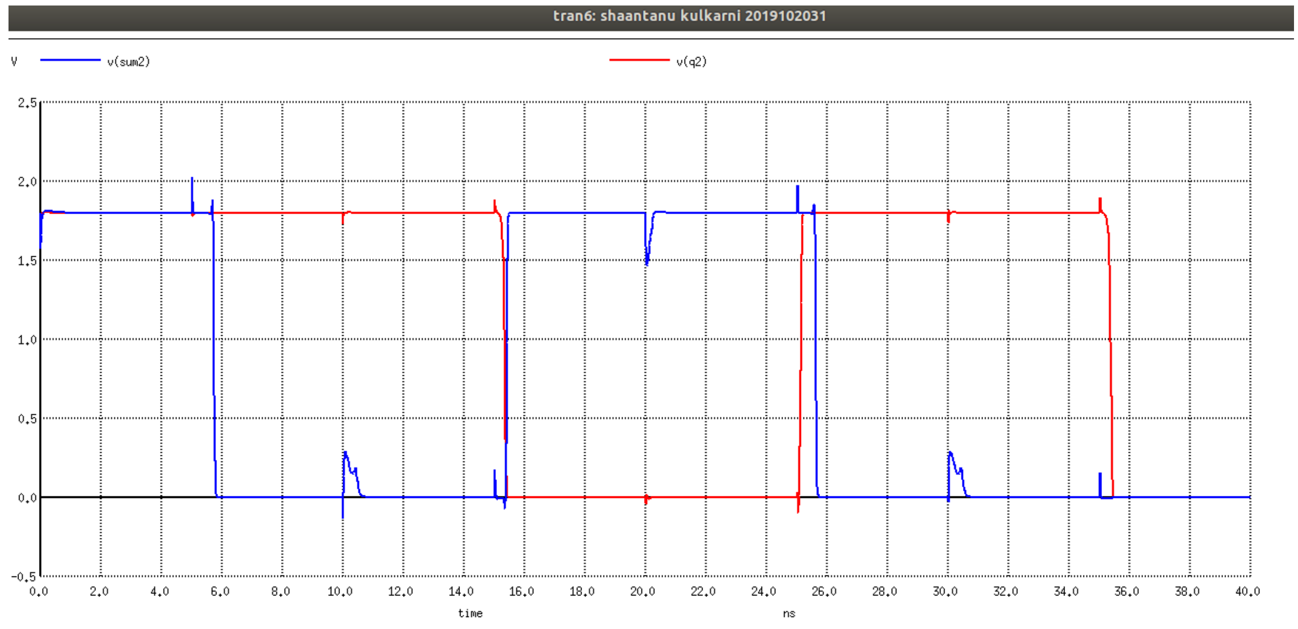
Note: The initially ‘1’ values are garbage initial values at the output of D flipflop. This are set to correct values after the first positive edge of clock at 5ns.

Note: Blue plot is the sum output and qi (red plot) is output of final D flipflop which is updated after one clock cycle as expected. So , initially for one and half clock cycle the output of D flipflop is corrupted.

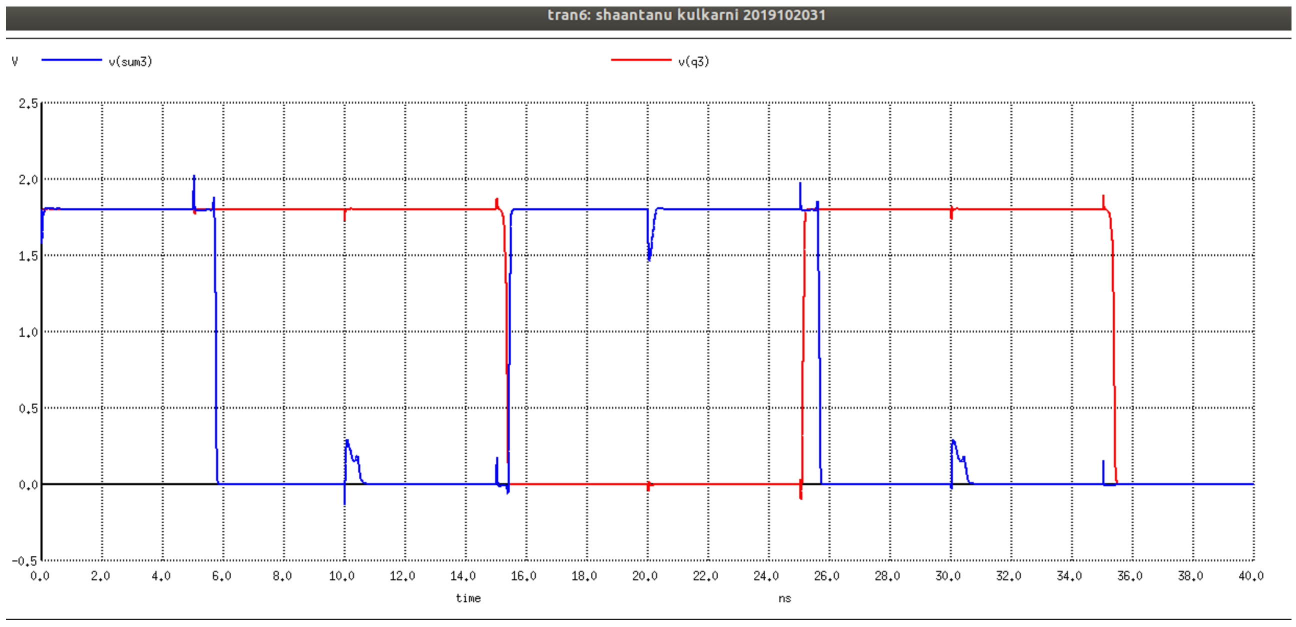
Sumbit 1 (LSB)



