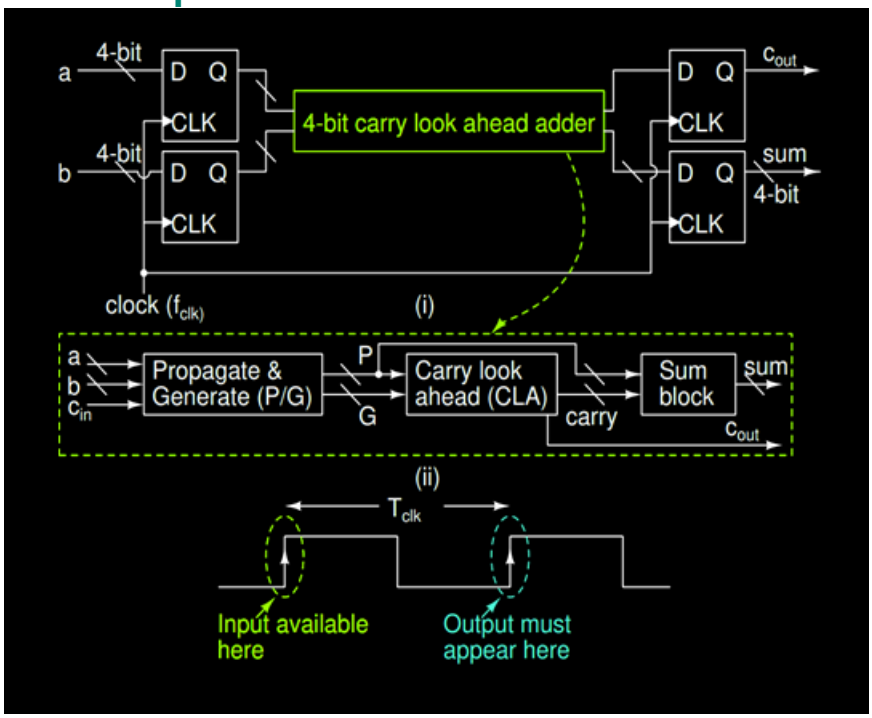
VLSI Design Project

4-Bit Carry Look Ahead Adder

Bhaiya Vaibhaw Kumar
2019112021

We need to build a 4-Bit Carry Ahead Adder. We must take input on the clock's positive edge and output on the clock's next positive side. A=1100 and b=0101 are the test cases used to validate all of the functionalities.

1. ANS ONE- Proposed Structure for the Adder

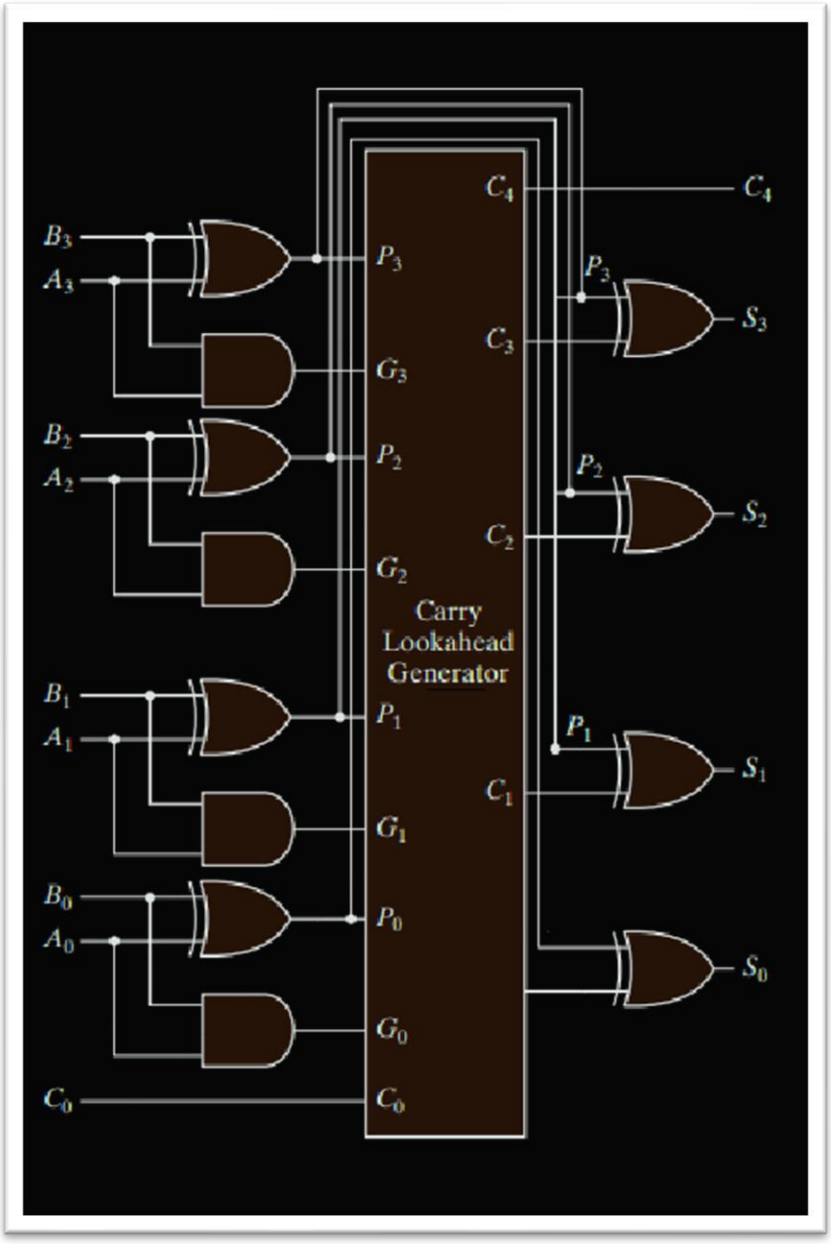


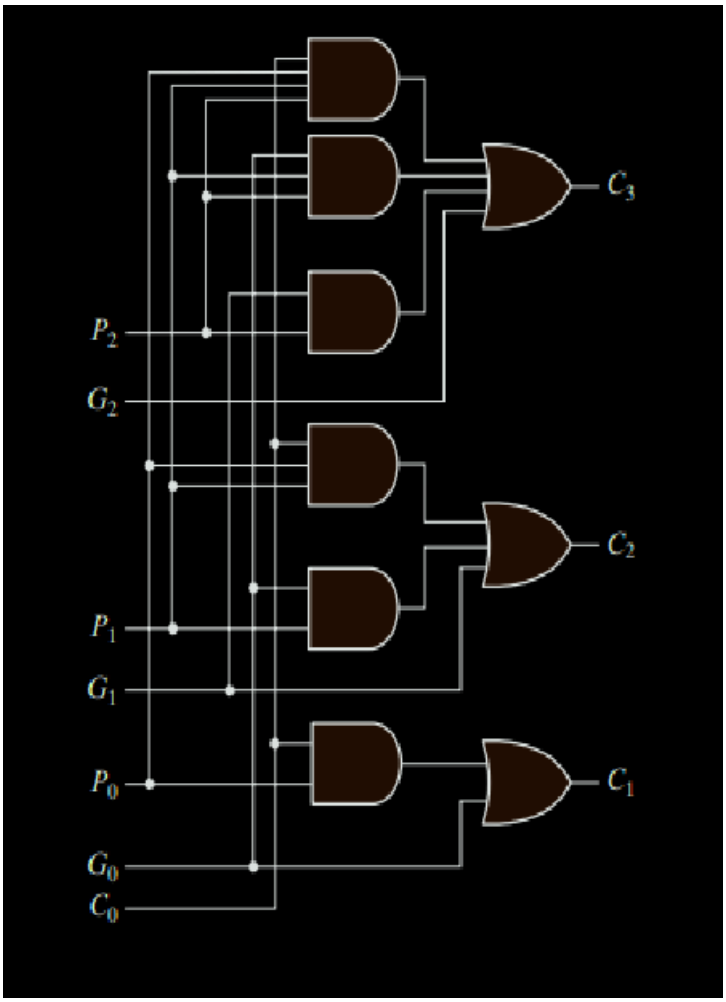
We will provide feedback on the clock's first positive edge, and the original flip-flop will move it to the adder on the clock's negative edge. The adder adds all inputs before proceeding to the next flip-flop. The final flip-flop outputs the number on the next positive edge of the clock. As a result, this is the best system for implementing the given scenario.

2. ANS TWO- Design Details

The adder looks as follows:

In this, Carry Look Ahead Generator looks as follows:





Explanation of the structure:

Let us assume we have two four-bit binary inputs, A3A2A1A0 and B3B2B1B0, respectively. We begin by defining two binary variables:

$P_i = A_i \oplus B_i$; $G_i = A_i B_i$;

Since it decides if a carry into stage, I can propagate into stage I + 1, Pi is known as carry propagate. When both Ai and Bi are 1, independent of the input carry Ci, Gi is defined as carry generate and generates a carry of 1.

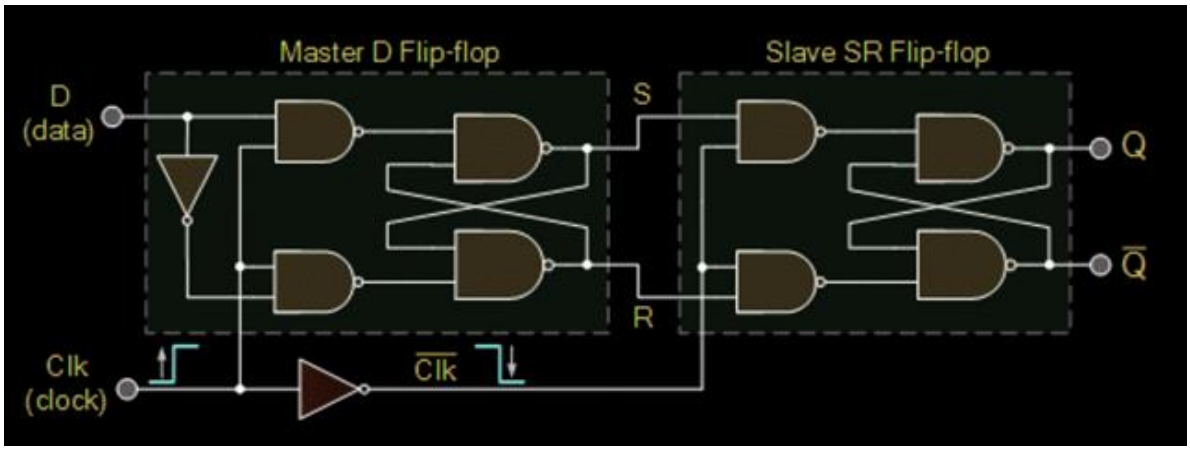
Both are then used to measure the number and carry of each bit.

Now, we will presume that there has not been any previous carry for the 0th bit, so C0 = 0.

Sum is calculated as :$S_i = P_i \oplus C_i$

Carry is calculated as :$C_{i+1} = G_i + P_i\ C_i$

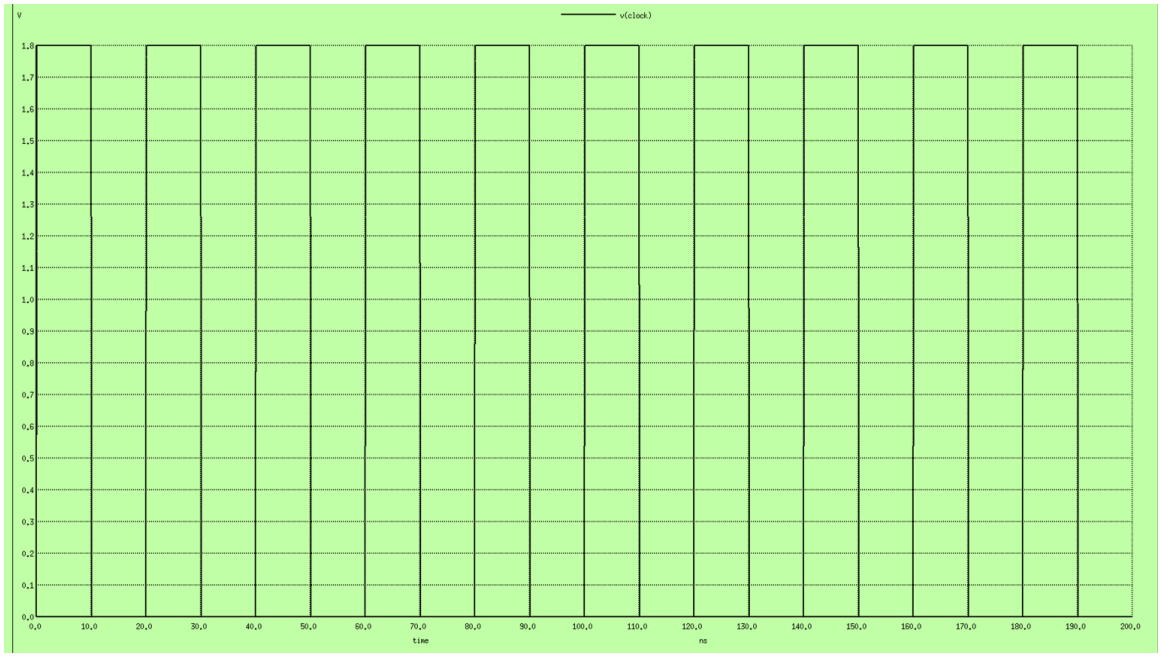
And for D-flip-flop, we are using NAND gates, and the design is as follows:



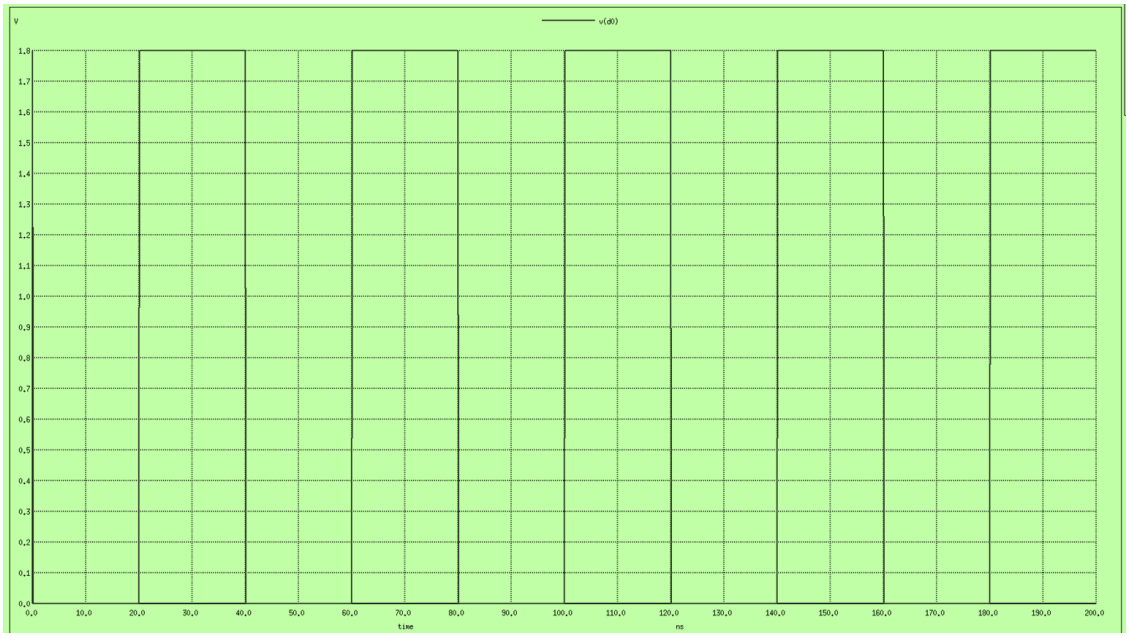
for mosfet, Wp=20*LAMBDA and Wn=10*LAMBDA.

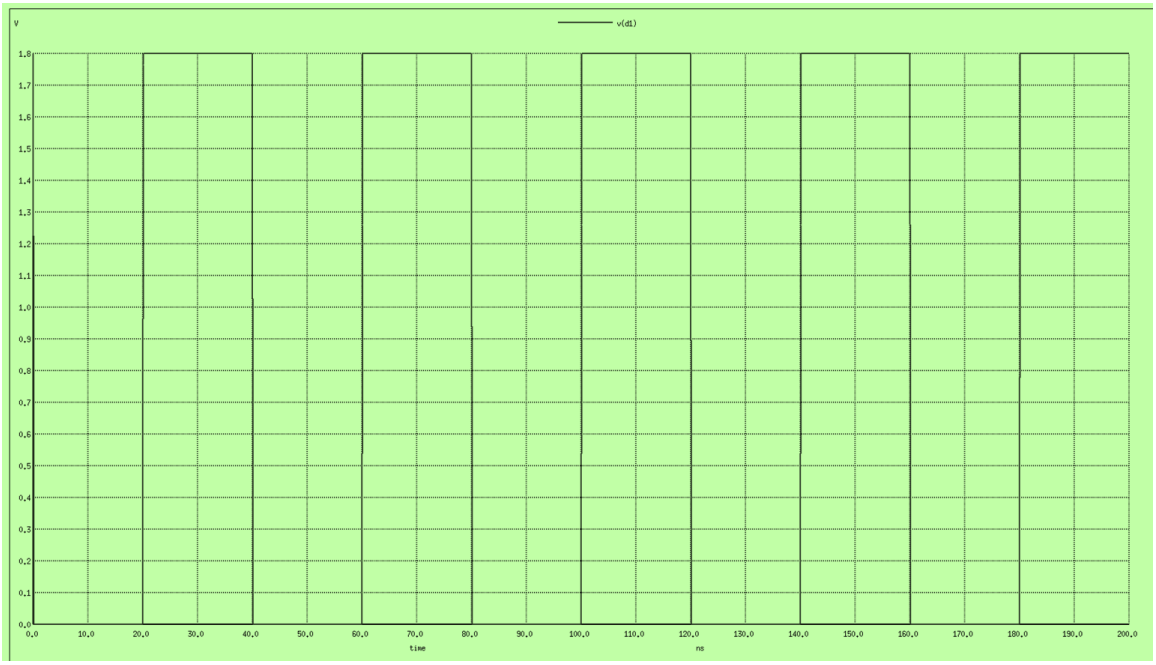
3. ANS THREE- Functions of each block:

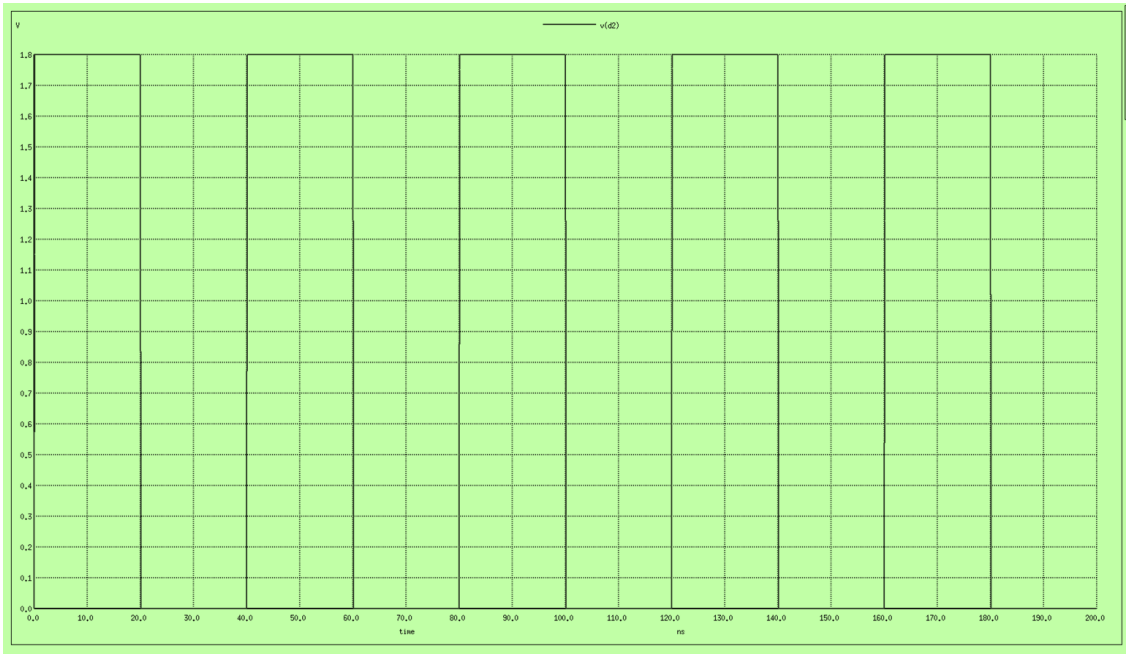
1. First, let us look at the D-flip-features. flop's Let the inputs be d3d2d1d0 and e3e2e1e0, and the outputs be a3a2a1a0 and b3b2b1b0, respectively. We must verify each bit individually according to the specified clock, which is as follows:

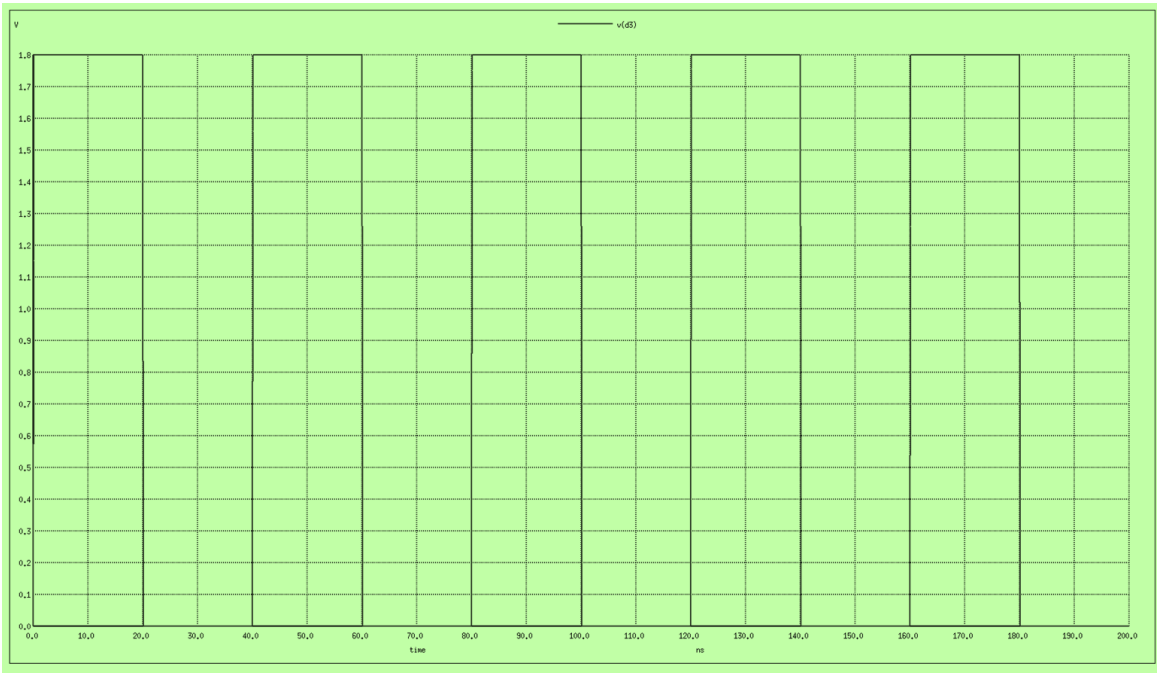


Now, for d3d2d1d0, plots are as follows:

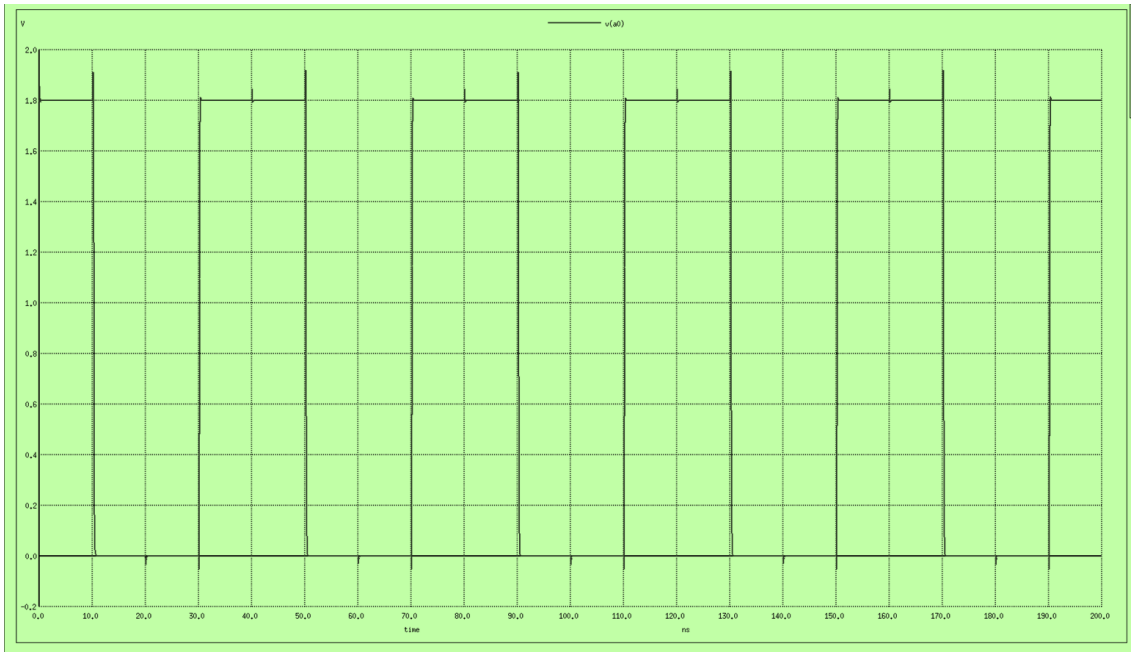


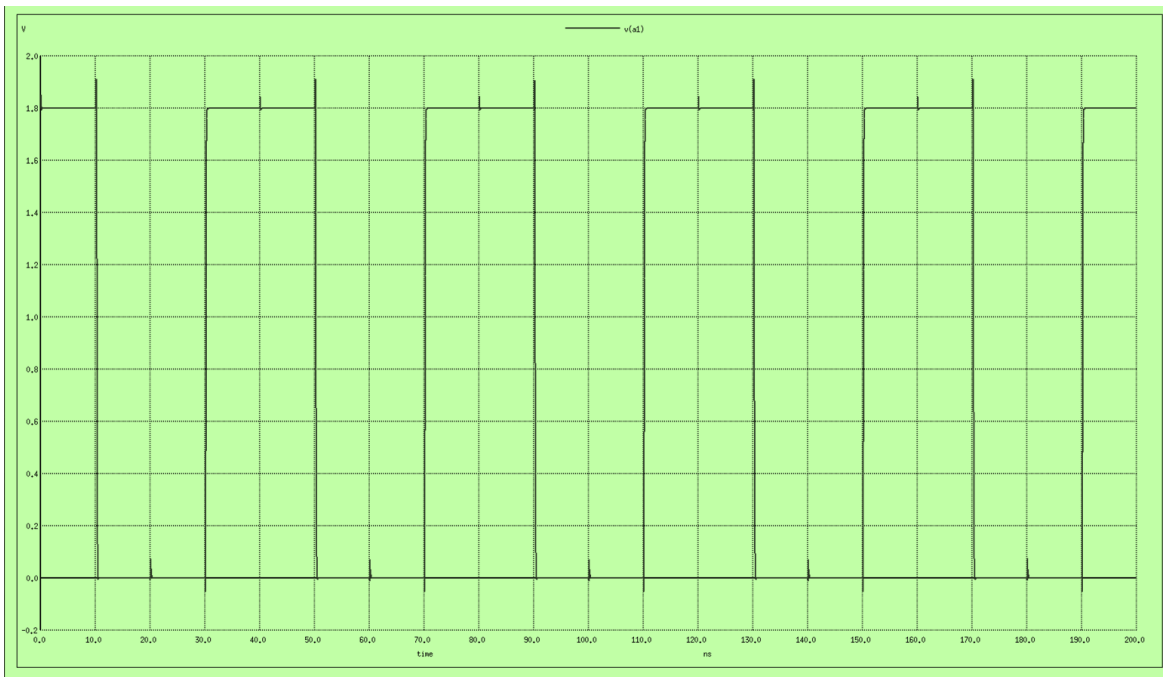

v(d1)
V
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns

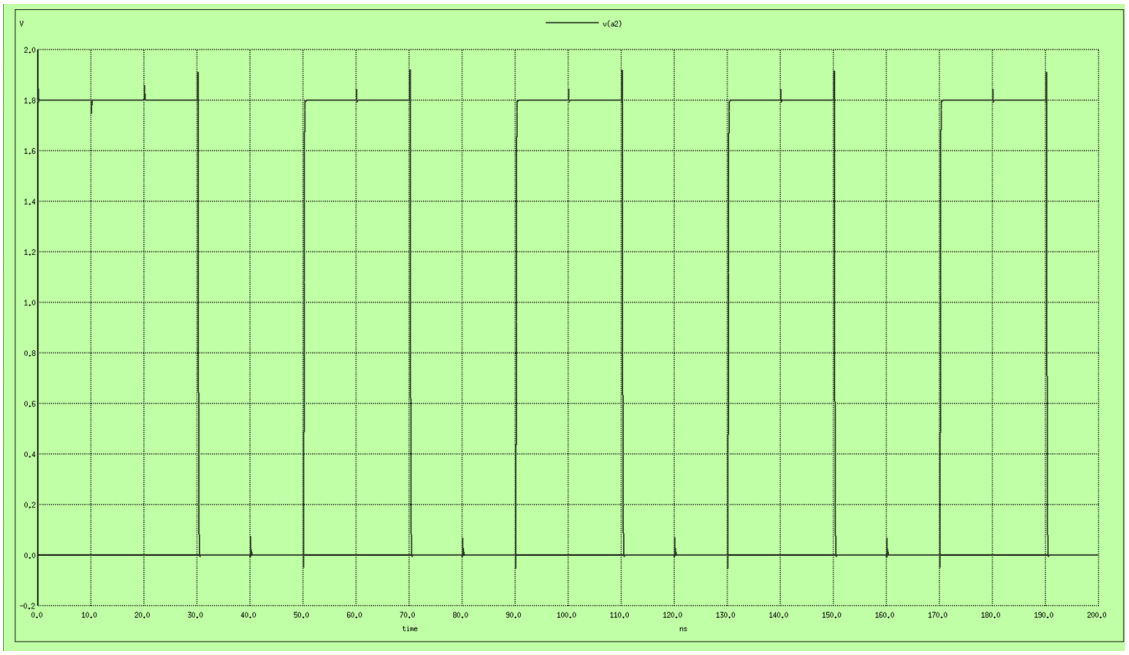

v(d2)
V
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns

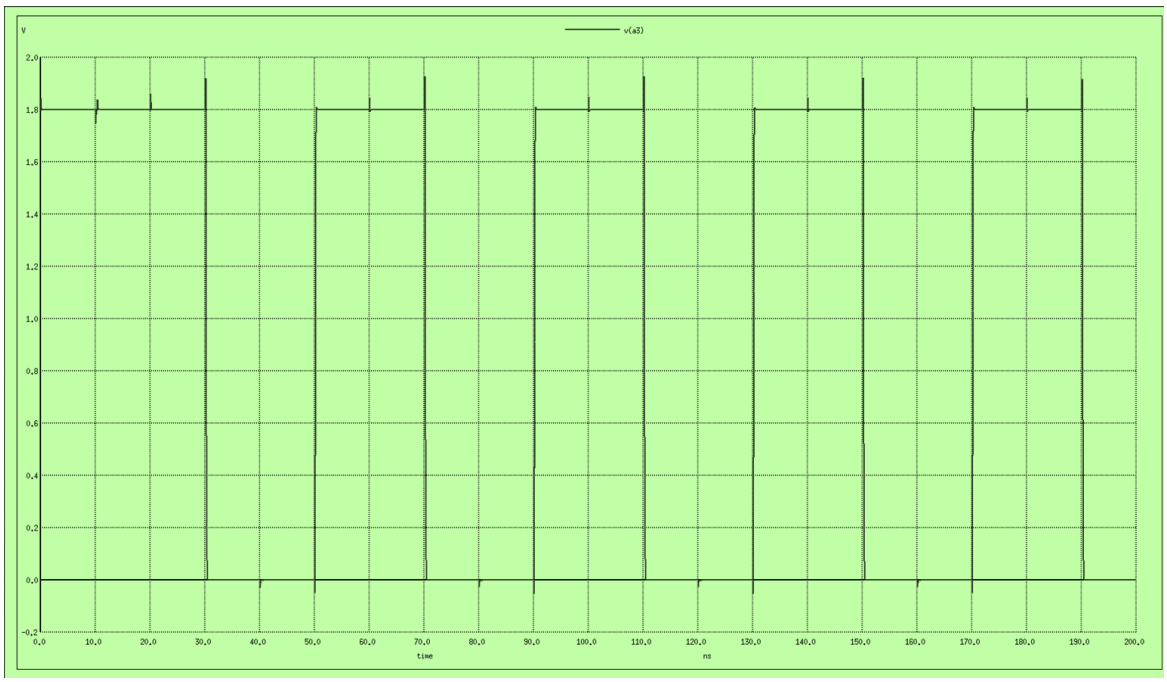


Now, the output a3a2a1a0 looks as follows:

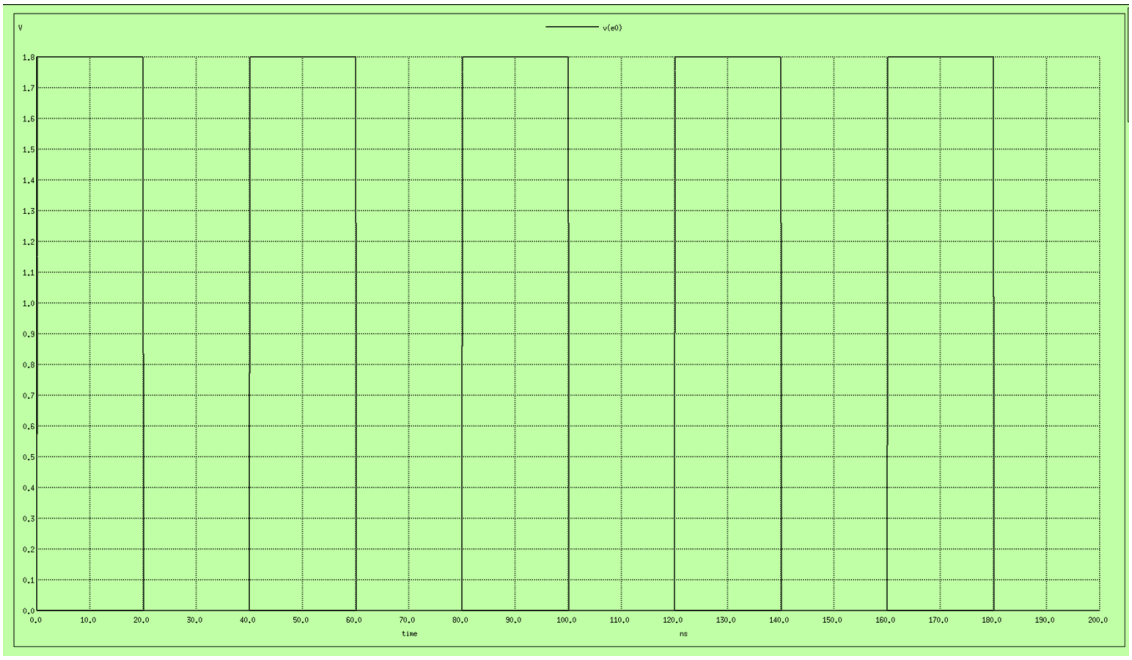


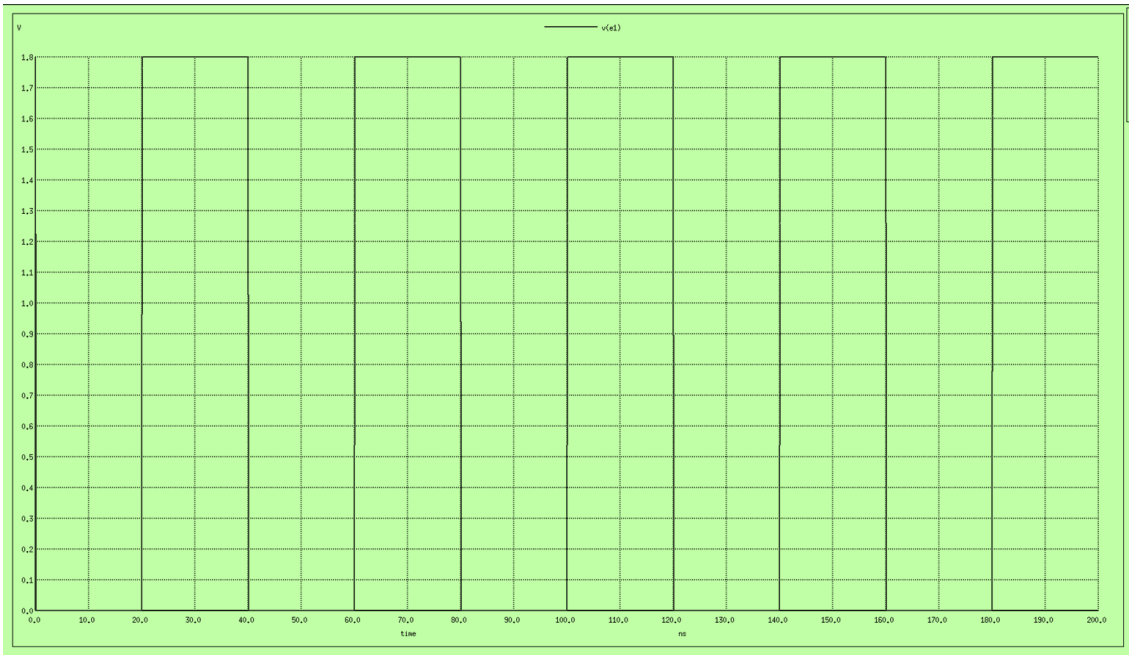

v(a1)
V
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns

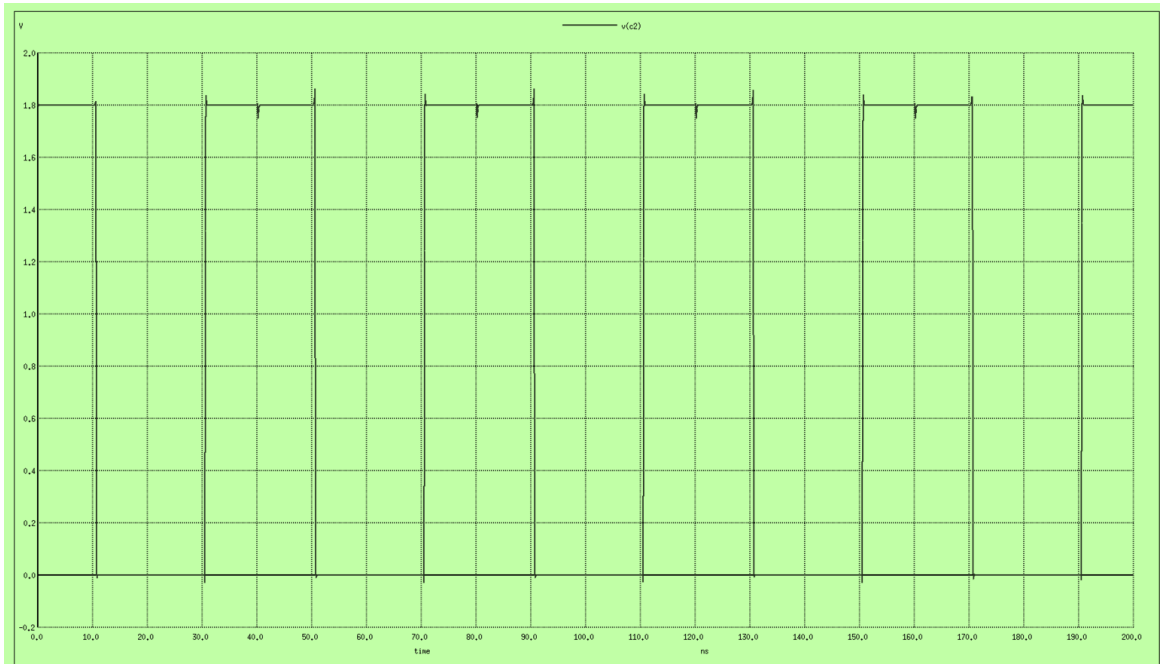

v(a2)
V
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns

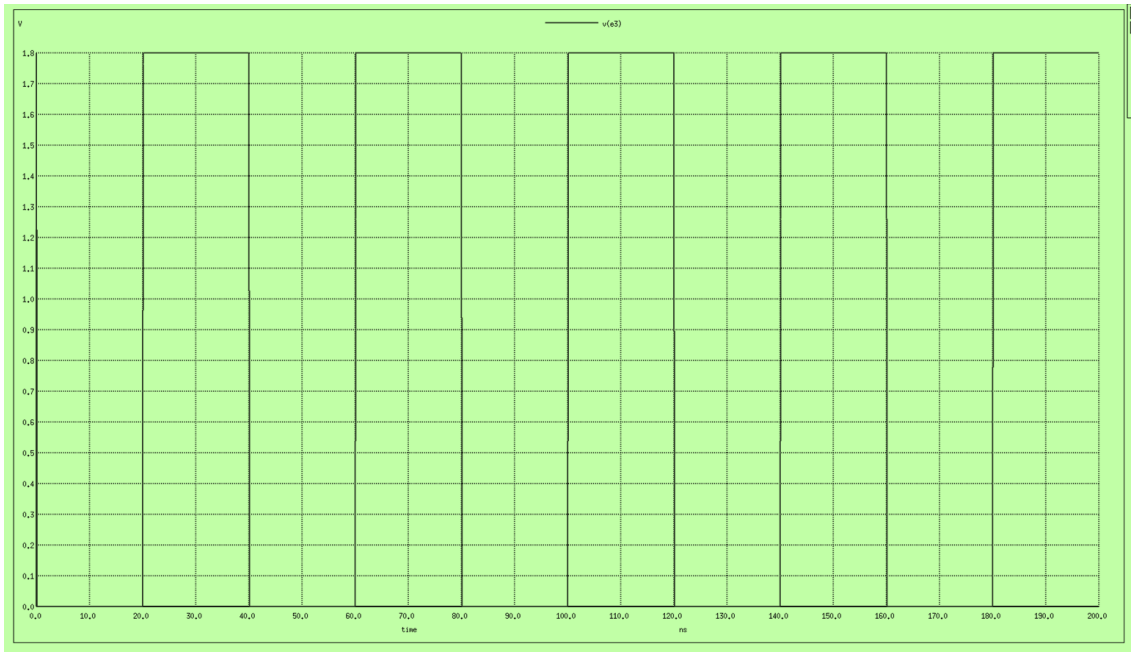


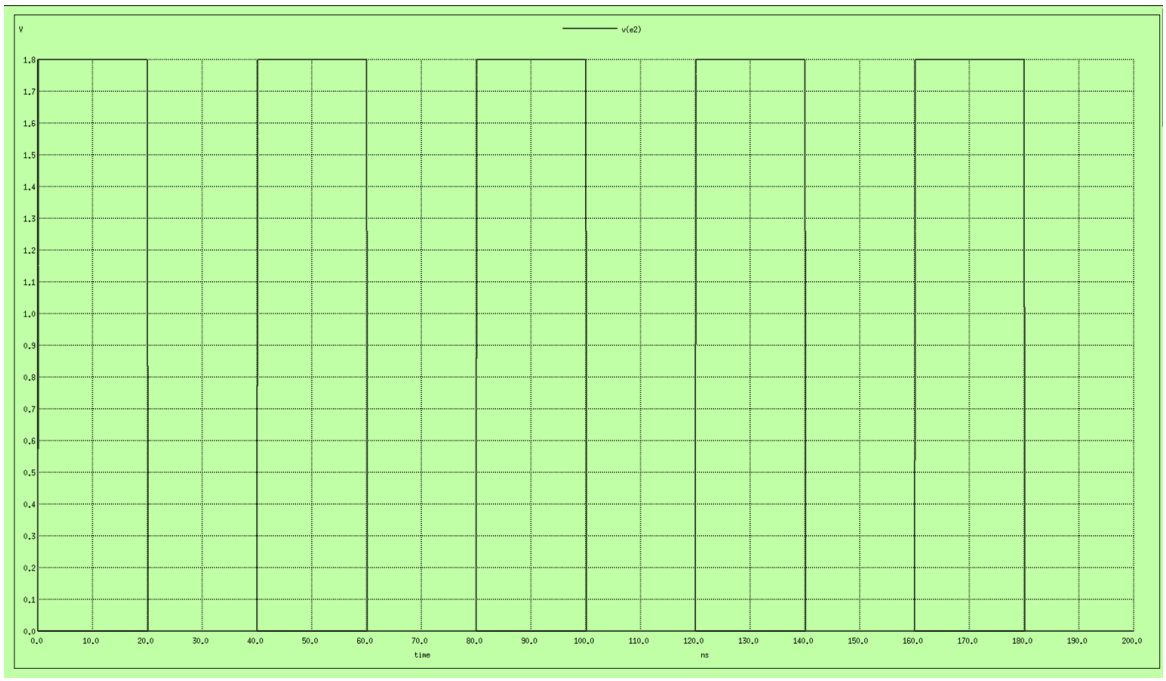
We can see that the processing is still delayed by 10ns. It's because the input enters the flip-flop at the positive edge of the clock and exits at the negative edge of the clock, a difference of 10ns.
Similarly, now we will check for e3e2e1e0:



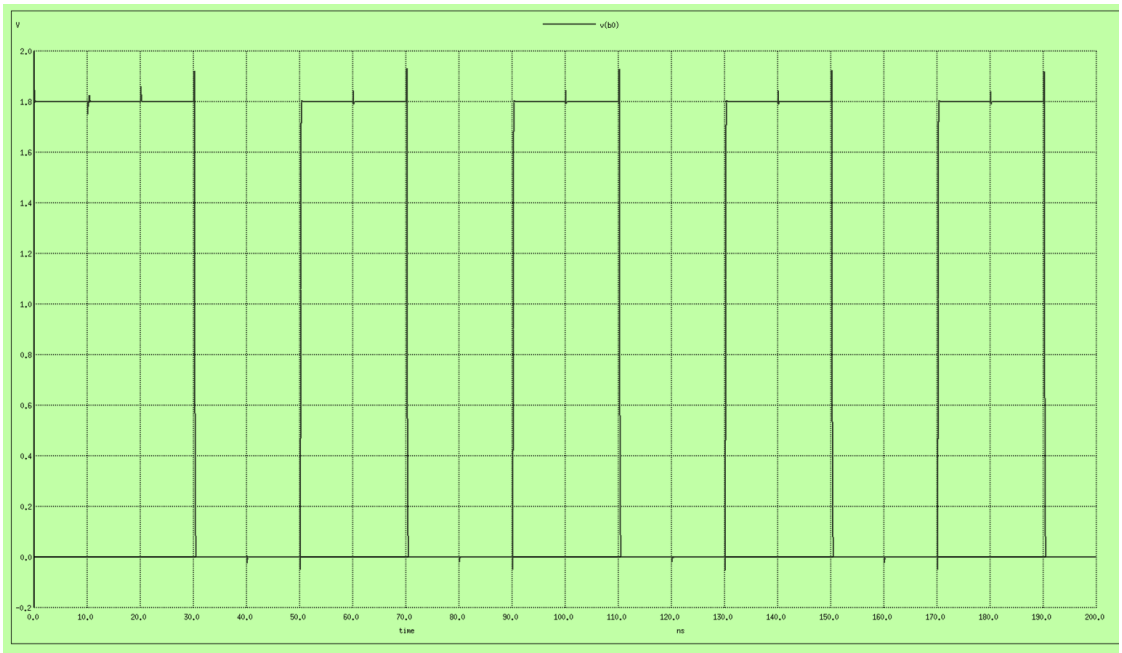

v(e1)
V
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
100.0
110.0
120.0
130.0
140.0
150.0
160.0
170.0
180.0
190.0
200.0
time
ns

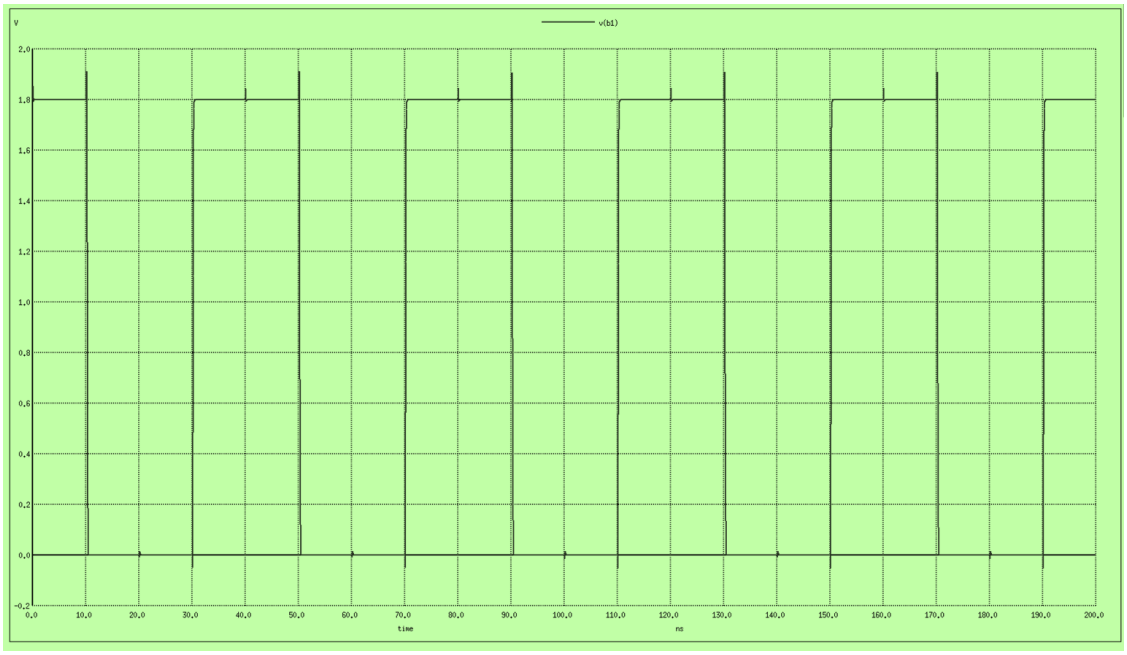

v(c2)
V
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns

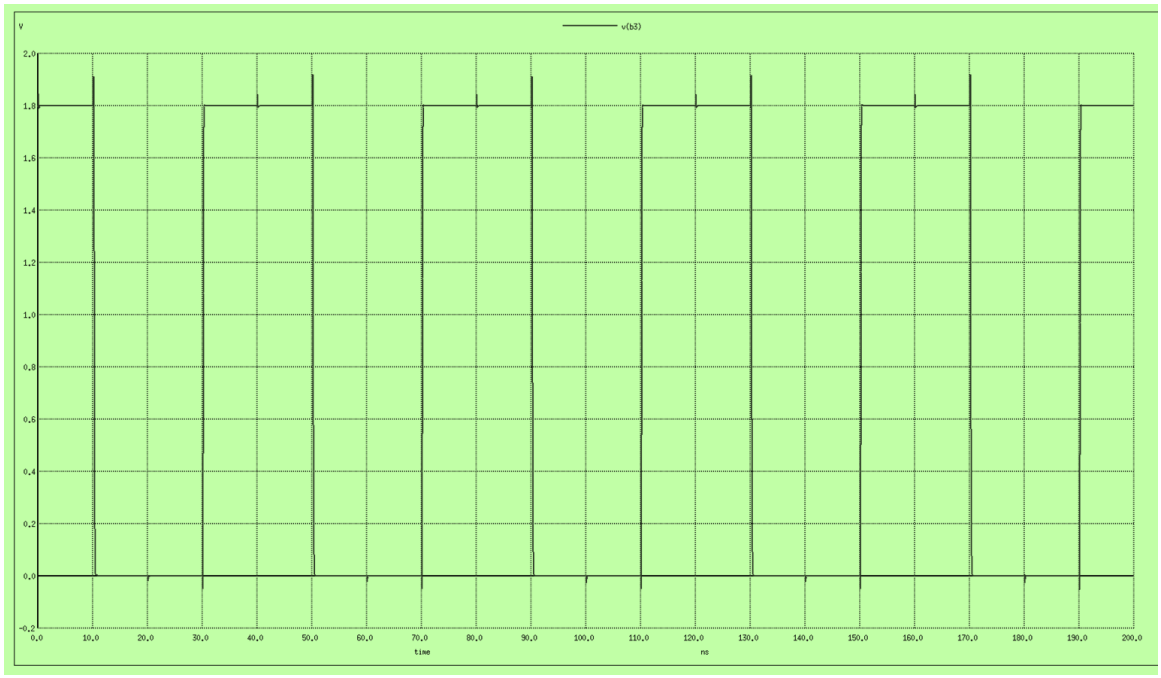

v(e3)
V
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns

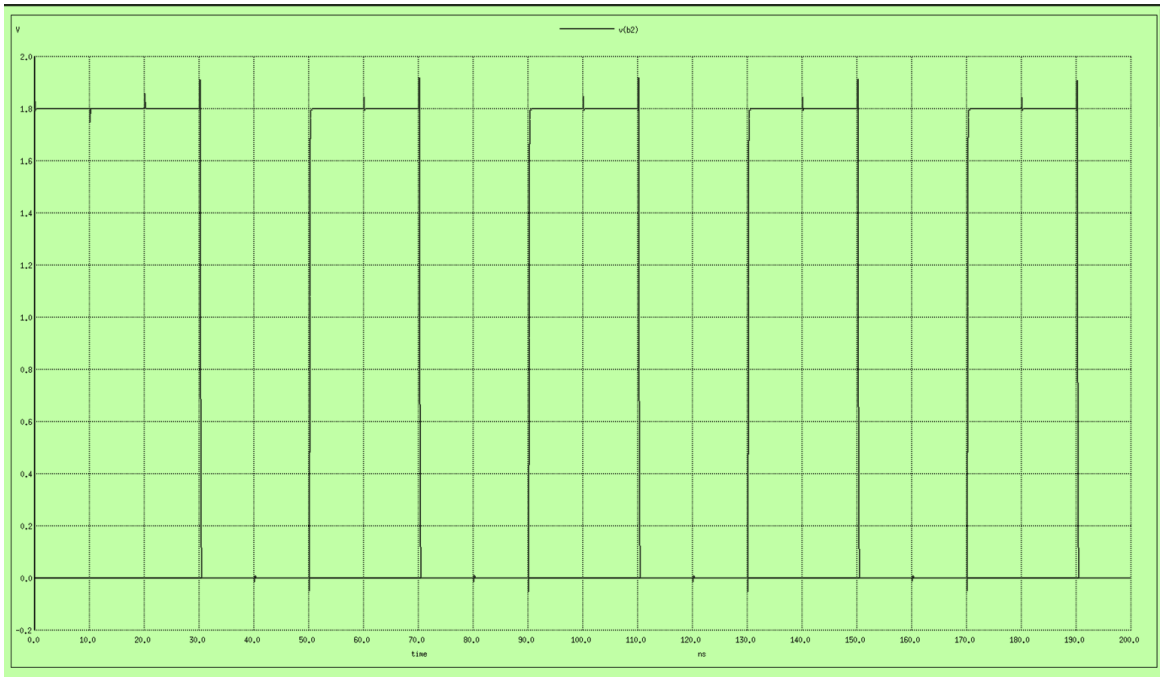


For this, the output b3b2b1b0 looks as follows:




v
v(b1)
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
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
170.0
180.0
190.0
200.0
time
ns

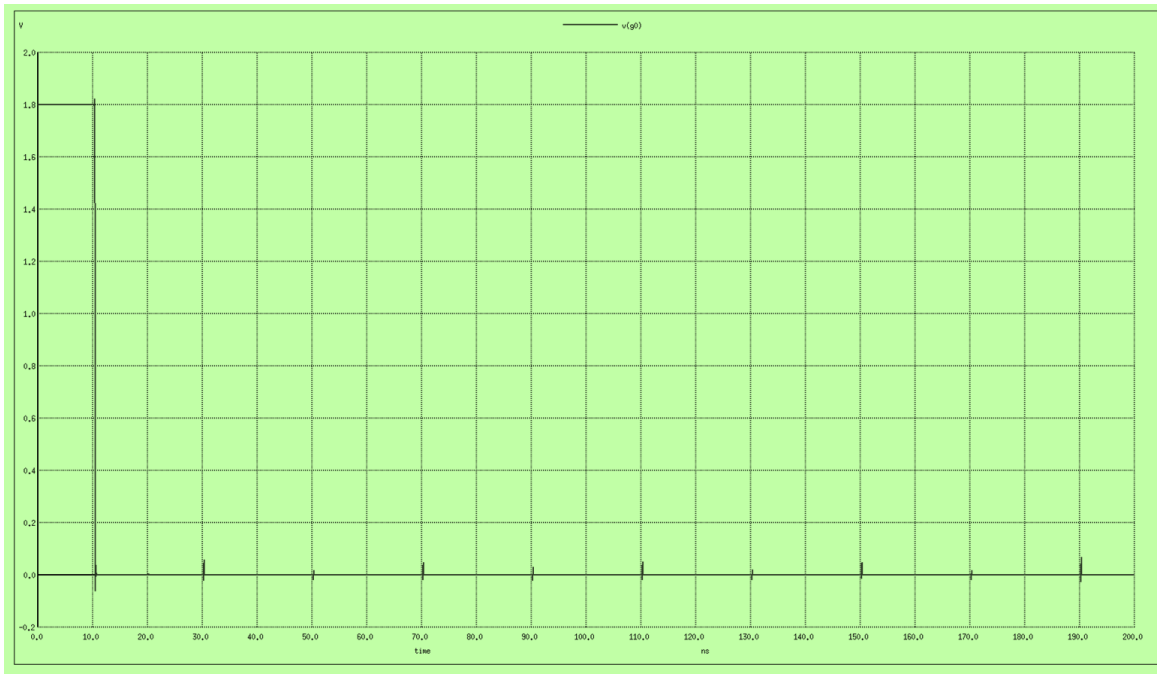


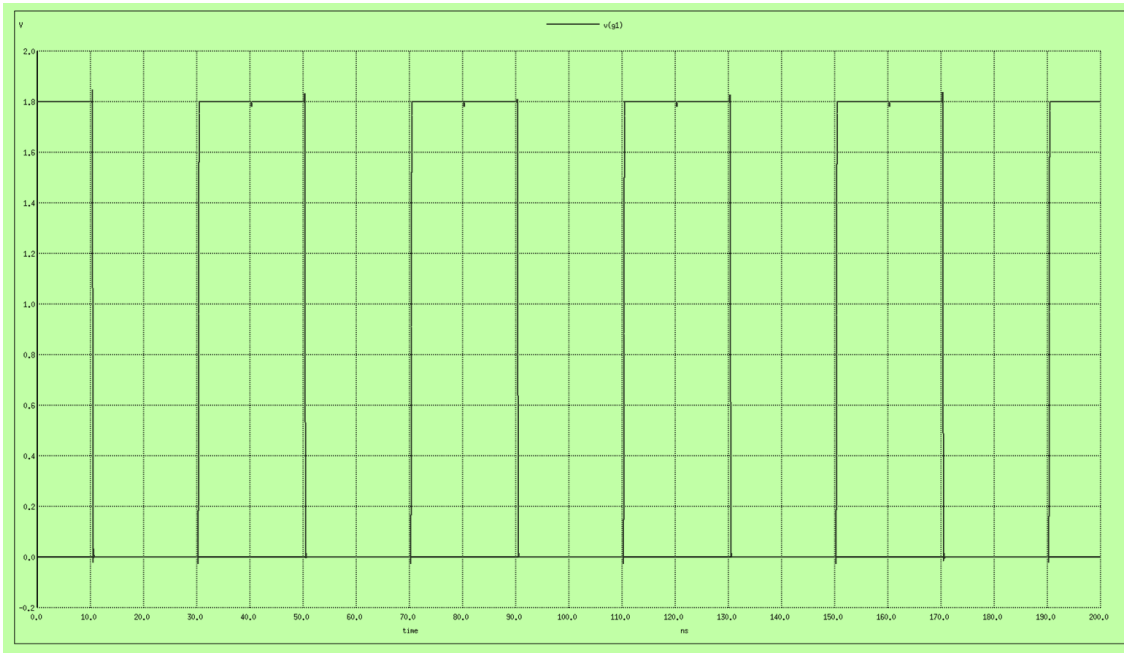


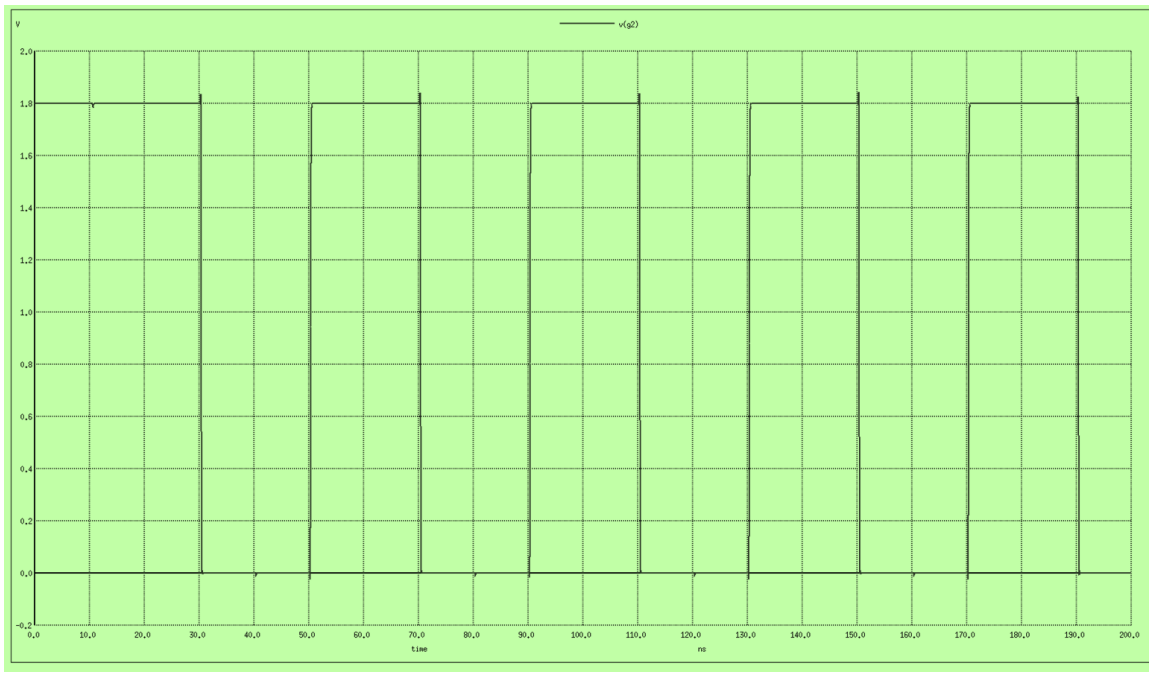
The explanation is the same as a3a2a1a0.

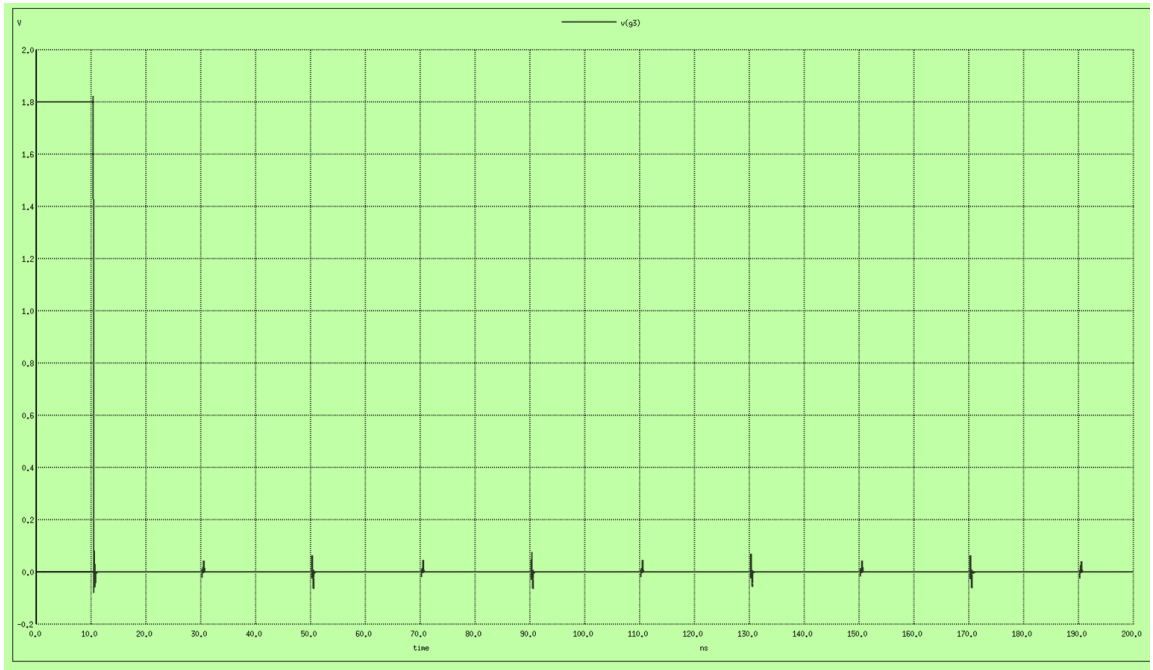
1. Checking the functionality of Propagate and generate block.