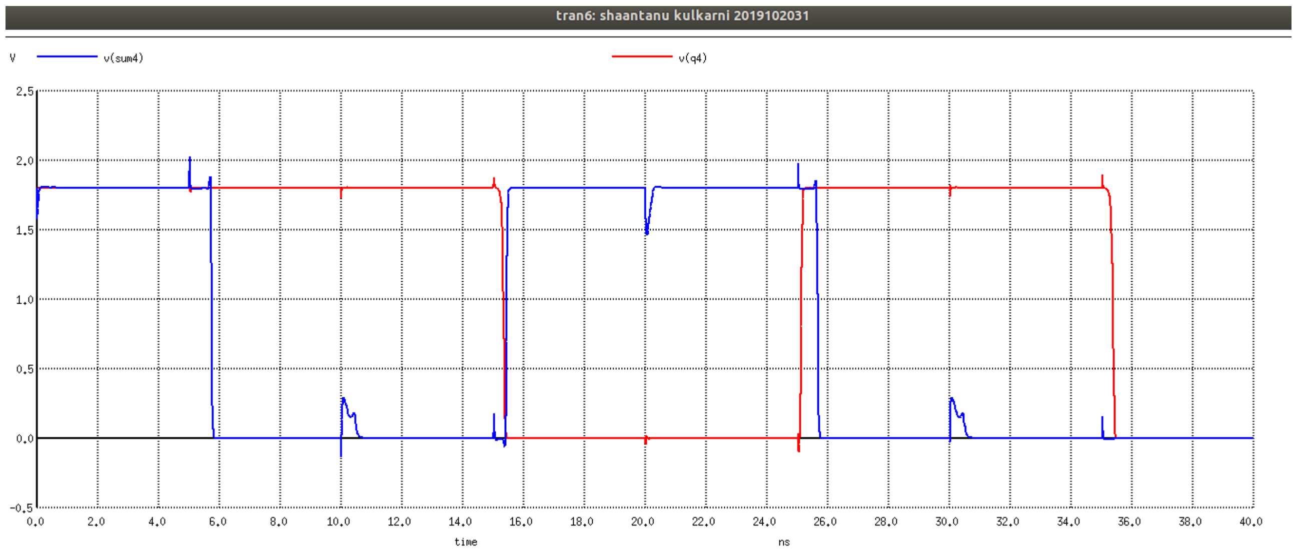
Sumbit 2



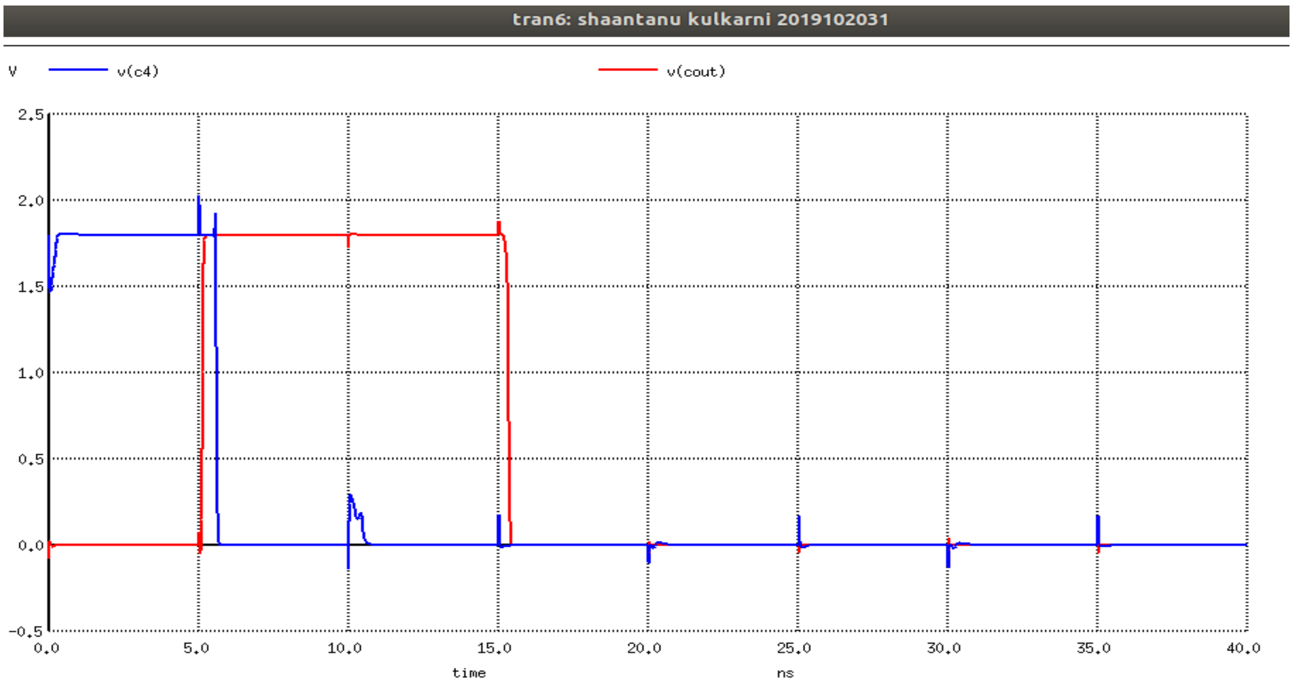
Sumbit 3



Sumbit4 (MSB)



Cout



Delays:

Sum bit	Tlh (ns)	Thl (ns)
S1	0.4164	0.4186
s2	0.4070	0.4352
s3	0.4375	0.4455
s4	0.4375	0.4776

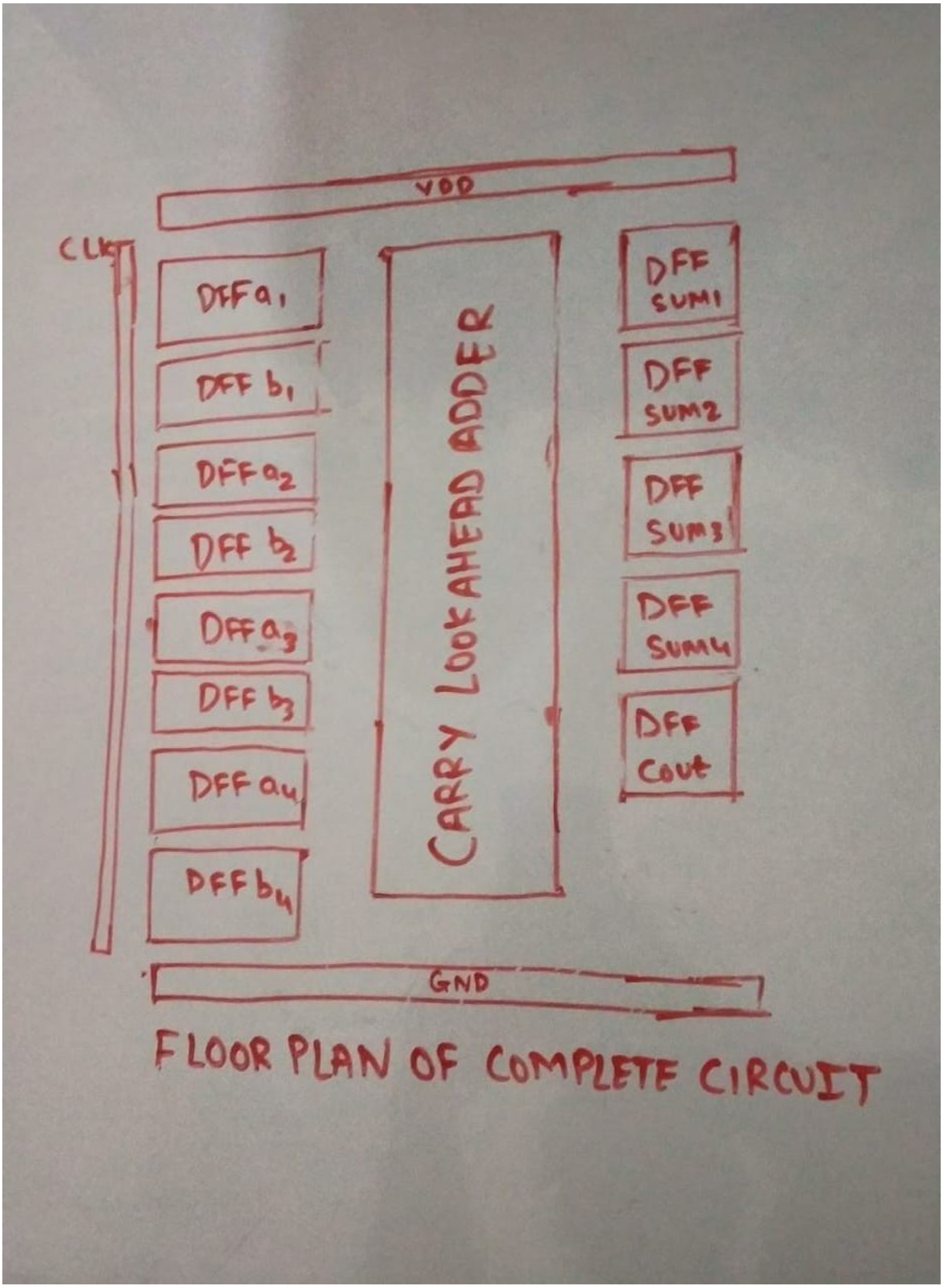
Thus the worst case delay is Thl for s4 ie 0.4776 ns . Thus maximum delay taken by any input , for the sum to be calculated is about 0.4776ns.

Maximum clock frequency = 1/minimum time period

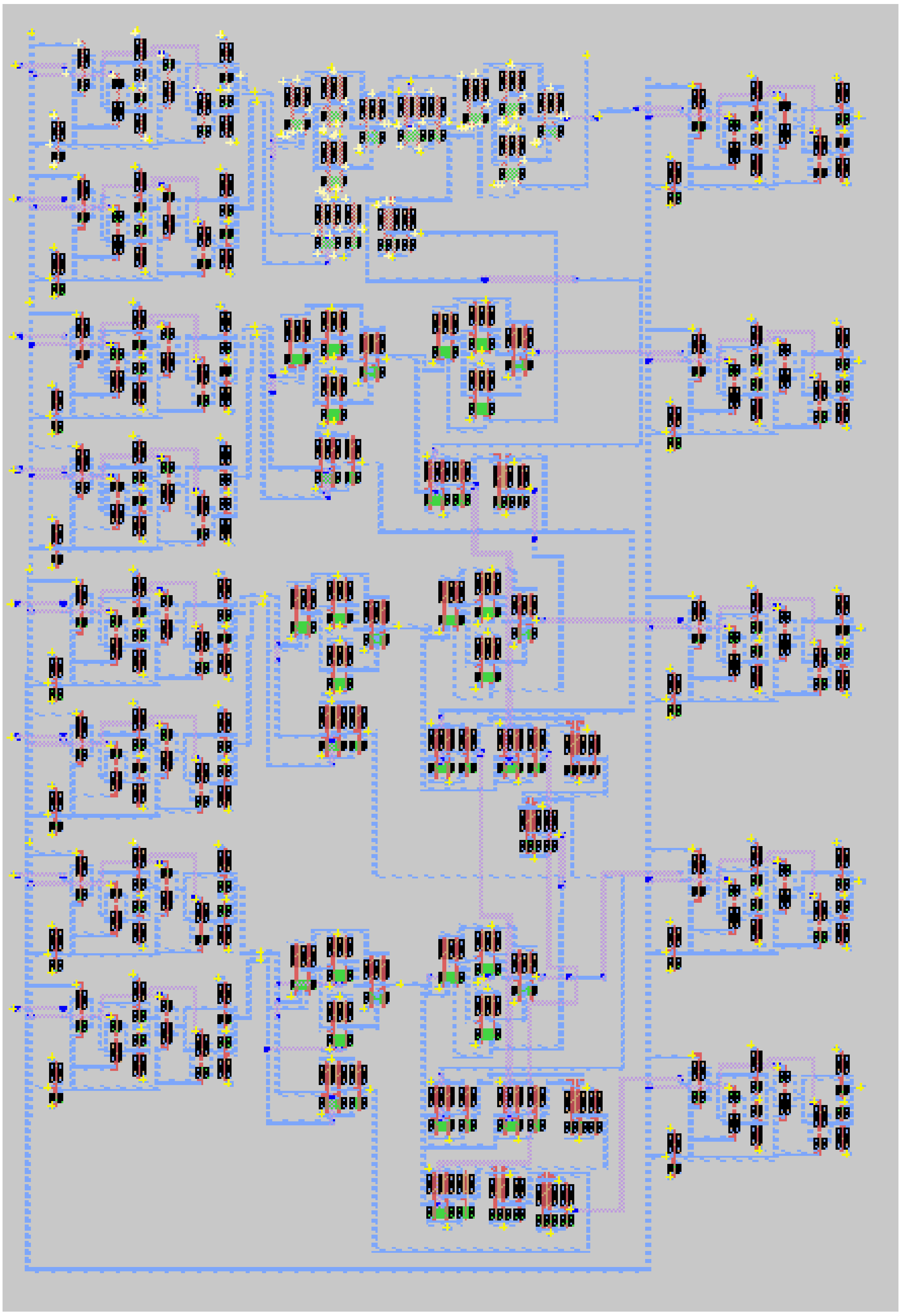
= 1/(delay of Dff + worst delay of CLA + hold time)