Generate Block:

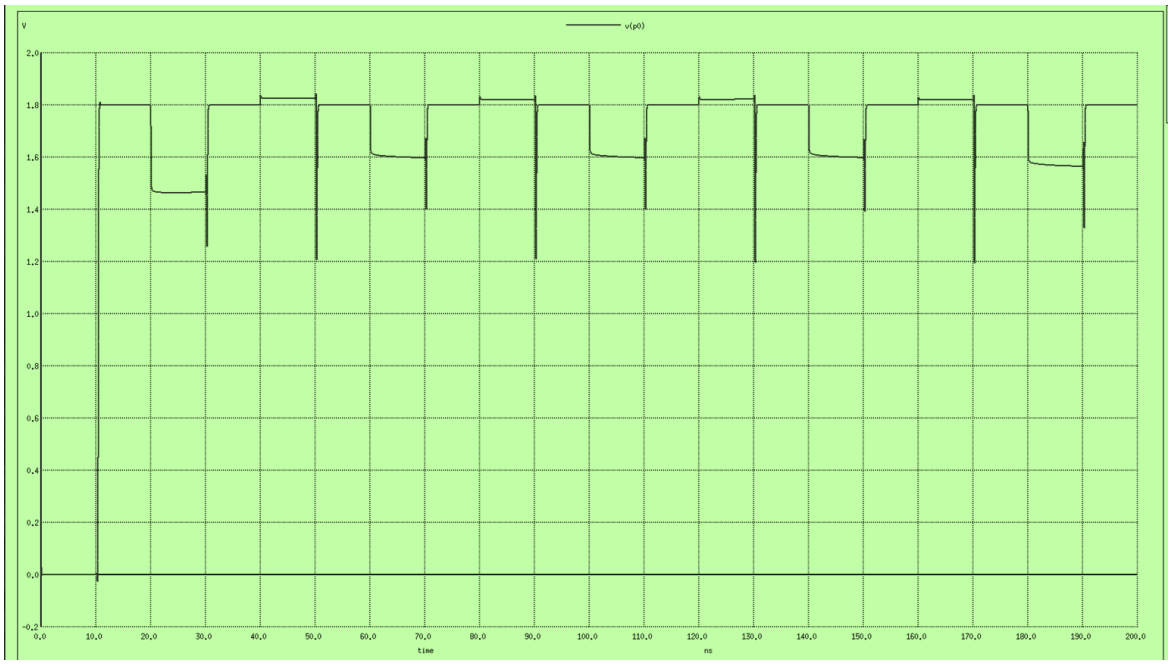


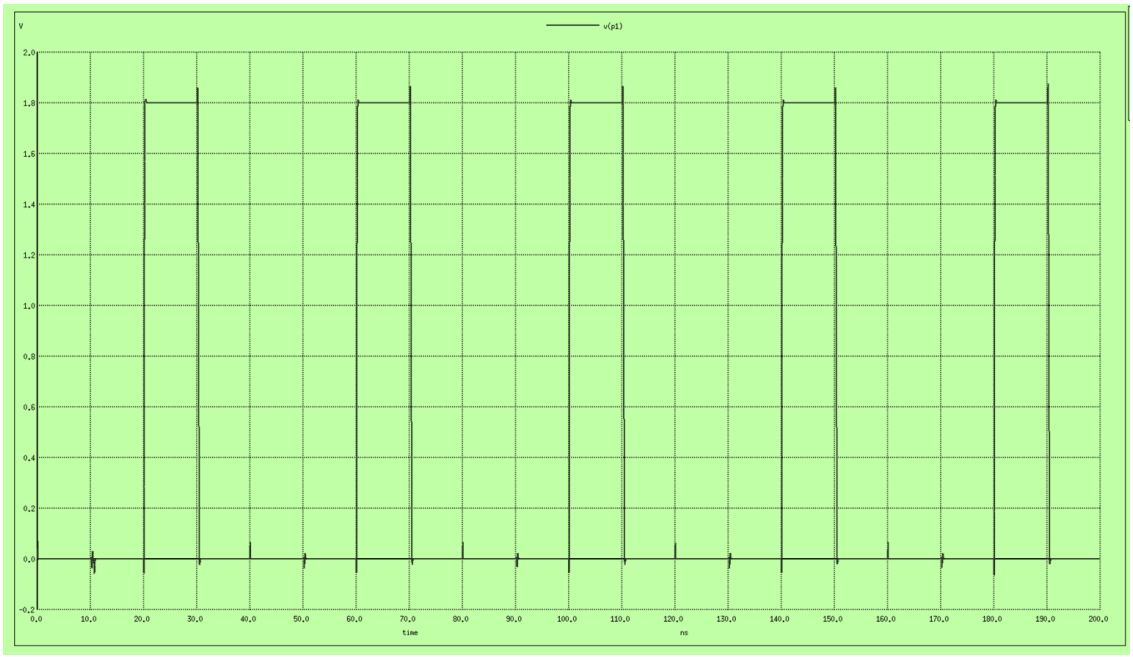


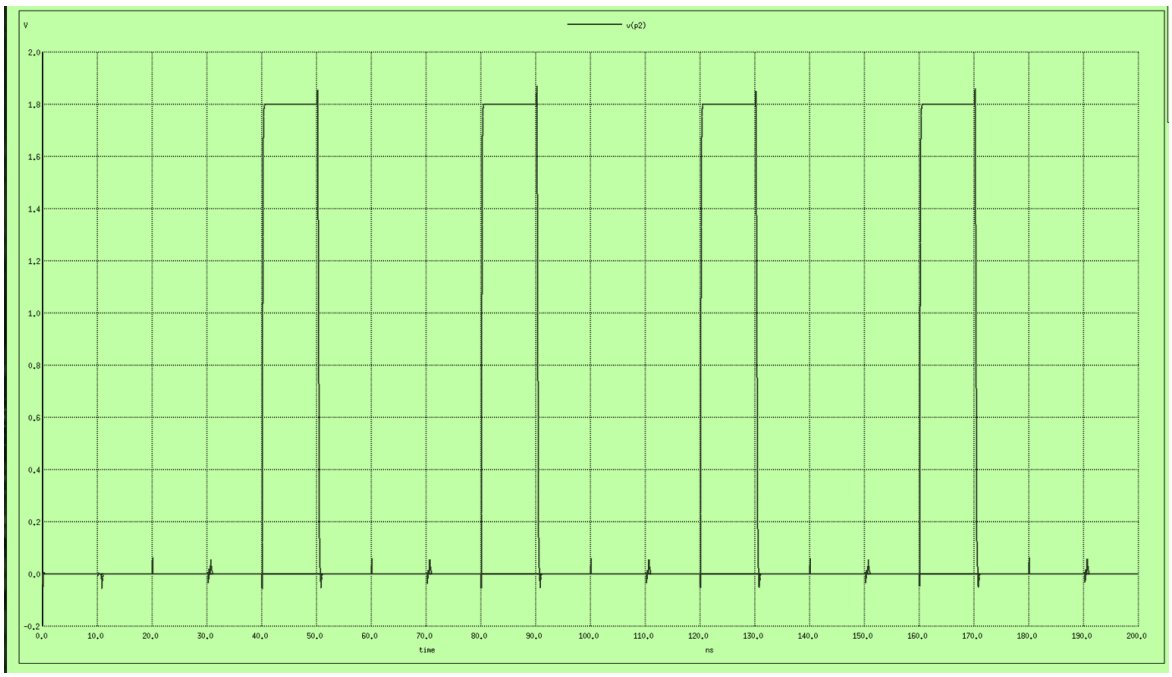


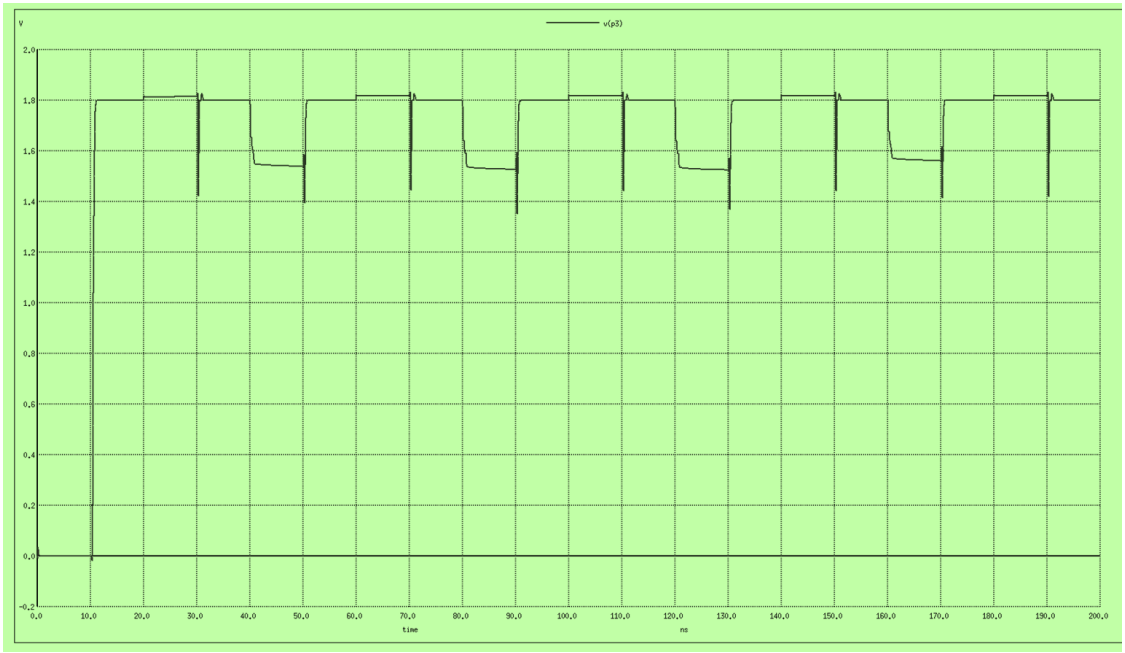


Propagate block:


v(p0)
V
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 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns


v(p1)
V
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 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns



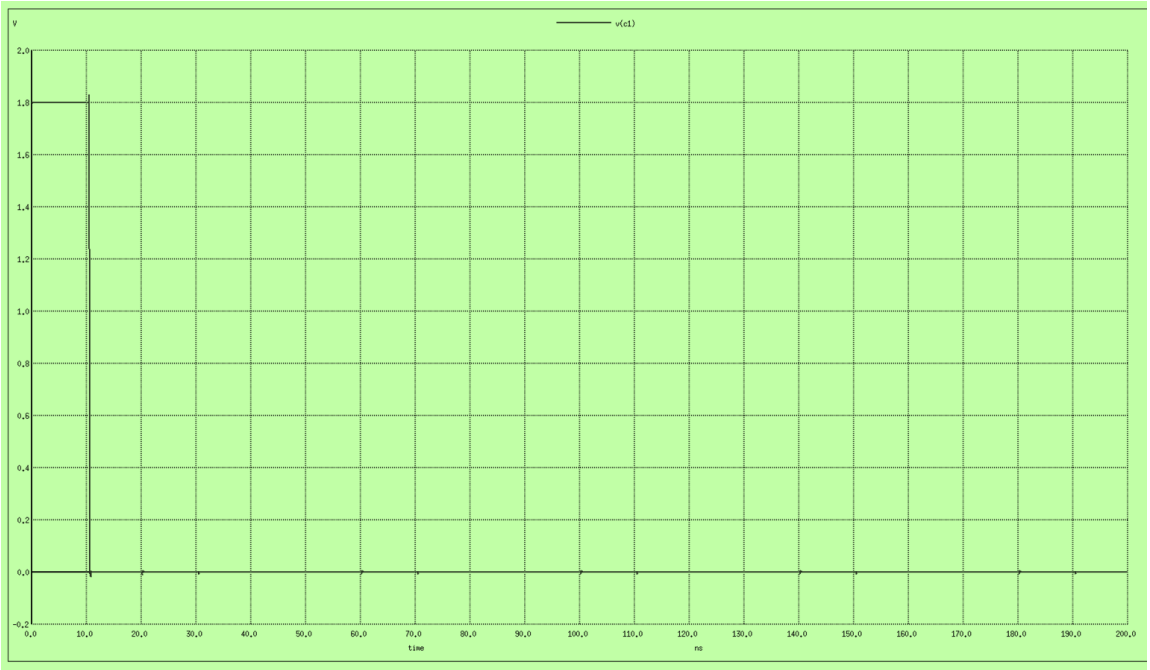


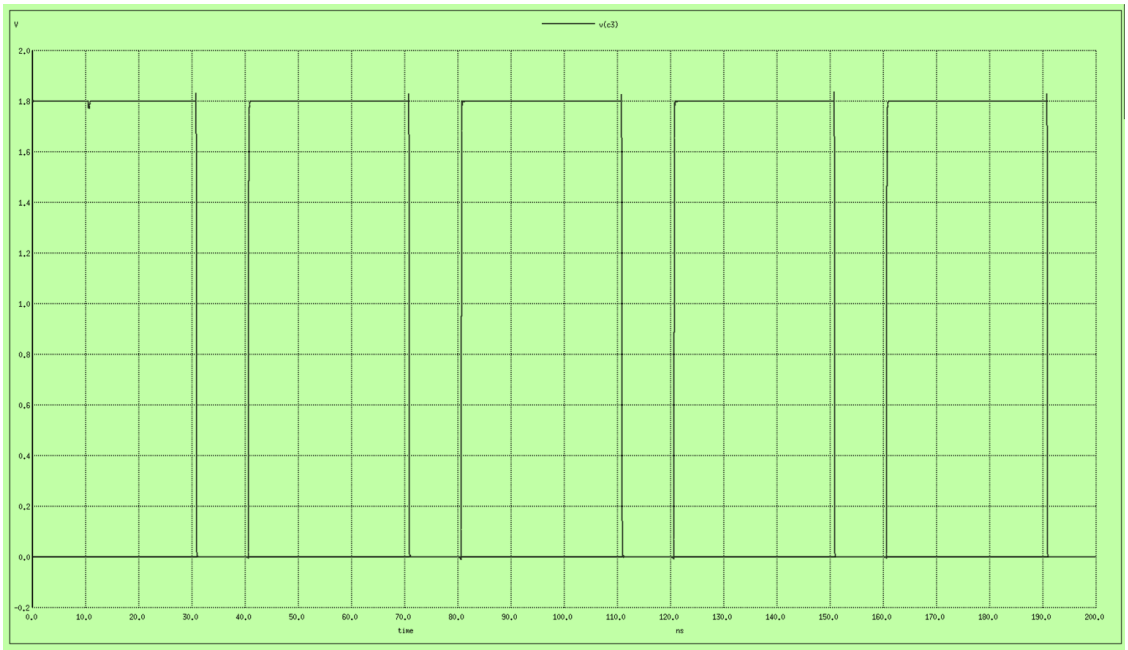
Each and every plot is as expected and both the blocks are working fine.

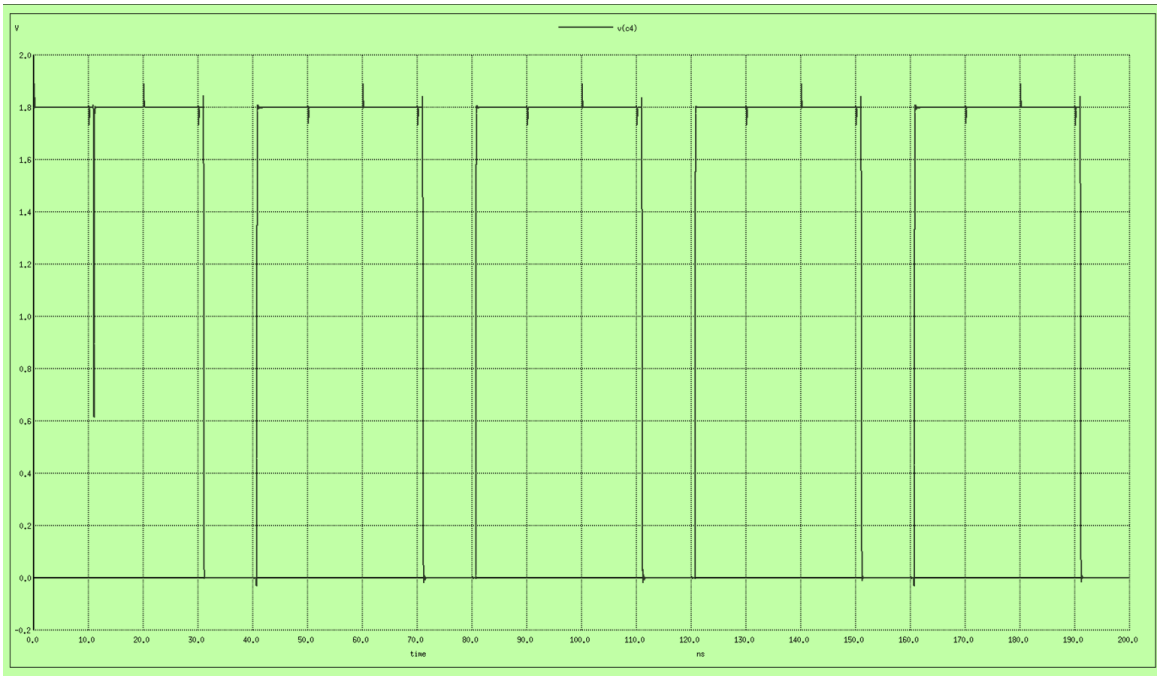
Please move to next page-

2. Carry look ahead block:

All the carries(for every bit) are as follows:

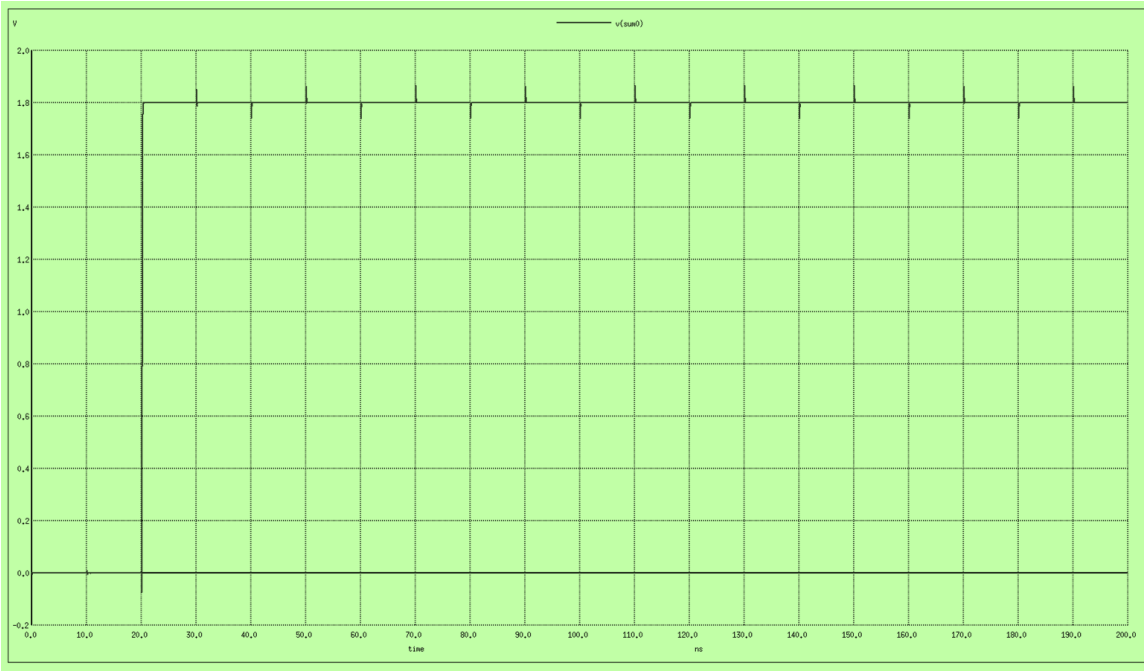


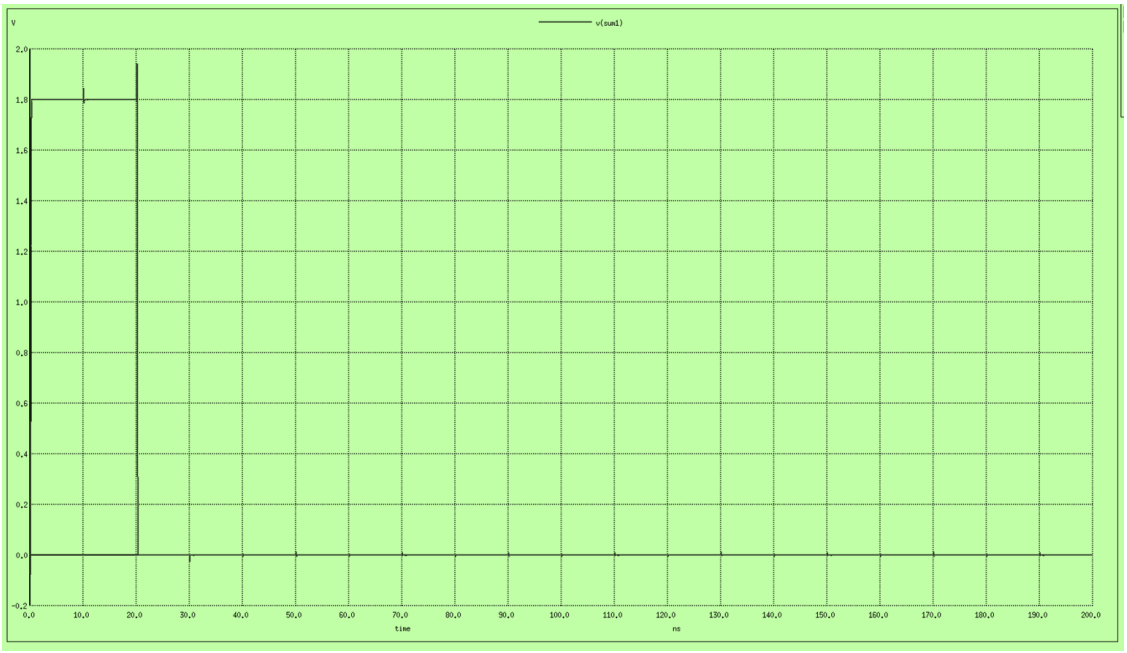


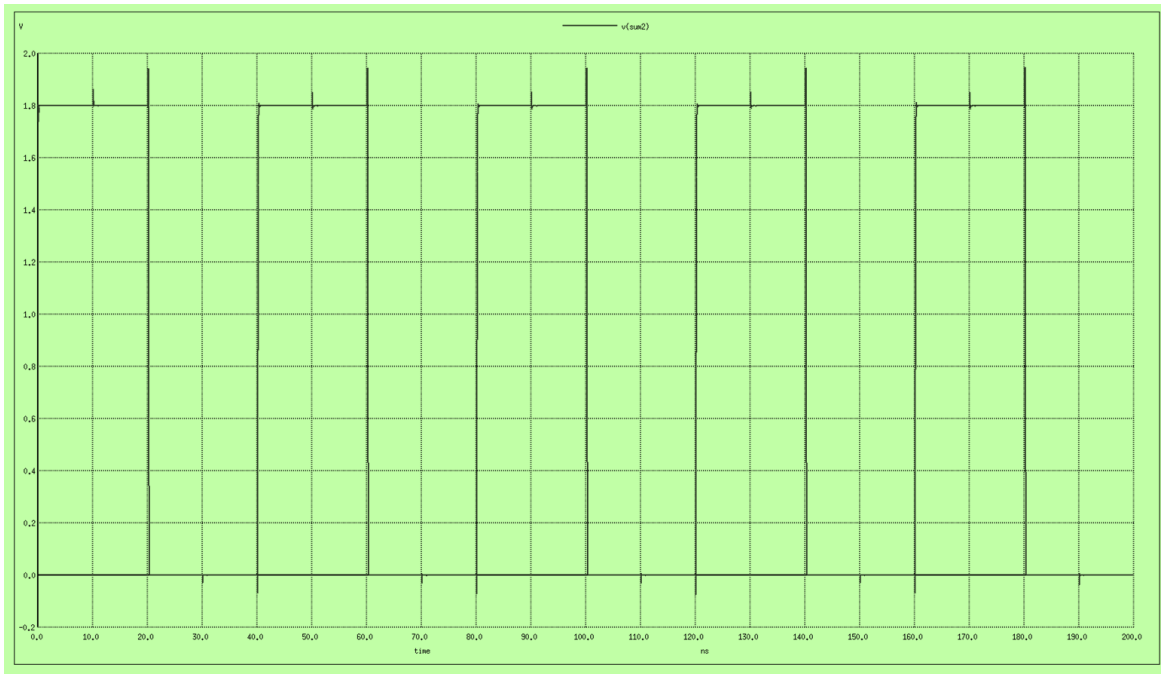


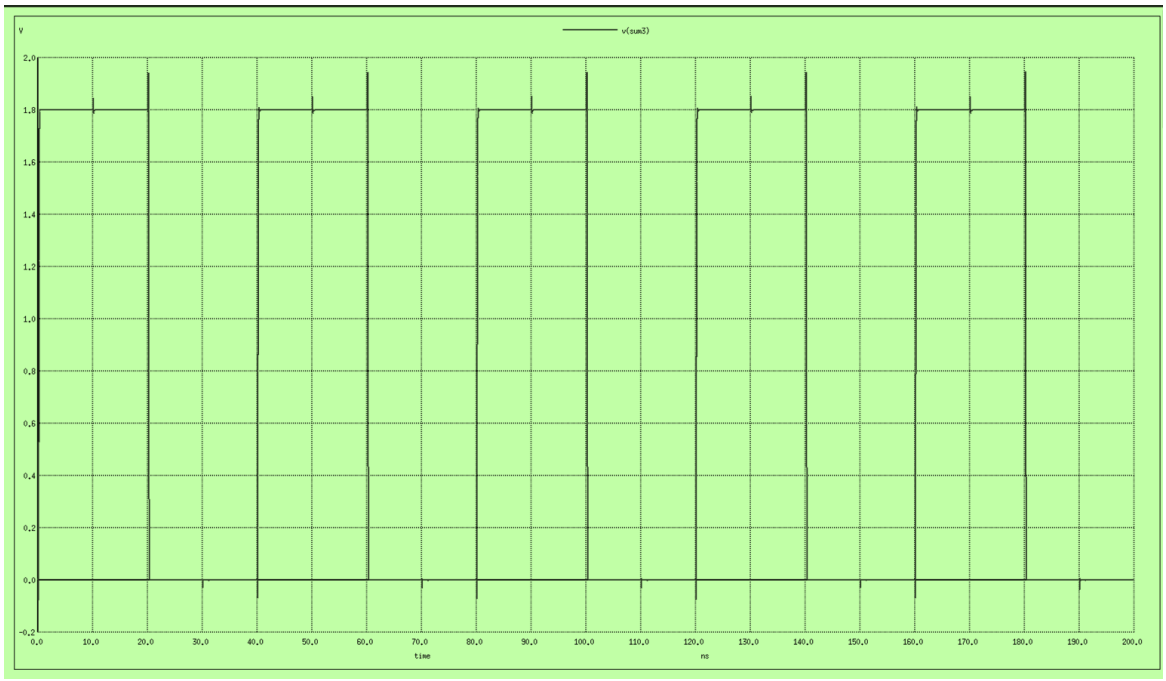
3. for sum block:

For every bit, we have named sum as sum0, sum1, sum2, sum3 which is as follows:


v(sum0)
V
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 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns


v(sum1)
V
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 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns


v(sum2)
V
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 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns

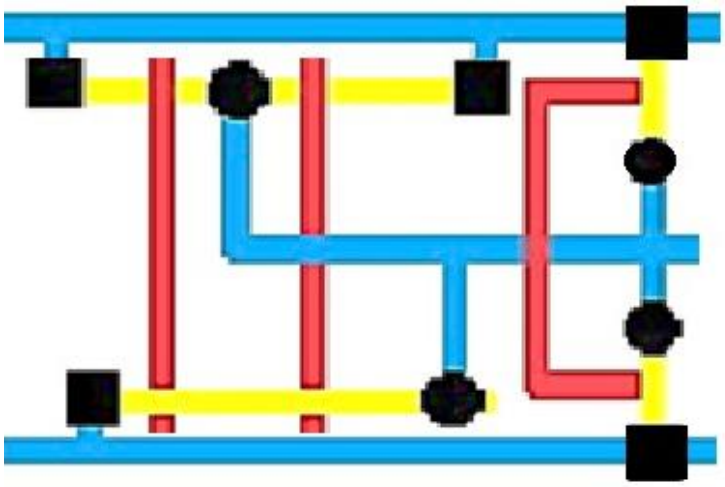

v(sum3)
V
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns

4. ANS FOUR- Setup time, Hold Time, and Delays:

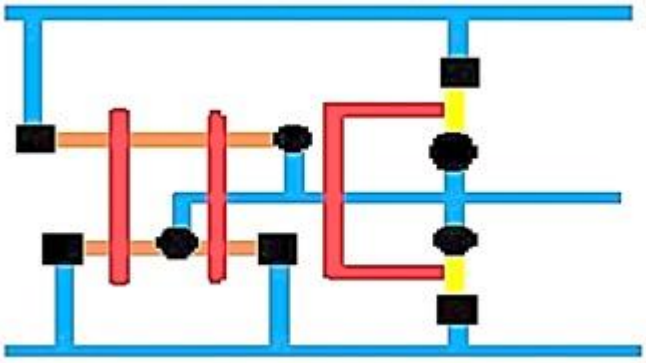
Setup Time is the amount of time until the clock's positive edge during which feedback must stay unchanged in order to get the right output.
The initialization time is 0.5 nanoseconds (calculated manually)
Hold Time is the shortest period of time after the clock's positive edge during which input must remain unchanged in order to obtain the right output.
It takes 0.2ns to complete the configuration (calculated manually)
The measure sentence calculates the C to Q delay, which is as follows:

5. ANS FIVE- Stick Diagrams

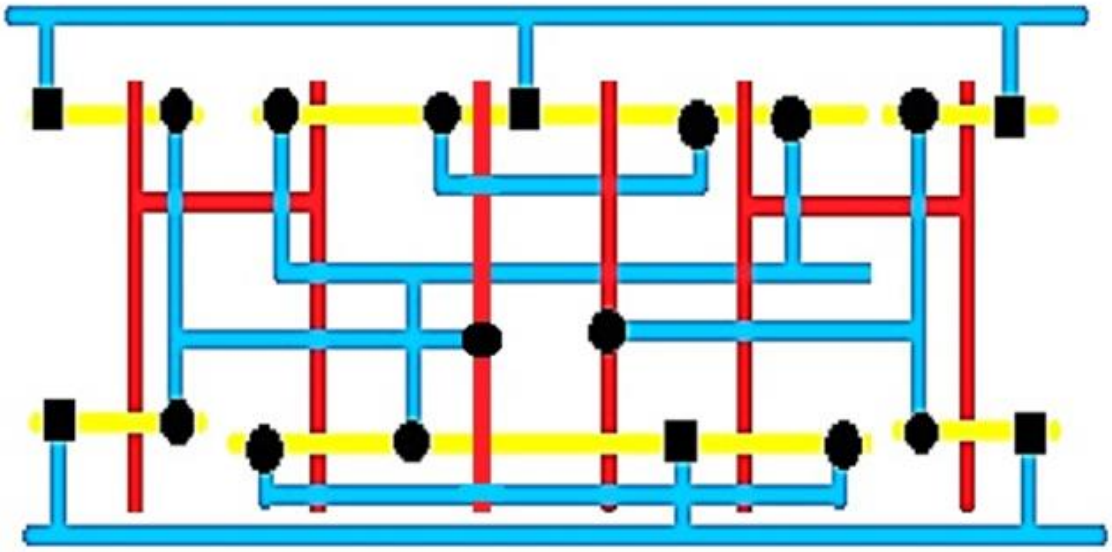
AND GATE:



OR GATE:



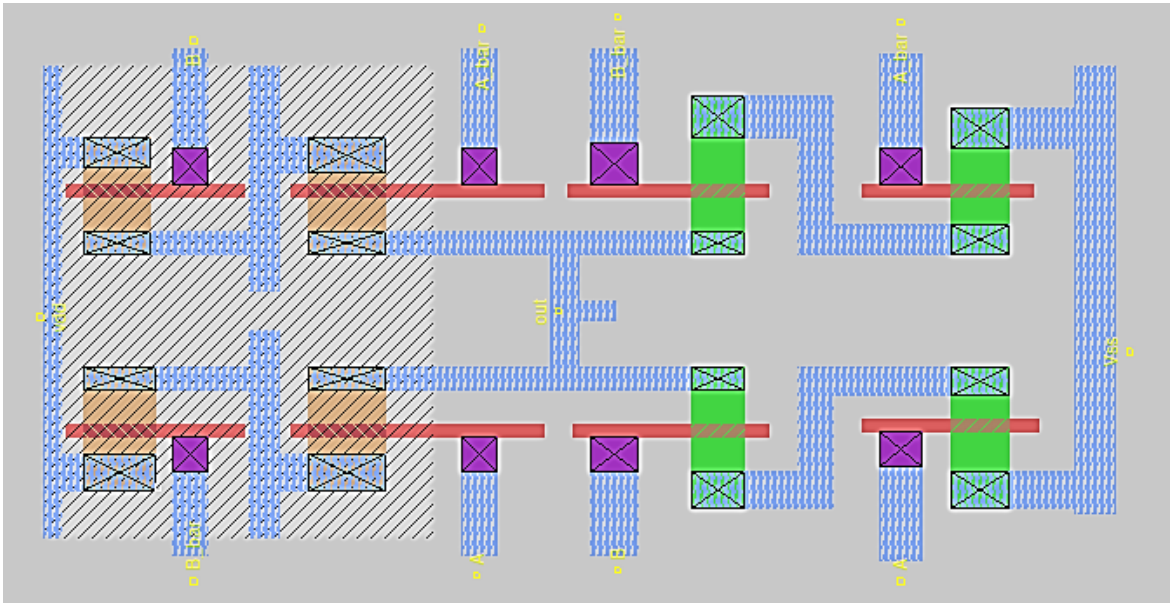
XOR GATE:



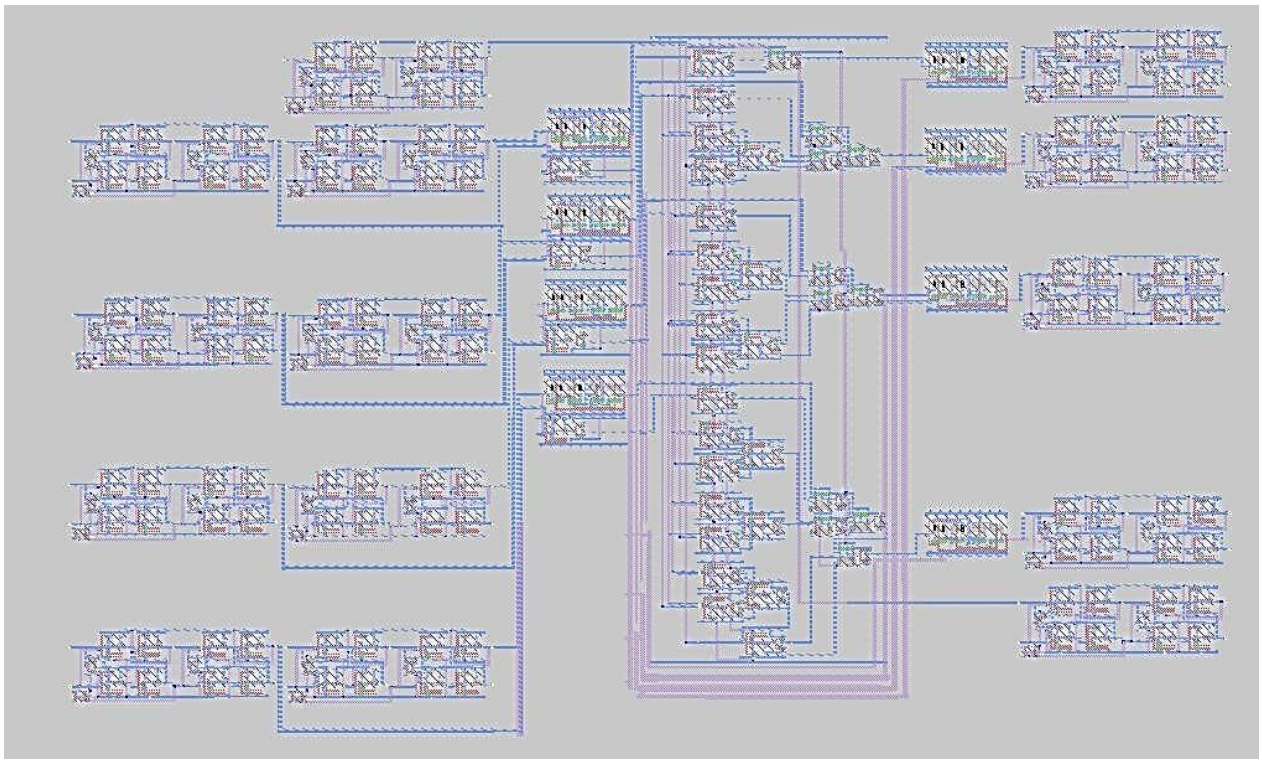
6- ANS SIX- Magic Layout and Simulation Results

First, since the propagate and generate blocks are made up of XOR and AND gates, we must test their functionality.

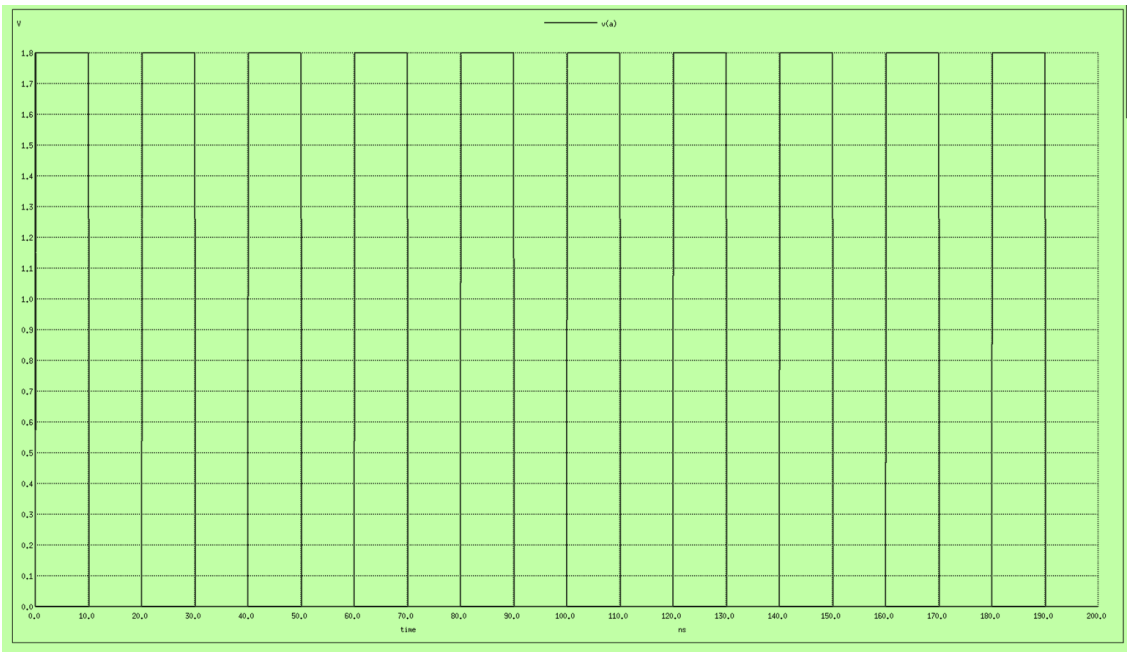
Layout for XOR gate:

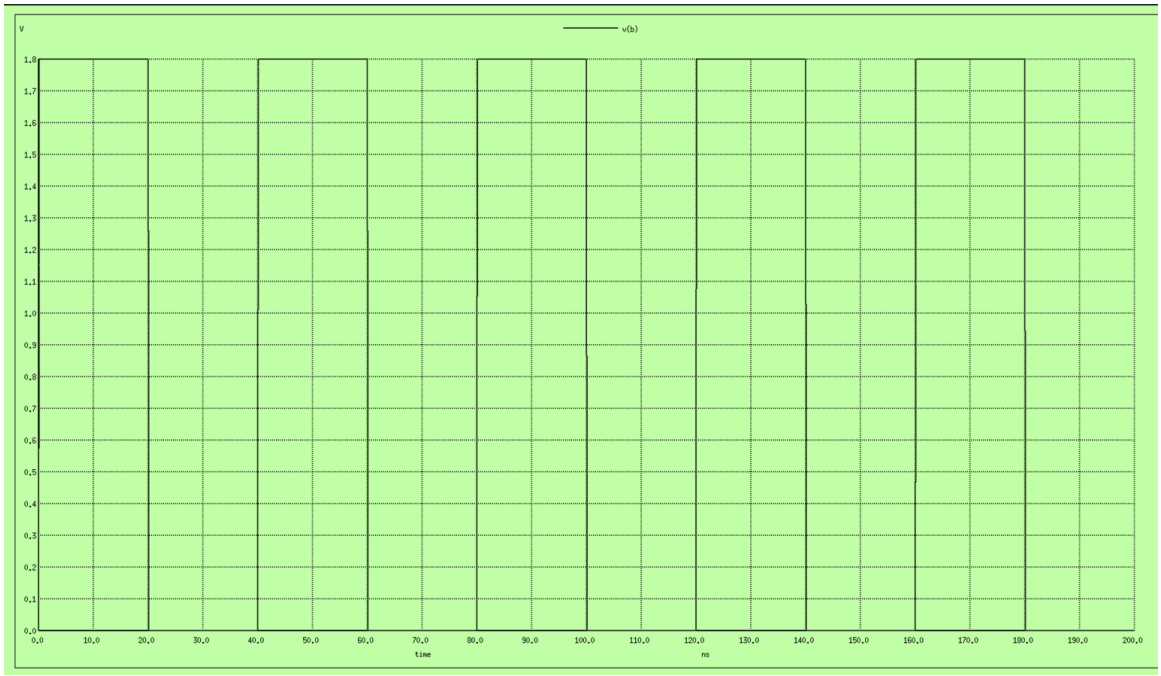


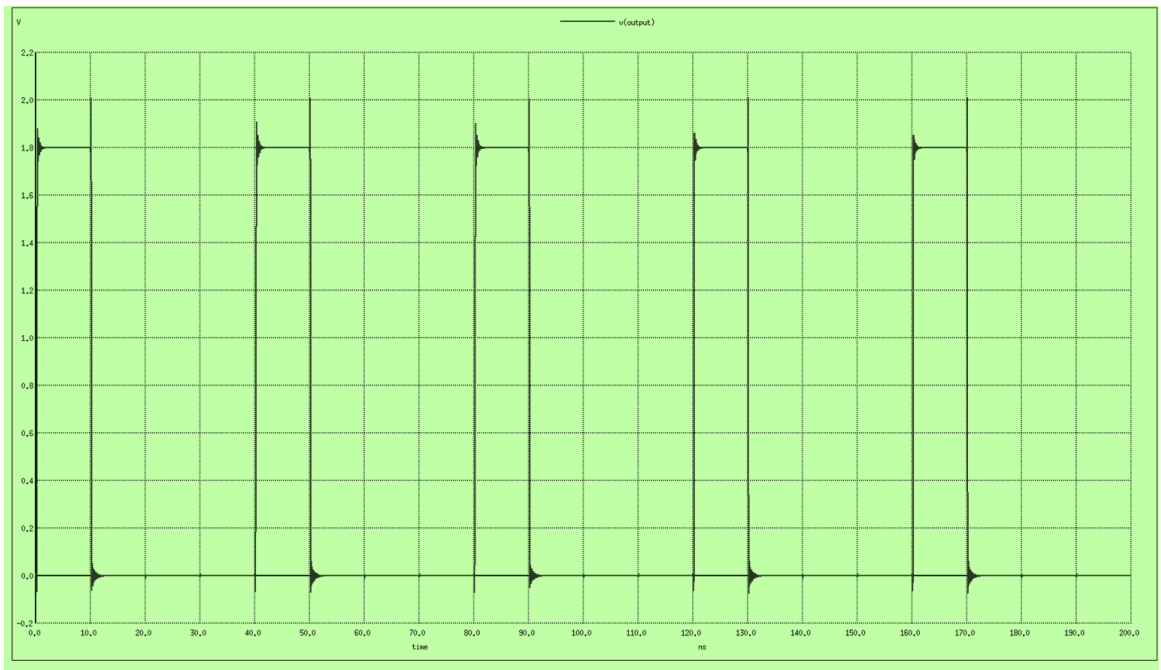
Layout for CLA Block:



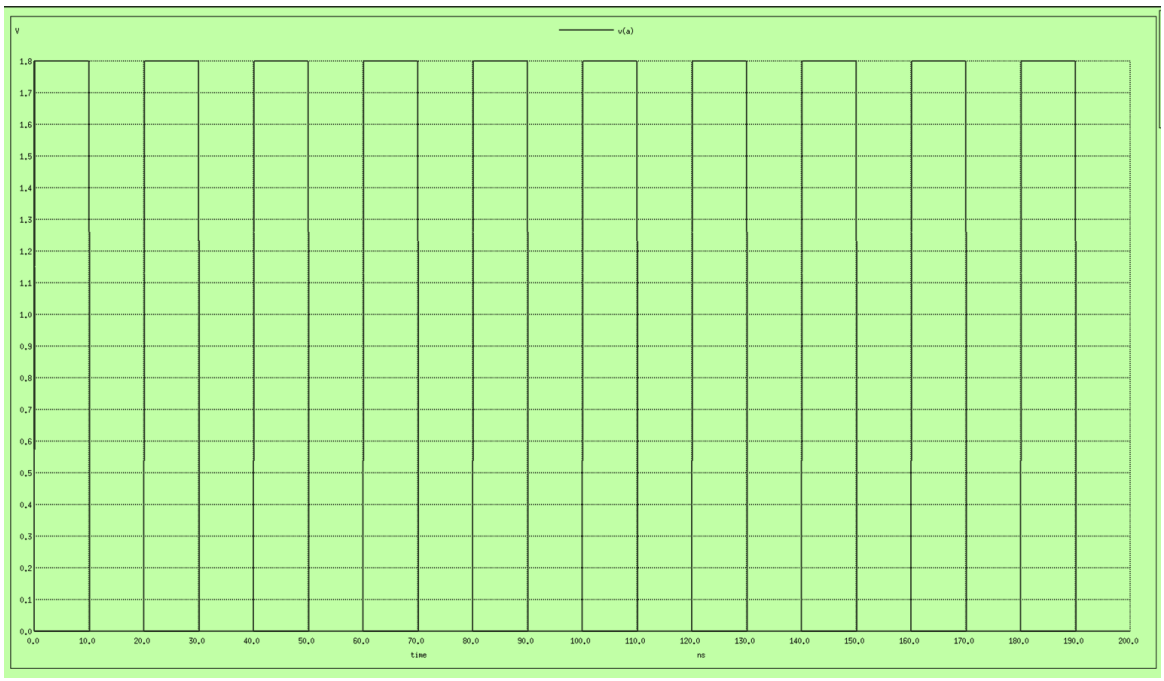
For AND gate:

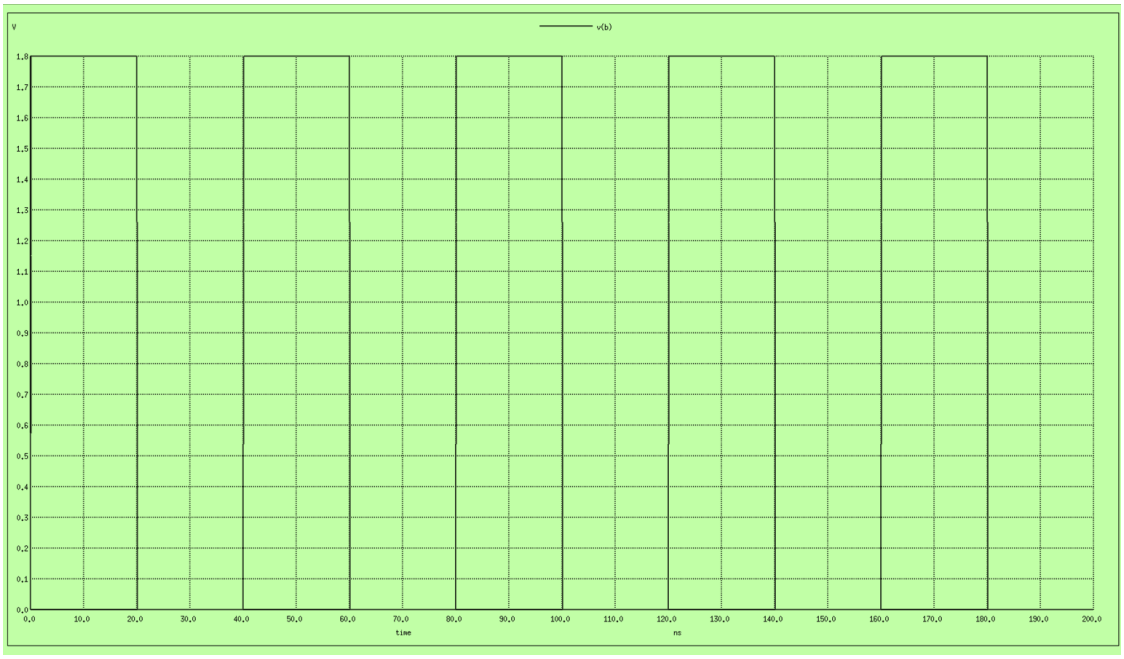

V
v(a)
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
100.0
110.0
120.0
130.0
140.0
150.0
160.0
170.0
180.0
190.0
200.0
time
ns

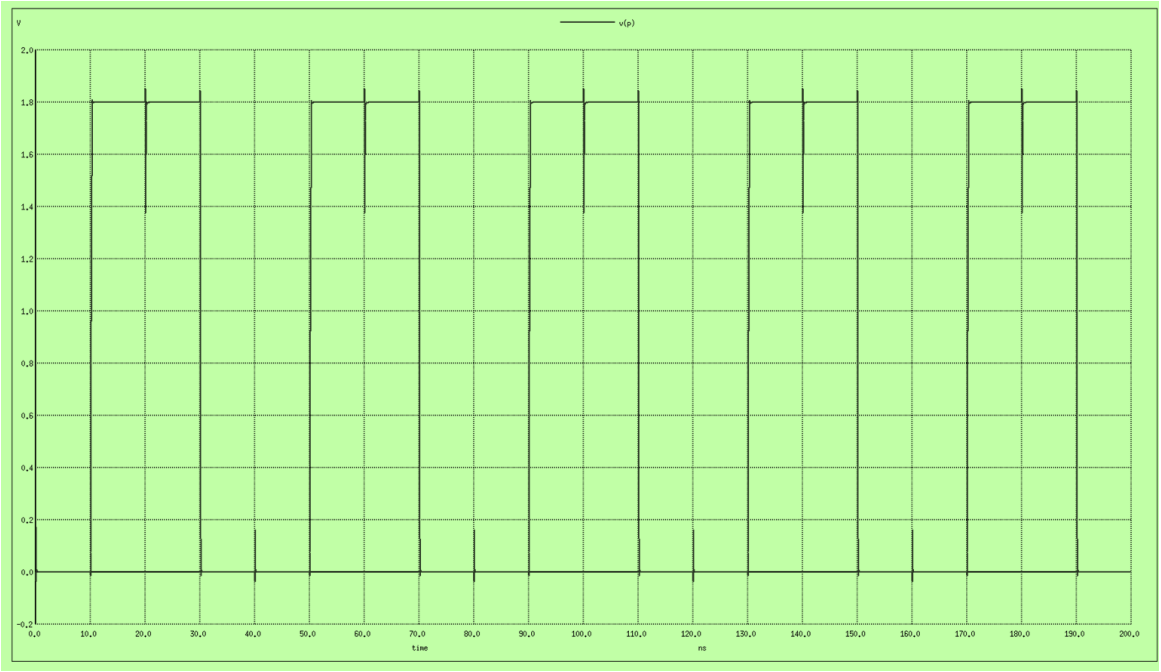




For XOR gate

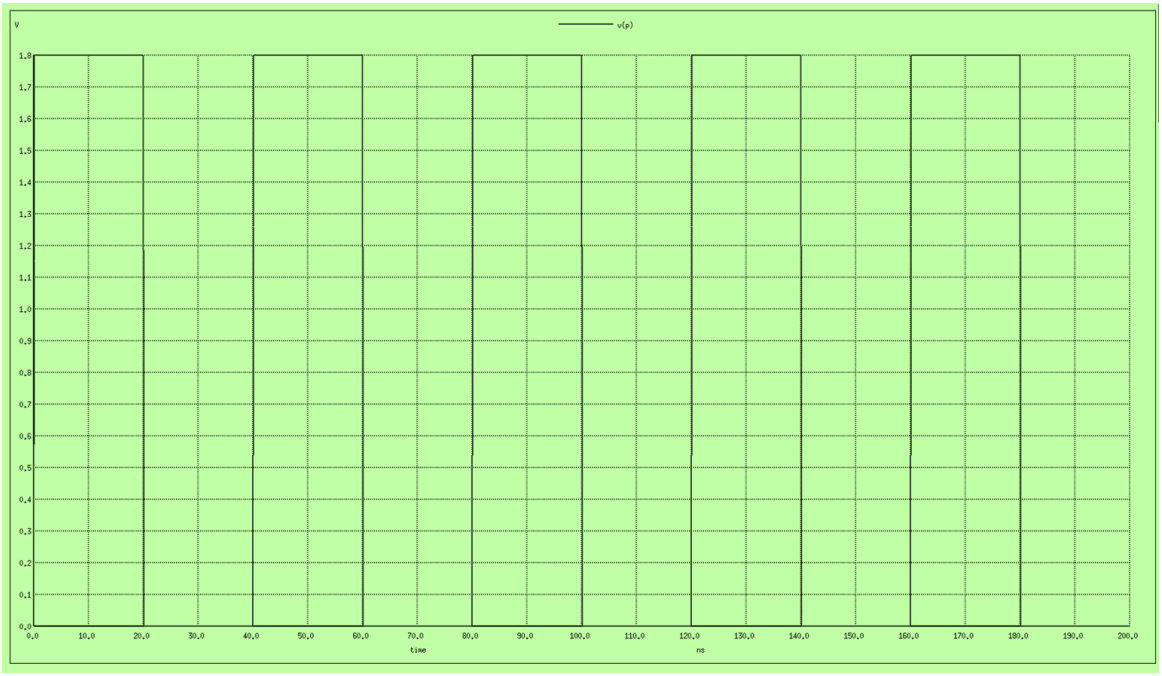

v(a)
V
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns

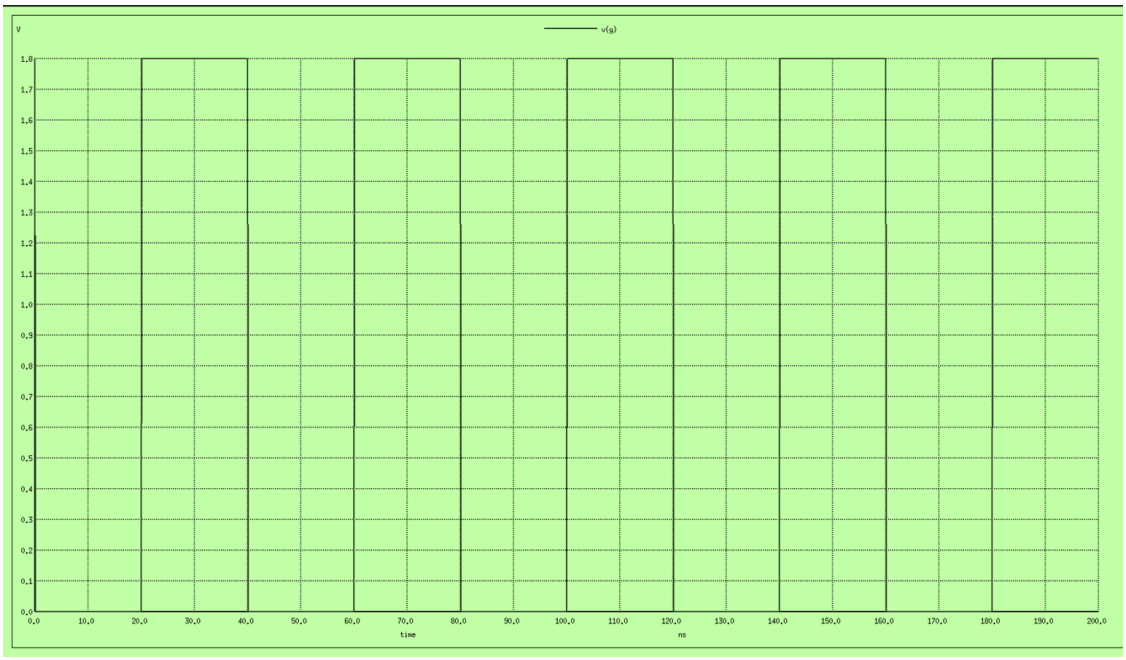

v(b)
V
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns

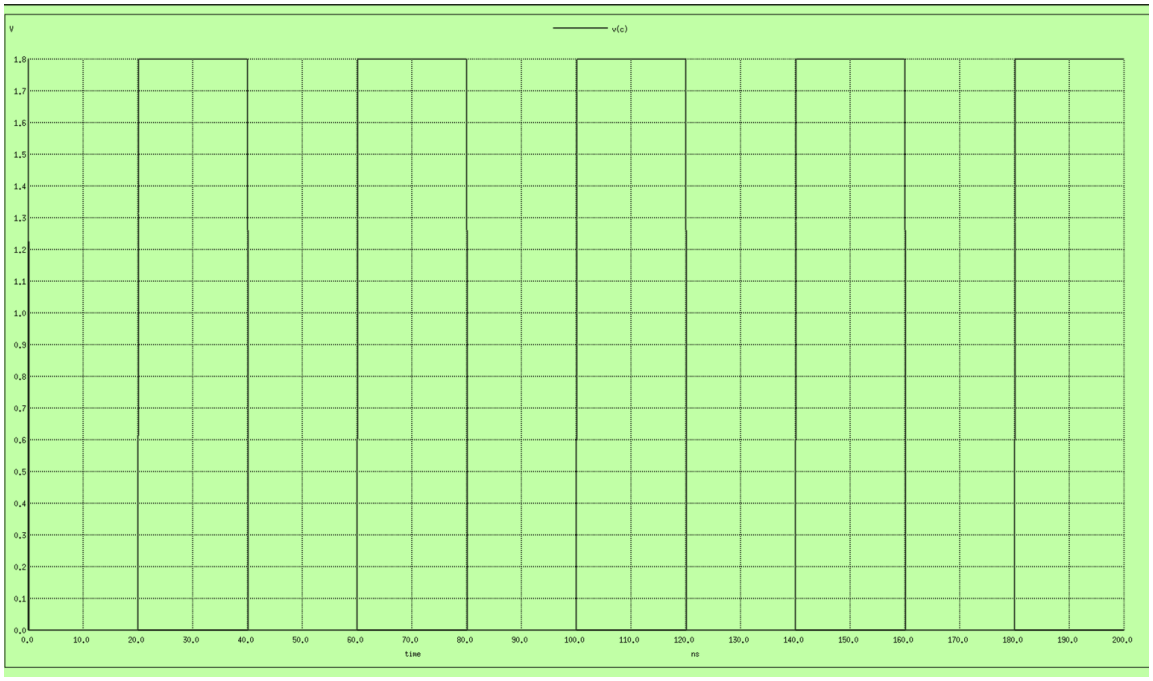


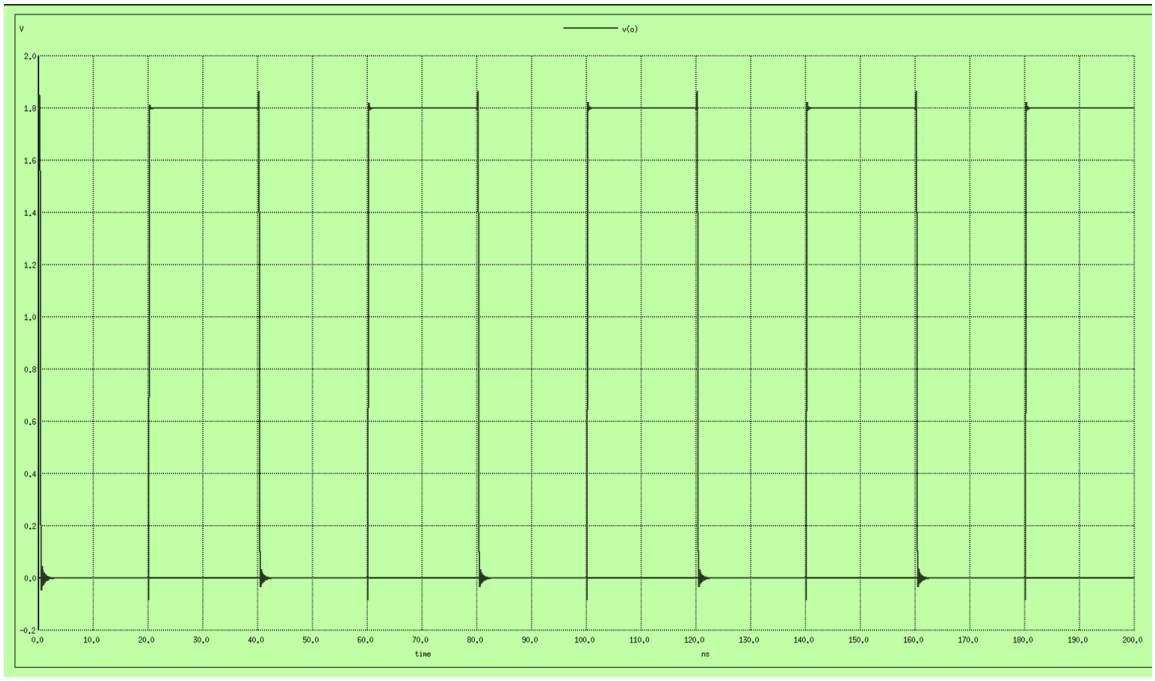
for CLA Block:

The inputs are p, g, and c, where p stands for propagate, g for generate, and c for hold. The following is the current functionality of the CLA block:


v(p)
V
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 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns


v(g)
V
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 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns





The results are almost identical to those of the schematic simulation; there may be a difference due to parasitic capacitances, although it is not discernible.

7- ANS SEVEN-NGSPICE

Code:

```
.include TSMC_180nm.txt
.param SUPPLY=1.8
.param LAMBDA=0.09u
.param width_P={20*LAMBDA}
.param width_N={10*LAMBDA}
.global gnd vdd
Vdd   vdd   gnd   'SUPPLY'
* A = 1100
vin1 d0 0 pulse 1.8 0 0ns 100ps 100ps 19.9ns 40ns
vin2 d1 0 pulse 1.8 0 0ns 100ps 100ps 19.9ns 40ns
vin3 d2 0 pulse 0 1.8 0ns 100ps 100ps 19.9ns 40ns
vin4 d3 0 pulse 0 1.8 0ns 100ps 100ps 19.9ns 40ns
* B = 0101
vin5 e0 0 pulse 0 1.8 0ns 100ps 100ps 19.9ns 40ns
vin6 e1 0 pulse 1.8 0 0ns 100ps 100ps 19.9ns 40ns
vin7 e2 0 pulse 0 1.8 0ns 100ps 100ps 19.9ns 40ns
vin8 e3 0 pulse 1.8 0 0ns 100ps 100ps 19.9ns 40ns
vin9 C0 0 pulse 0 0 0ns 100ps 100ps 99.9ns 200ns
vin10 a0_bar 0 pulse 0 1.8 0ns 100ps 100ps 19.9ns 40ns
vin11 a1_bar 0 pulse 0 1.8 0ns 100ps 100ps 19.9ns 40ns
vin12 a2_bar 0 pulse 1.8 0 0ns 100ps 100ps 19.9ns 40ns
vin13 a3_bar 0 pulse 1.8 0 0ns 100ps 100ps 19.9ns 40ns
vin14 b0_bar 0 pulse 1.8 0 0ns 100ps 100ps 19.9ns 40ns
vin15 b1_bar 0 pulse 0 1.8 0ns 100ps 100ps 19.9ns 40ns
vin16 b2_bar 0 pulse 1.8 0 0ns 100ps 100ps 19.9ns 40ns
vin17 b3_bar 0 pulse 0 1.8 0ns 100ps 100ps 19.9ns 40ns
vin18 clock 0 pulse 0 1.8 0ns 100ps 100ps 9.9ns 20ns
vin19 notclock 0 pulse 1.8 0 0ns 100ps 100ps 9.9ns 20ns
.subckt xor P a b a_bar b_bar vdd gnd
.param width_P={20*LAMBDA}
.param width_N={10*LAMBDA}
M1      P       b       f     f  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      f       a       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}
M3      P       b_bar       g     g  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}
M4      g       a_bar       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}
M5      d       b_bar       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}
M6      P       a       d    d  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}
```