= 1/(0.44+0.4776+0.05) =1.03348 x10^{9} Hz

Floor Plan

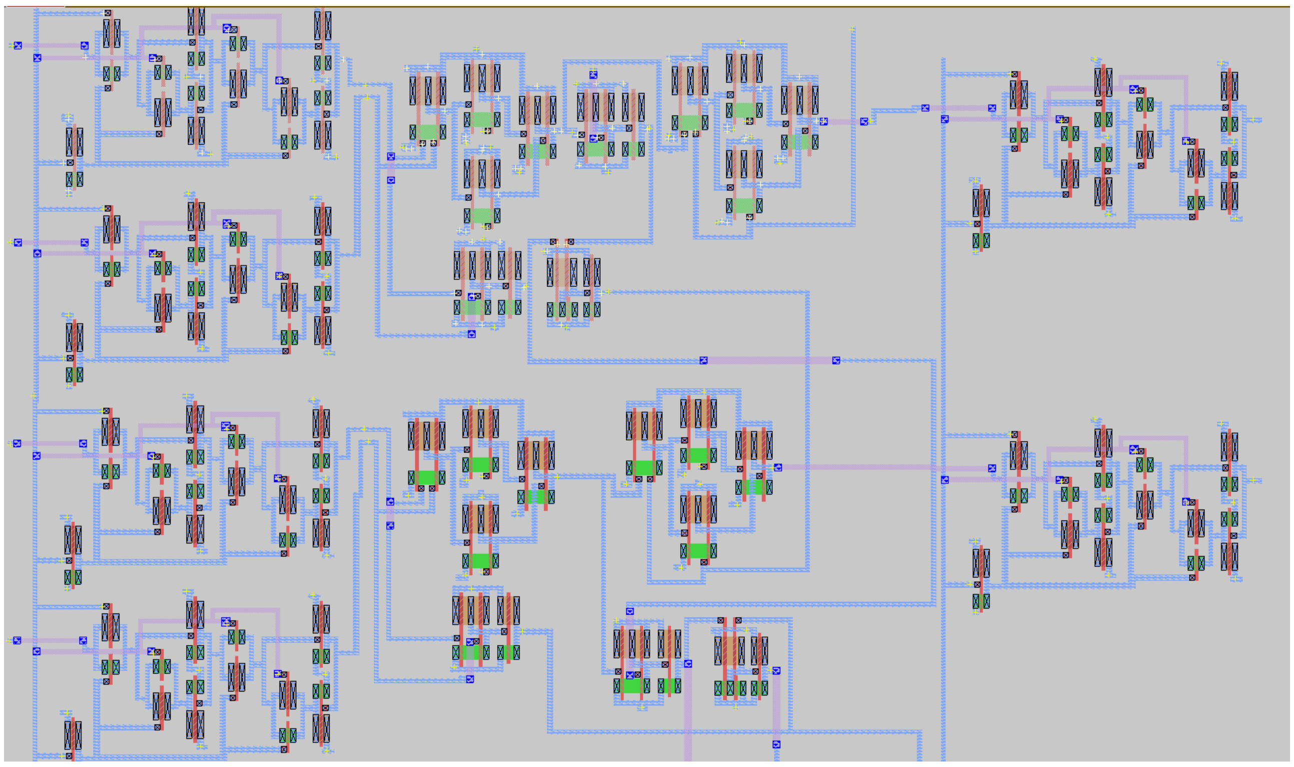

VDD
CLK
DFF a1
DFF b1
DFF a2
DFF b2
DFF a3
DFF b3
DFF a4
DFF b4
CARRY LOOK AHEAD ADDER
DFF SUM1
DFF SUM2
DFF SUM3
DFF SUM4
DFF Cout
GND
FLOOR PLAN OF COMPLETE CIRCUIT

Complete Layout



Some other screenshots of the complete layout

Zoomed in view of the layout for the first two bits. Here we can see 4 D flipflops on the left hand side and 2 D flipflops on the right hand sideZ

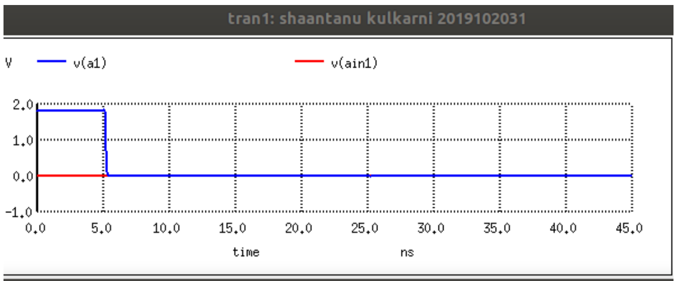


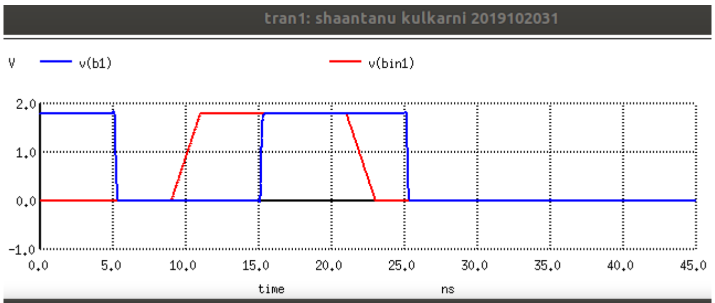
Simulation specific note:

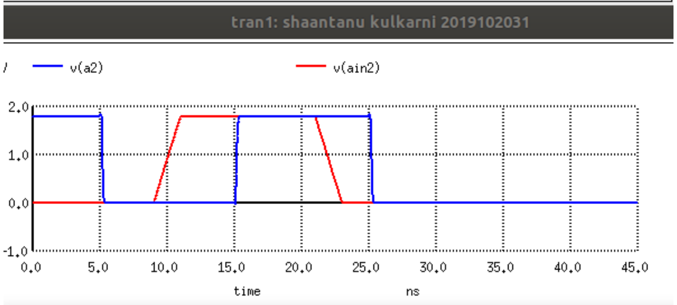
As explained in previous part about the choice of input test cases, for the simulation of this part also we have taken the same test cases.