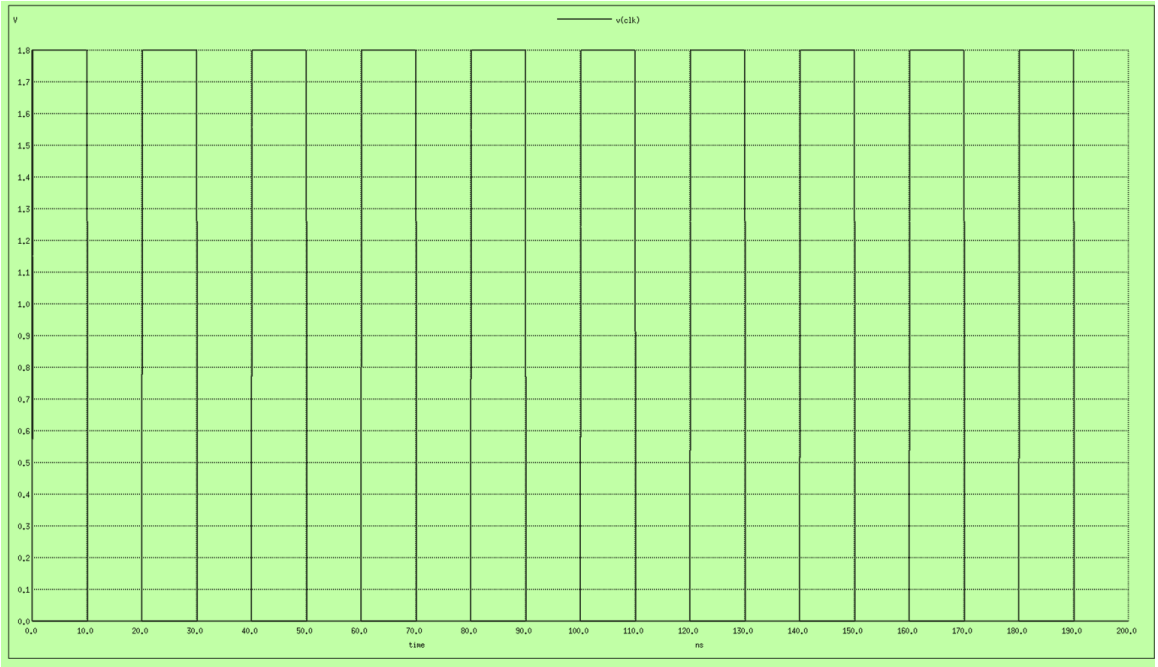
```
M7      e       b       vdd     vdd  CMOSP   W={width_P}   L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA}
PD={10*LAMBDA+2*width_P}
M8      P       a_bar      e    e  CMOSP   W={width_P}   L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA}
PD={10*LAMBDA+2*width_P}
.ends xor
.subckt and G a b vdd gnd
.param width_P={20*LAMBDA}
.param width_N={10*LAMBDA}
M1      G       r       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      G       r       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 r a 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}
M4  r  b   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}
M5      r     a      q    q  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}
M6      q     b      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}
.ends and
.subckt or C a b vdd gnd
M1      u       a      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}
M2      v       b      u    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      gnd       a       v     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}
M4      gnd       b       v     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}
M5      C       v      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}
M6      gnd     v       C     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 or
.subckt inv yi xi 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     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 inv
```

```
.subckt nand fout a b vdd gnd
.param width_P={20*LAMBDA}
.param width_N={10*LAMBDA}
M1 fout a 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}
M2  fout  b   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      fout     a      c    c  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}
M4      c     b      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}
.ends nand
.subckt dlatch q d cl vdd gnd
x100 fout d cl vdd gnd nand
x200 d_bar d vdd gnd inv
x101 fout1 d_bar cl vdd gnd nand
x102 q fout nq vdd gnd nand
x103 nq q fout1 vdd gnd nand
.ends dlatch
*input flipflops
x0 y0 d0 clock vdd gnd dlatch
x1 y1 d1 clock vdd gnd dlatch
x2 y2 d2 clock vdd gnd dlatch
x3 y3 d3 clock vdd gnd dlatch
x4 z0 e0 clock vdd gnd dlatch
x5 z1 e1 clock vdd gnd dlatch
x6 z2 e2 clock vdd gnd dlatch
x7 z3 e3 clock vdd gnd dlatch
x8 a0 y0 notclock vdd gnd dlatch
x9 a1 y1 notclock vdd gnd dlatch
x10 a2 y2 notclock vdd gnd dlatch
x11 a3 y3 notclock vdd gnd dlatch
x12 b0 z0 notclock vdd gnd dlatch
x13 b1 z1 notclock vdd gnd dlatch
x14 b2 z2 notclock vdd gnd dlatch
x15 b3 z3 notclock vdd gnd dlatch
*Propagate and generate block
x16 P0 a0 b0 a0_bar b0_bar vdd gnd xor
x17 P1 a1 b1 a1_bar b1_bar vdd gnd xor
x18 P2 a2 b2 a2_bar b2_bar vdd gnd xor
x19 P3 a3 b3 a3_bar b3_bar vdd gnd xor
x20 G0 a0 b0 vdd gnd and
x21 G1 a1 b1 vdd gnd and
x22 G2 a2 b2 vdd gnd and
x23 G3 a3 b3 vdd gnd and
*CLA block
x24 OP0 P0 C0 vdd gnd and
x25 C1 G0 OP0 vdd gnd or
x26 OP1 P1 OP0 vdd gnd and
x27 temp1 P1 G0 vdd gnd and
x28 temp2 temp1 G1 vdd gnd or
x29 C2 temp2 OP1 vdd gnd or
x30 OP2 OP1 P2 vdd gnd and
```

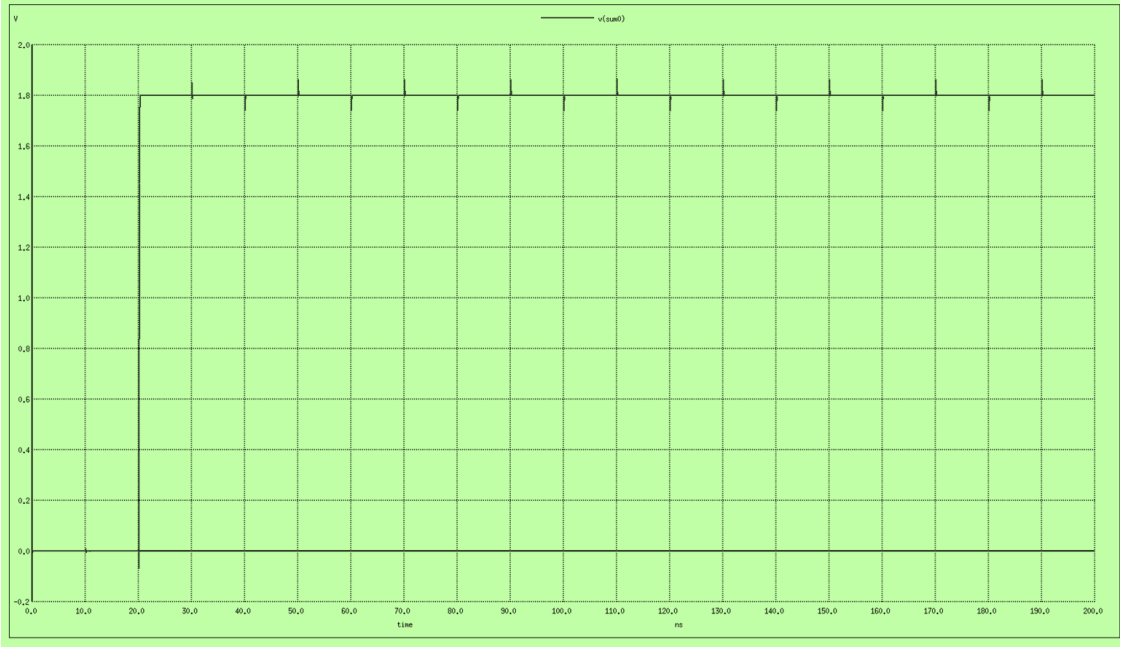
```
x31 temp3 temp1 P2 vdd gnd and
x32 temp4 P2 G1 vdd gnd and
x33 temp5 G2 temp4 vdd gnd or
x34 temp6 temp5 temp3 vdd gnd or
x35 C3 temp6 OP2 vdd gnd or
x36 OP3 P3 OP2 vdd gnd and
x37 temp7 temp3 P3 vdd gnd and
x38 temp8 temp4 P3 vdd gnd and
x39 temp9 P3 G2 vdd gnd and
x40 temp10 temp9 G3 vdd gnd or
x41 temp11 temp10 temp8 vdd gnd or
x42 temp12 temp11 temp7 vdd gnd or
x43 C4 temp12 OP3 vdd gnd or
*Sum block
x44 P0_bar P0 vdd gnd inv
x45 C0_bar C0 vdd gnd inv
x46 S0 P0 C0 P0_bar C0_bar vdd gnd xor
x47 P1_bar P1 vdd gnd inv
x48 C1_bar C1 vdd gnd inv
x49 S1 P1 C1 P1_bar C1_bar vdd gnd xor
x50 P2_bar P2 vdd gnd inv
x51 C2_bar C2 vdd gnd inv
x52 S2 P2 C2 P2_bar C2_bar vdd gnd xor
x53 P3_bar P3 vdd gnd inv
x54 C3_bar C3 vdd gnd inv
x55 S3 P3 C3 P3_bar C3_bar vdd gnd xor
*output flipflops
x56 out0 S0 notclock vdd gnd dlatch
x57 out1 S1 notclock vdd gnd dlatch
x58 out2 S2 notclock vdd gnd dlatch
x59 out3 S3 notclock vdd gnd dlatch
x60 sum0 out0 clock vdd gnd dlatch
x61 sum1 out1 clock vdd gnd dlatch
x62 sum2 out2 clock vdd gnd dlatch
x63 sum3 out3 clock vdd gnd dlatch
x64 car4 C4 notclock vdd gnd dlatch
x65 carry4 car4 clock vdd gnd dlatch
x66 car2 C2 notclock vdd gnd dlatch
x67 carry2 car2 clock vdd gnd dlatch
.tran 0.1n 200n
.control
set hcopypscolor = 1
set color0=white
set color1=black
run
set curplottitle= "bvaibhawk_2019112021"
plot v(clock)
plot v(d0)
plot v(e0)
* plot v(C0)
* plot v(P2)
* plot v(G2)
plot v(sum0)
plot v(sum1)
* plot v(sum2)
* plot v(sum3)
.endc
```

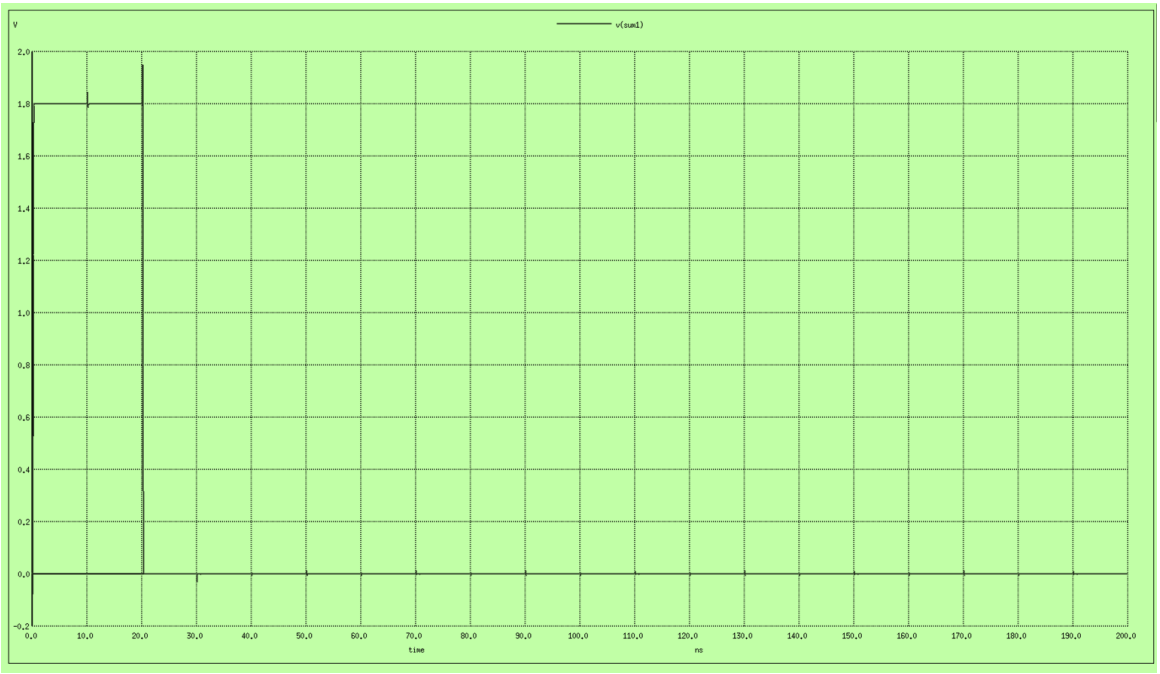
Simulations:

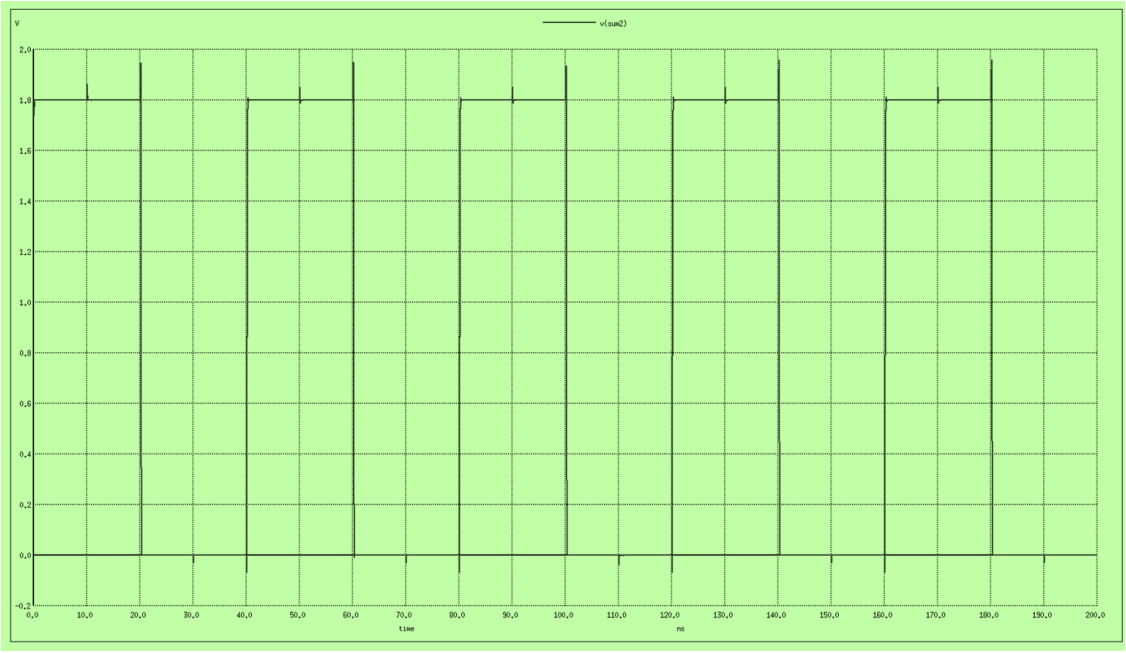
Checking clock

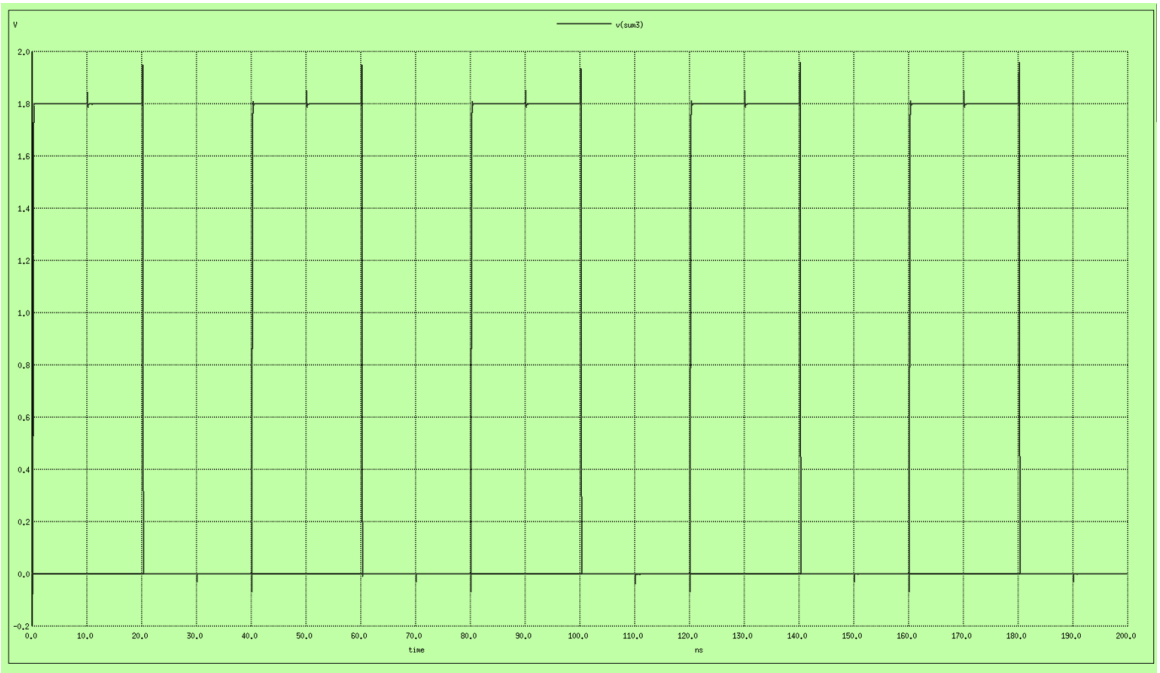


Sums for all the bits:

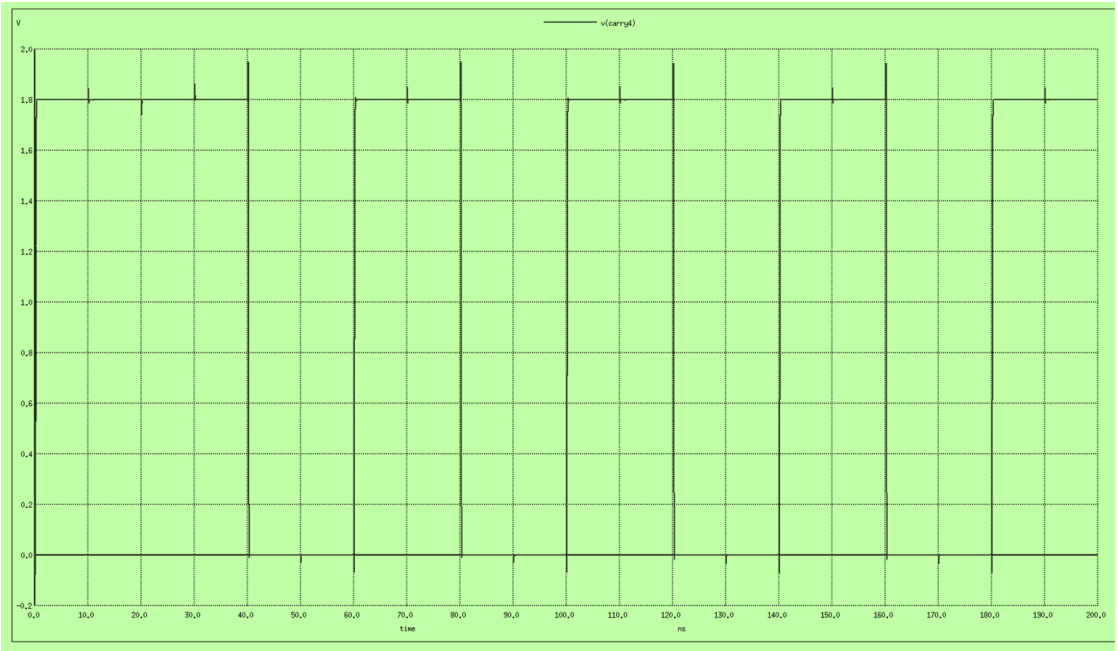



v(sum1)
V
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns


v(sum2)
V
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 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns


V
v(sum3)
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 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns

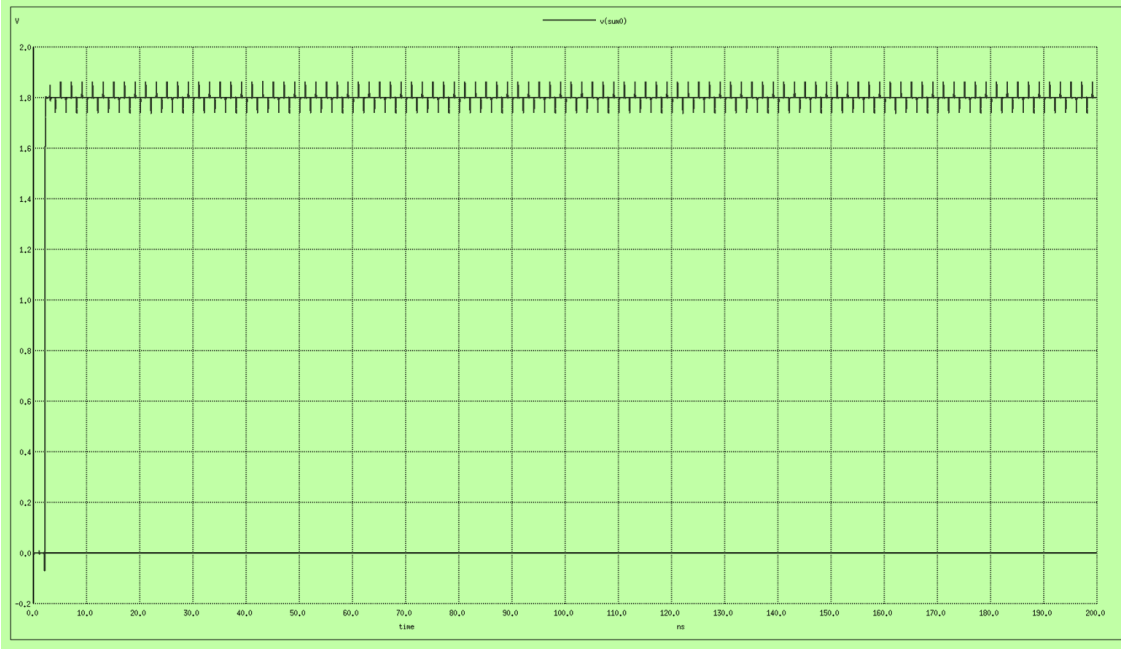
Carry bit-


V
v(carry4)
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 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns

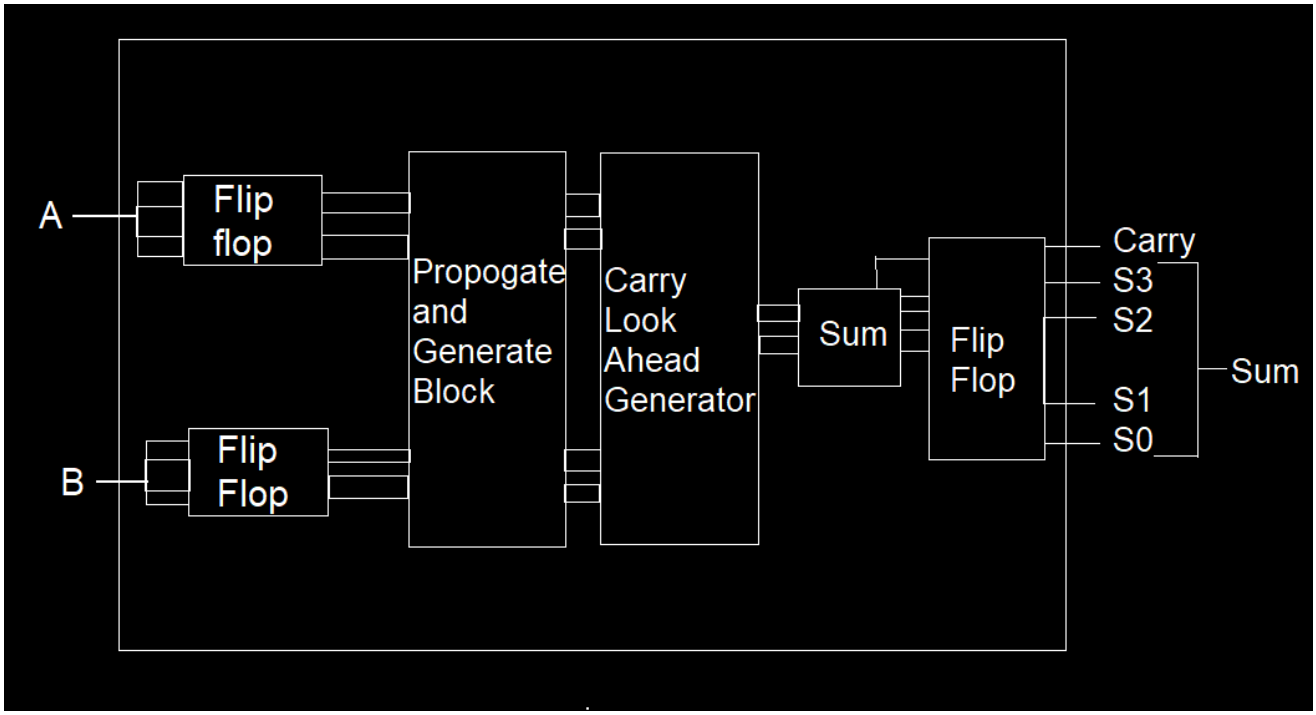
All the outputs are right, as can be shown. Our netlist is up and running.