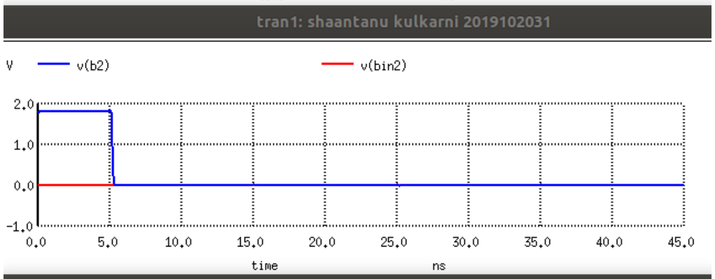
Simulations and Results:

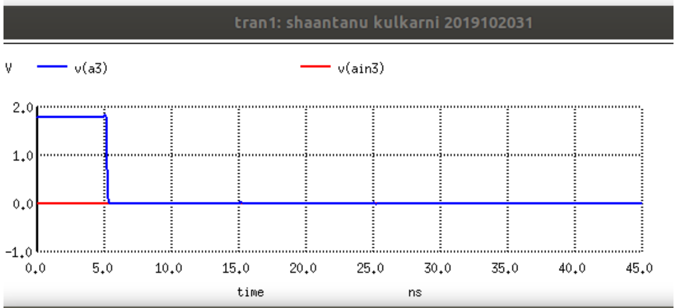
Inputs:

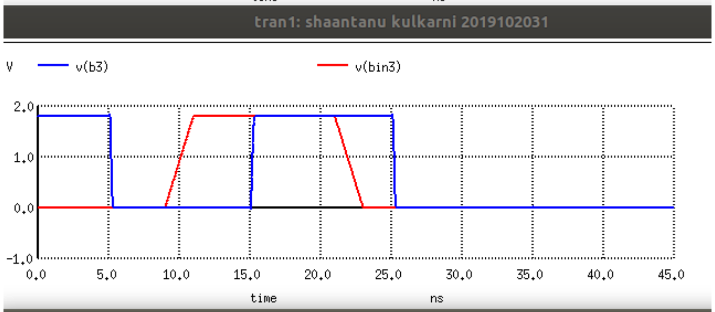


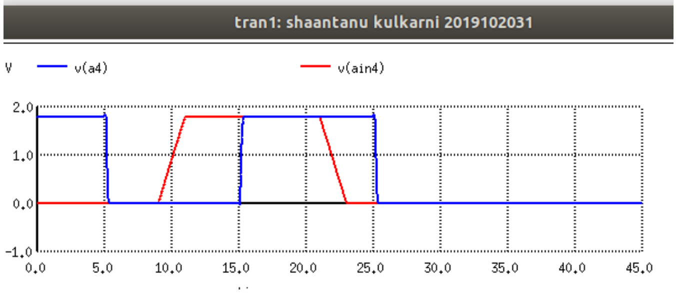


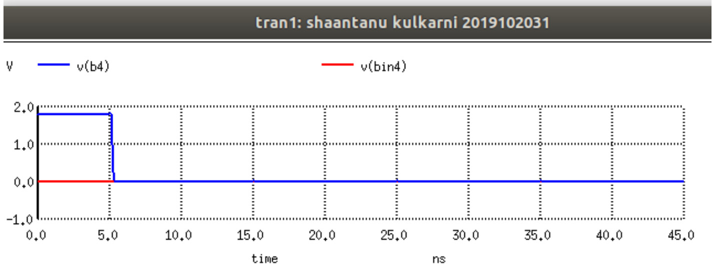




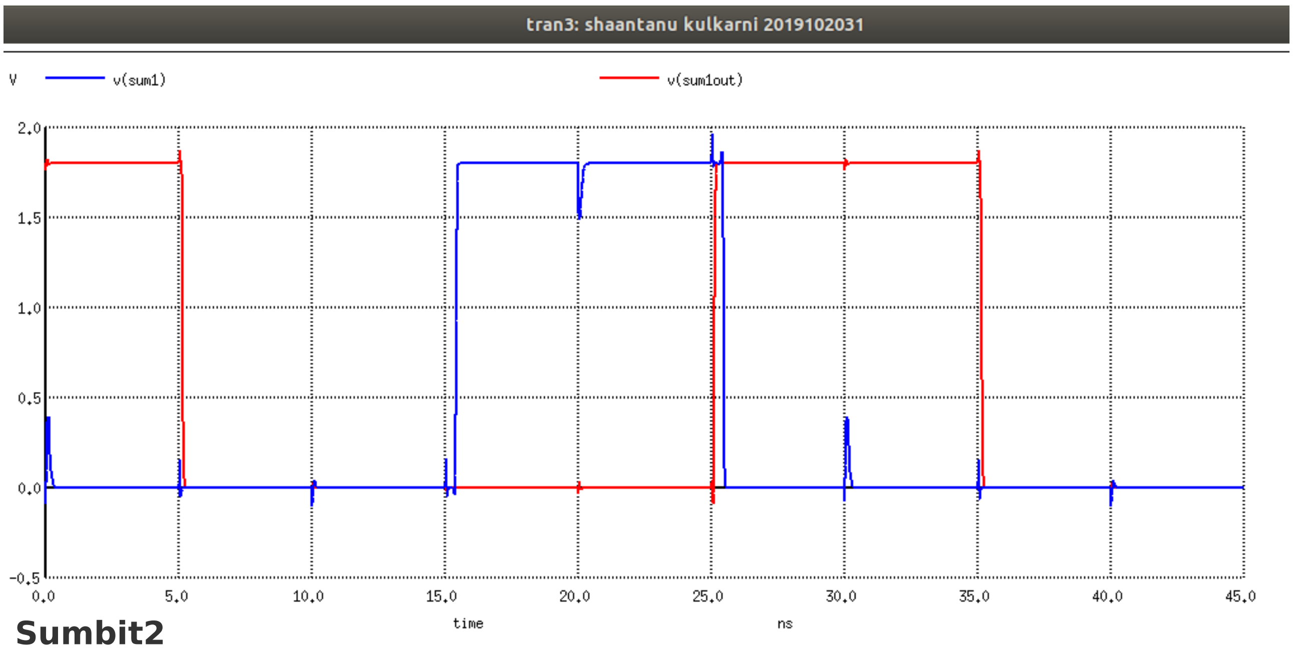




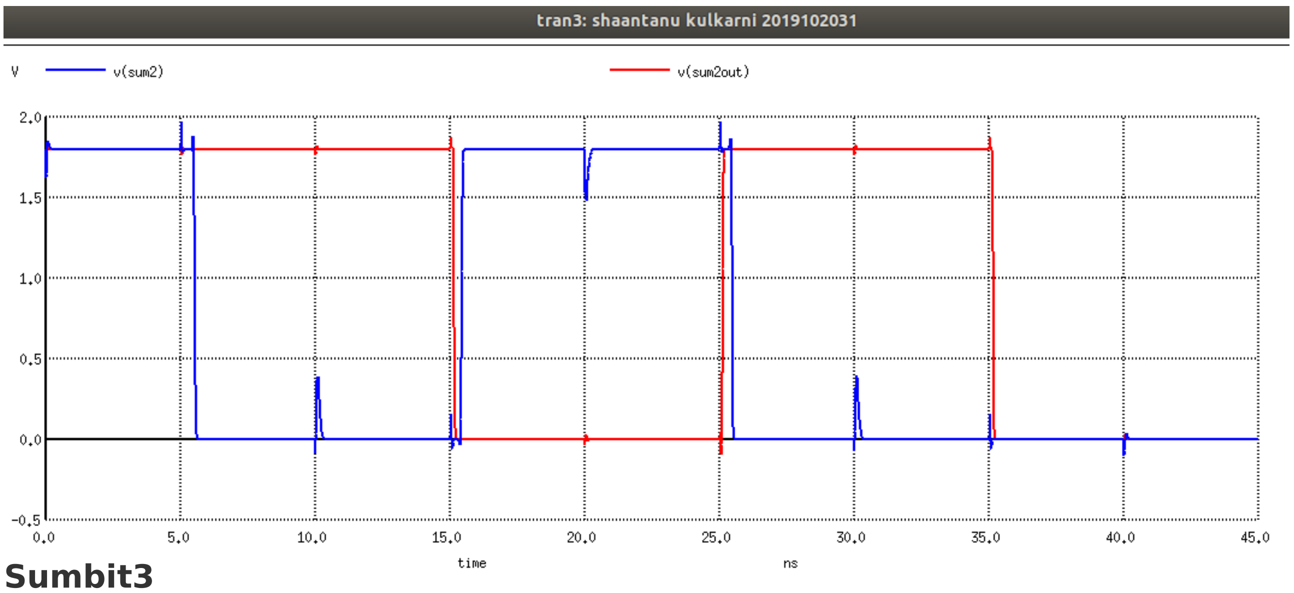




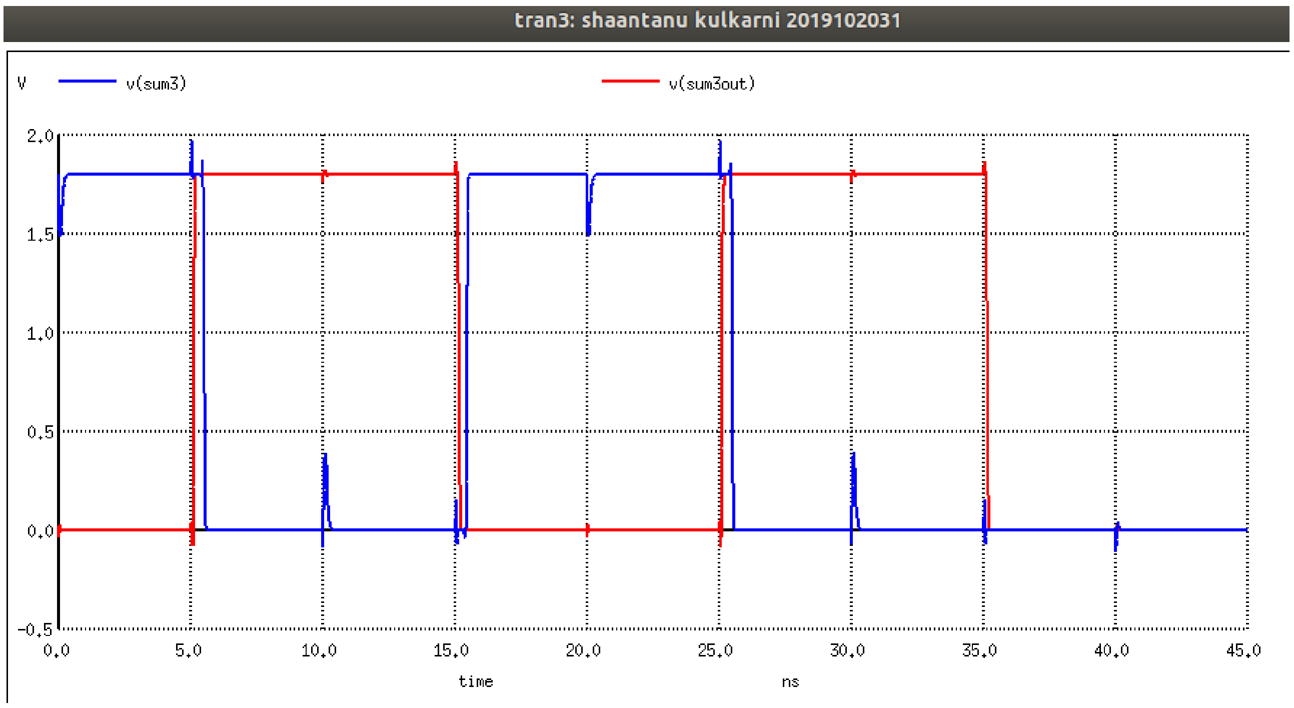
Sumbit1 (LSB)



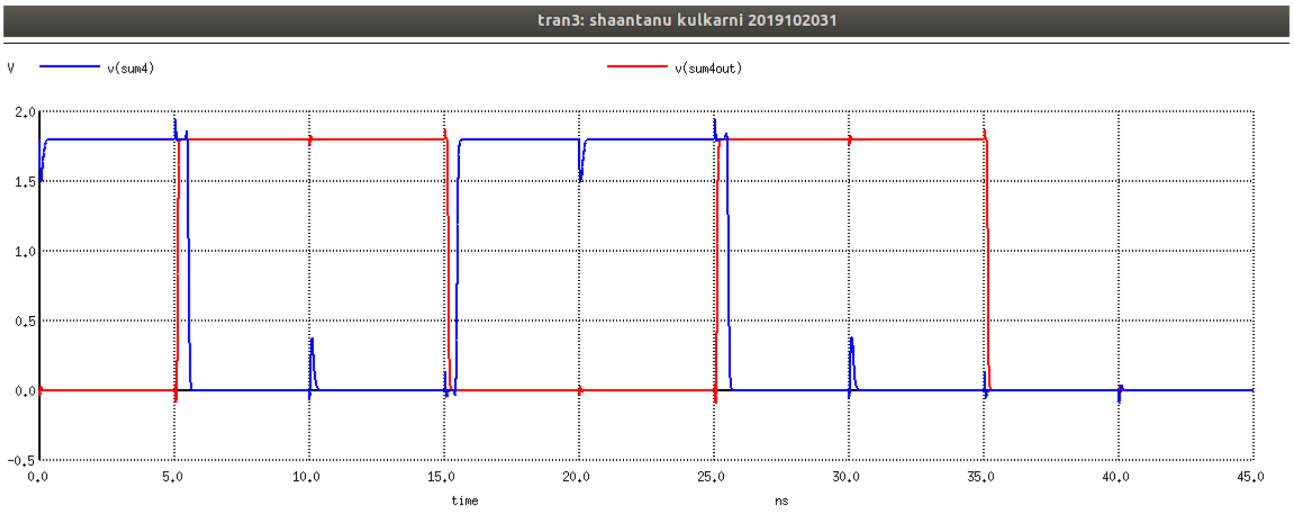
Sumbit2



Sumbit3



Sumbit4 (MSB)



Delays:

Sum bit	Tlh (ns)	Thl (ns)
S1	0.4478	0.5219
s2	0.4627	0.5319
s3	0.4861	0.5597
s4	0.5320	0.5670

Thus the worst case delay is Thl for s4 ie 0.567 ns . Thus maximum delay taken by any input , for the sum to be calculated is about 0.567ns.

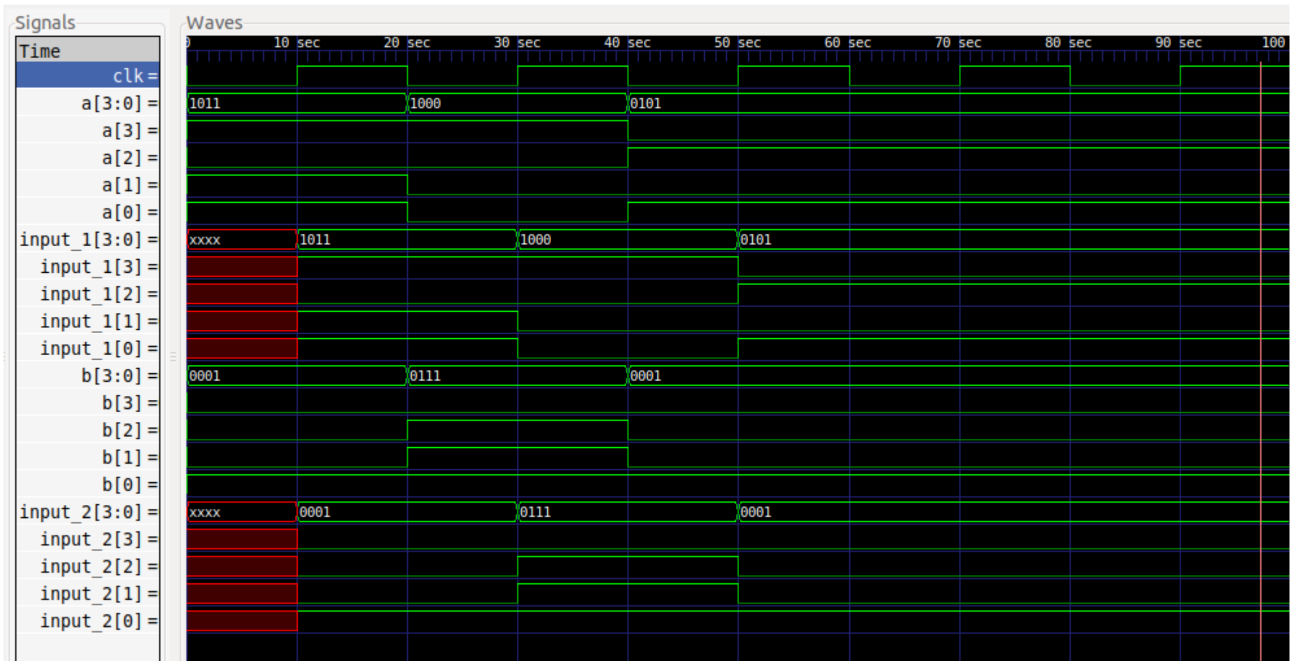
Maximum clock frequency = minimum time period

= delay of Dff + worst delay of CLA + hold time

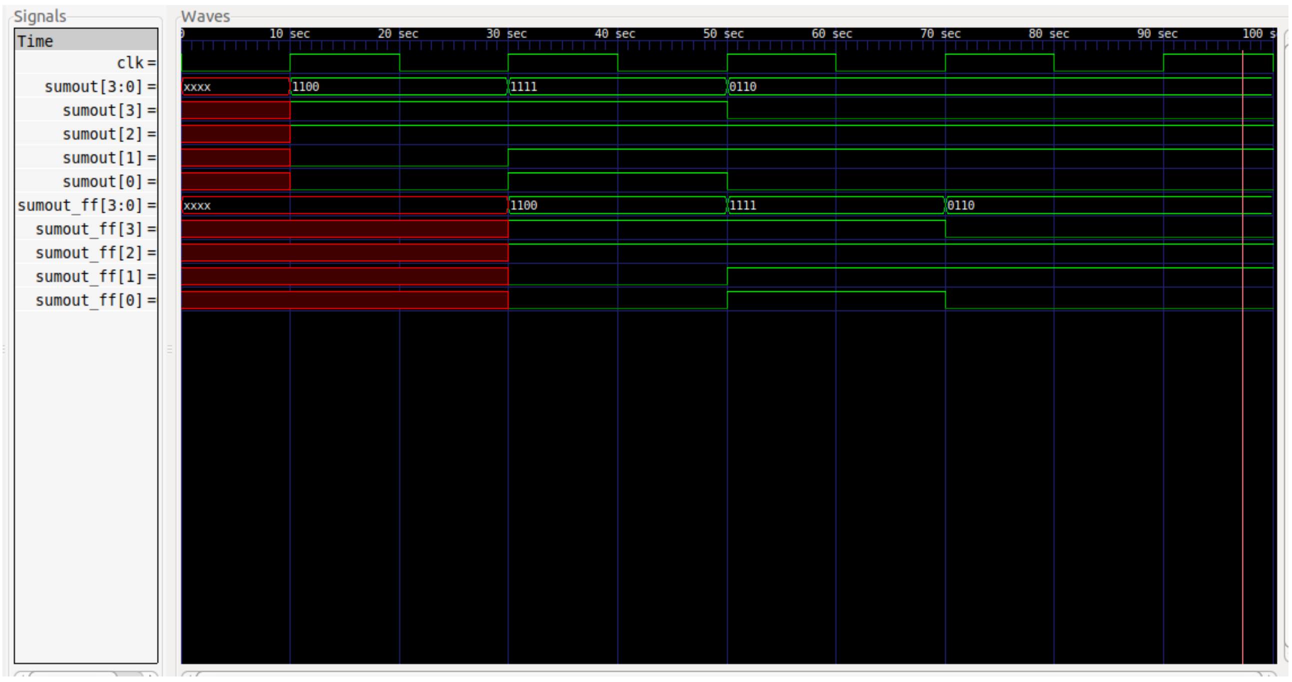
= 0.1095+0.567+0.035 = 1.4054 x 10^9 Hz

Verilog Simulations:

Inputs

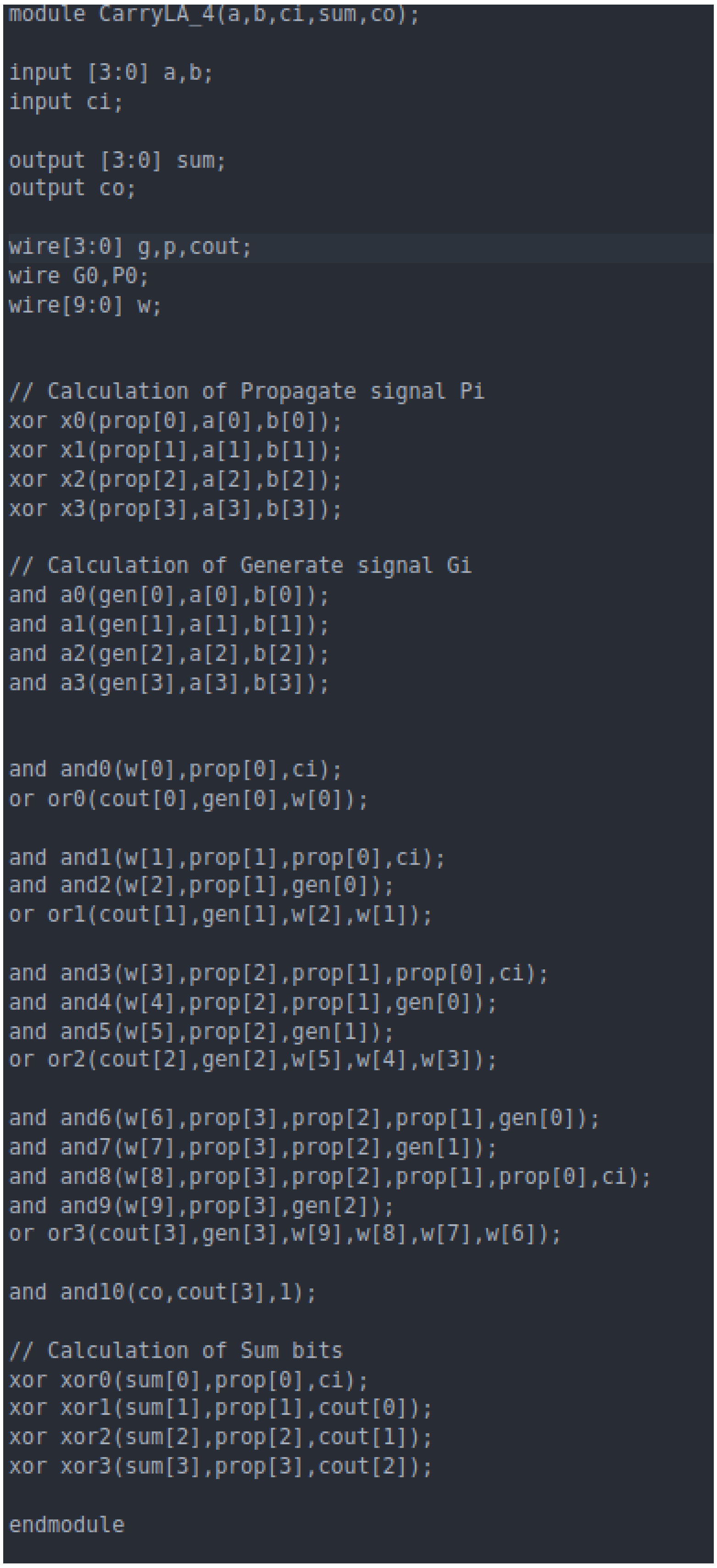


Output sums and output of flipflops:



- We can see that the outputs of flipflops are set after one clock cycle delay as expected
- The positive edge behavior of the D flipflop can be seen clearly
- The outputs of sum are as expected which supports the fact that the functionality of the circuit is correct.

Verilog Structural description



```
module CarryLA_4(a,b,ci,sum,co);

input [3:0] a,b;
input ci;

output [3:0] sum;
output co;

wire[3:0] g,p,cout;
wire G0,P0;
wire[9:0] w;


// Calculation of Propagate signal Pi
xor x0(prop[0],a[0],b[0]);
xor x1(prop[1],a[1],b[1]);
xor x2(prop[2],a[2],b[2]);
xor x3(prop[3],a[3],b[3]);

// Calculation of Generate signal Gi
and a0(gen[0],a[0],b[0]);
and a1(gen[1],a[1],b[1]);
and a2(gen[2],a[2],b[2]);
and a3(gen[3],a[3],b[3]);


and and0(w[0],prop[0],ci);
or or0(cout[0],gen[0],w[0]);

and and1(w[1],prop[1],prop[0],ci);
and and2(w[2],prop[1],gen[0]);
or or1(cout[1],gen[1],w[2],w[1]);

and and3(w[3],prop[2],prop[1],prop[0],ci);
and and4(w[4],prop[2],prop[1],gen[0]);
and and5(w[5],prop[2],gen[1]);
or or2(cout[2],gen[2],w[5],w[4],w[3]);

and and6(w[6],prop[3],prop[2],prop[1],gen[0]);
and and7(w[7],prop[3],prop[2],gen[1]);
and and8(w[8],prop[3],prop[2],prop[1],prop[0],ci);
and and9(w[9],prop[3],gen[2]);
or or3(cout[3],gen[3],w[9],w[8],w[7],w[6]);

and and10(co,cout[3],1);

// Calculation of Sum bits
xor xor0(sum[0],prop[0],ci);
xor xor1(sum[1],prop[1],cout[0]);
xor xor2(sum[2],prop[2],cout[1]);
xor xor3(sum[3],prop[3],cout[2]);

endmodule
```