When I ran the code for various test cases, the worst delay was 0.63821ns, and the clock's minimum toggling time interval should be 2ns or the production would be incorrect.

Below is the distorted signal:

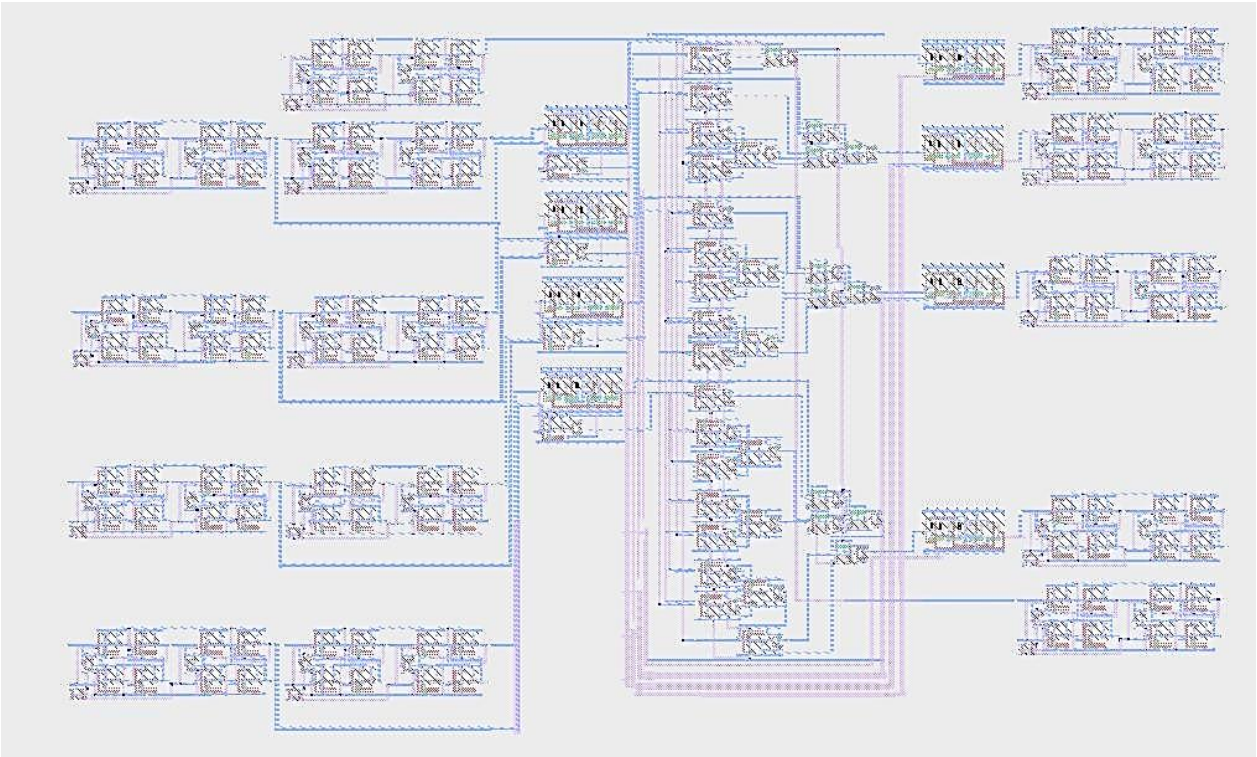


8- ANS EIGHT- Floor Plan:

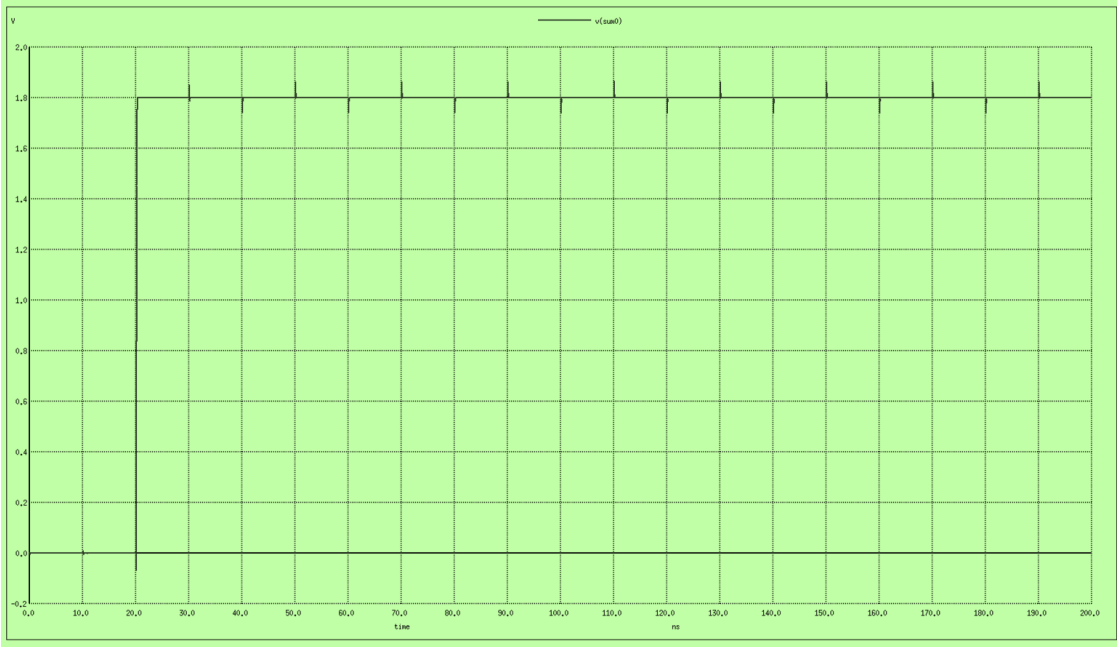


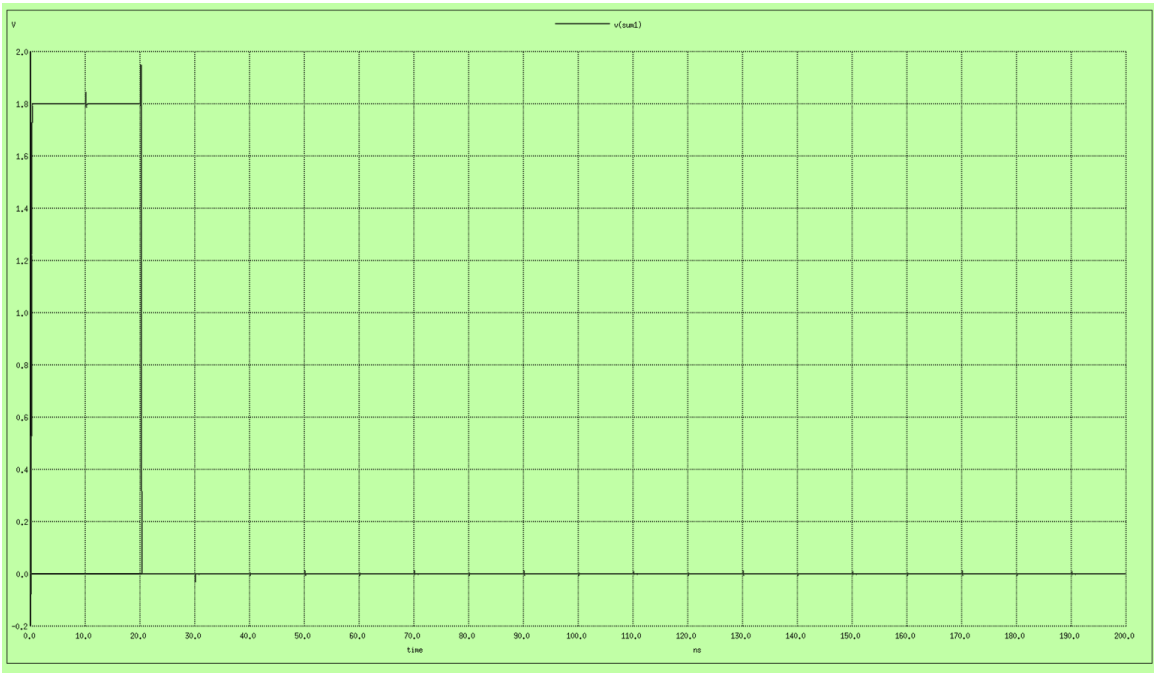
Vertical pitches include the Propagate & Generate block and the Carry Look Ahead Generator. Many of the remaining blocks fall into the horizontal pitch category.

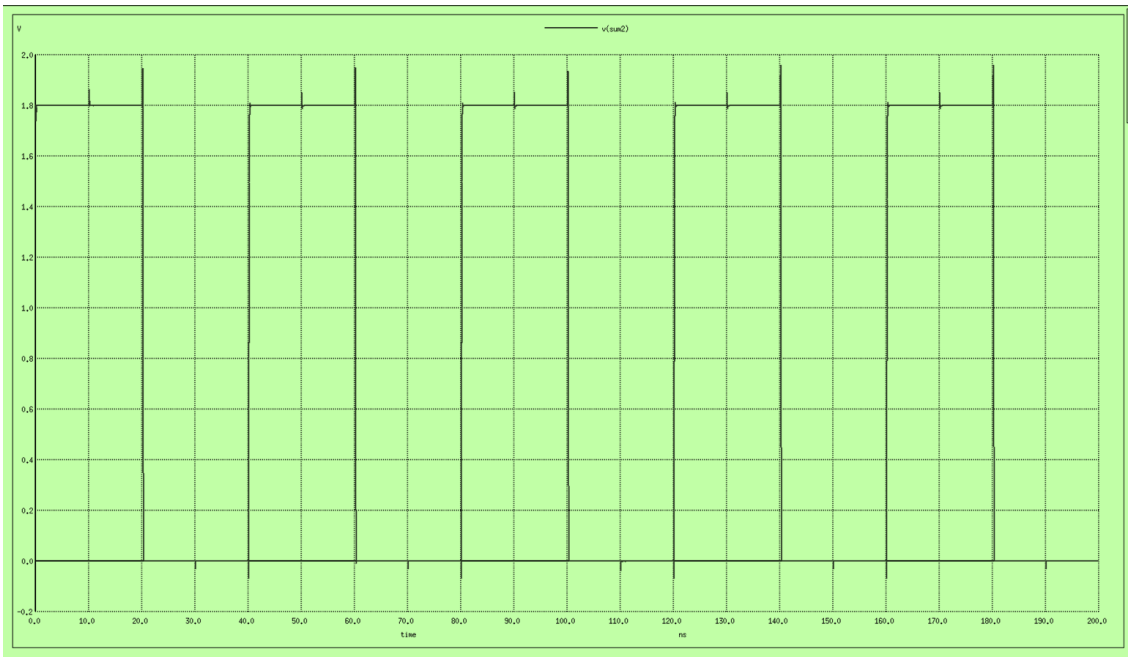
9- ANS NINE- Magic Layout:

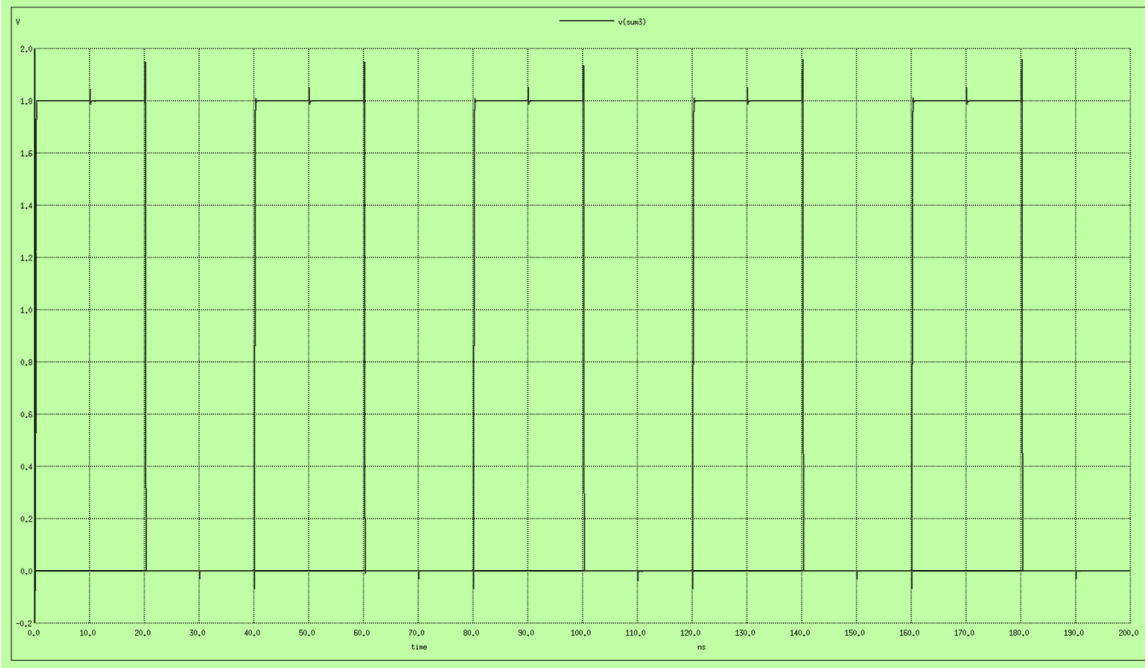


All the plots by running magic netlist are:

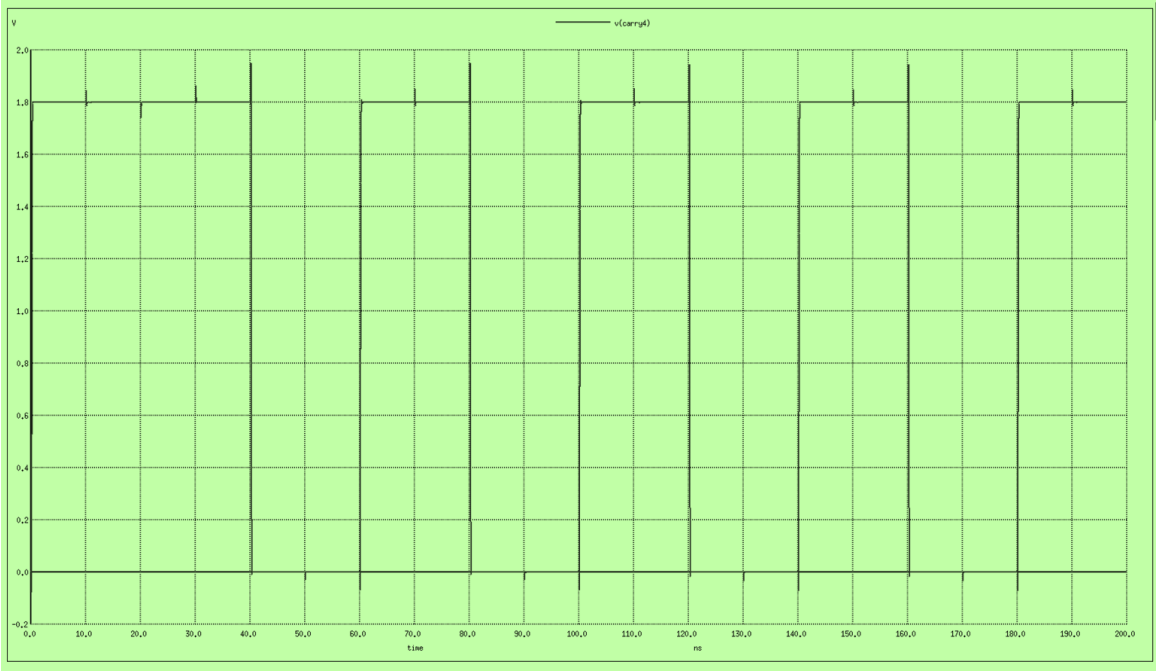



v(sum1)
V
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns


v(sum2)
V
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 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0 110.0 120.0 130.0 140.0 150.0 160.0 170.0 180.0 190.0 200.0
time
ns



And Carry bit is:



Delay Comparison between pre-layout and post-layout is:

Block	Pre-Layout	Post-Layout

AND	0.08294ns	0.08591ns
FLIP-FLOP	0.23132ns	0.30021ns
XOR	0.09202ns	0.10901ns
OR	0.07199ns	0.07698ns
CLA	0.15994ns	0.19012ns
FINAL	0.63821ns	0.76223ns

10- ANS TEN- Delays

The delay of CLA-adder is 0.15994ns and minimum toggling time at which circuit works is 2ns.

11- ANS ELEVEN- Verilog

Code:

```
`timescale 1ns / 1ps
module four_bit_xor(out,a,b);
input [3:0]a,b;
output [3:0]out;
xor (out[0],a[0],b[0]);
xor (out[1],a[1],b[1]);
xor (out[2],a[2],b[2]);
xor (out[3],a[3],b[3]);
endmodule
51
module four_bit_and(out,a,b);
input [3:0]a,b;
output [3:0]out;
and (out[0],a[0],b[0]);
and (out[1],a[1],b[1]);
and (out[2],a[2],b[2]);
and (out[3],a[3],b[3]);
endmodule
module carry(pc,p,c,g);
input [3:0]p,g;
output [4:0]c;
output [15:0]pc;
and (pc[0], p[0], c[0]);
xor (c[1], g[0], pc[0]);
and (pc[1], pc[0], p[1]);
and (pc[2], p[1], g[0]);
xor (pc[3], pc[1], pc[2]);
xor (c[2], pc[3], g[1]);
and (pc[4], pc[1], p[2]);
and (pc[5], pc[2], p[2]);
and (pc[6], p[2], g[1]);
```

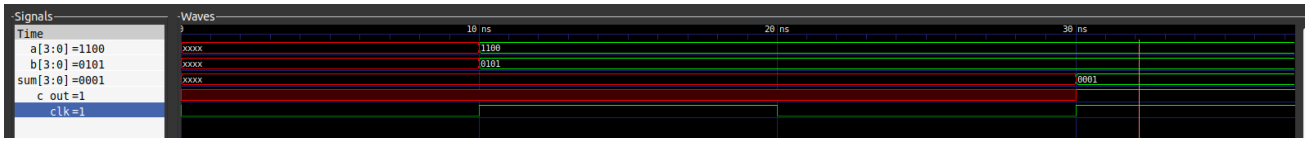
```
xor (pc[7], pc[4], pc[5]);
xor (pc[8], pc[6], pc[7]);
xor (c[3], pc[8], g[2]);
52
and (pc[9], pc[4], p[3]);
and (pc[10], pc[5], p[3]);
and (pc[11], pc[6], p[3]);
and (pc[12], p[3], g[2]);
xor (pc[13], pc[9], pc[10]);
xor (pc[14], pc[11], pc[13]);
xor (pc[15], pc[12], pc[14]);
xor (c[4], pc[15], g[3]);
endmodule
module pro(a,b,clk,c_out,sum,result);
input [3:0]a,b;
input clk;
output c_out;
output [3:0]sum;
output [4:0]result;
wire [3:0] ff_sum,ff_a,ff_b;
wire ff_c_out;
dff i10(a[0],clk,ff_a[0]);
dff i11(a[1],clk,ff_a[1]);
dff i12(a[2],clk,ff_a[2]);
dff i13(a[3],clk,ff_a[3]);
dff i20(b[0],clk,ff_b[0]);
dff i21(b[1],clk,ff_b[1]);
53
dff i22(b[2],clk,ff_b[2]);
dff i23(b[3],clk,ff_b[3]);
cla CLA(ff_a, ff_b, ff_c_out, ff_sum);
dff o1(ff_c_out, ~clk, c_out);
dff o20(ff_sum[0], ~clk, sum[0]);
dff o21(ff_sum[1], ~clk, sum[1]);
dff o22(ff_sum[2], ~clk, sum[2]);
dff o23(ff_sum[3], ~clk, sum[3]);
assign result = {c_out, sum};
endmodule
module dff(D,clk,Q);
input D,clk;
output Q;
wire w1, w2, w3, w4, w5, w6, w7, w8, d_bar, clk_bar;
not (d_bar, D);
not (clk_bar, clk);
nand (w1, D, clk);
nand (w2, d_bar, clk);
nand (w3, w1, w4);
nand (w4, w2, w3);
nand (w5, w3, clk_bar);
nand (w6, w4, clk_bar);
nand (w7, w5, w8);
54
nand (w8, w6, w7);
```

```
assign Q = w7;
endmodule
module cla(a,b,C_out,s);
input [3:0] a,b;
output C_out;
output [3:0]s;
wire [4:0] c;
wire [3:0] g,p;
wire [15:0] pc;
assign c[0] = 1'b0;
four_bit_xor x3(p,a,b);
four_bit_and x2(g,a,b);
four_bit_xor x1(s,p,c[3:0]);
carry x4(pc,p,c,g);
assign C_out = c[4];
endmodule
```

Testbench:

```
`timescale 1ns / 1ps
module pro_tb;
reg [3:0] a,b;
55
reg clk = 0;
wire c_out;
wire [3:0] sum;
wire [4:0] result;
pro uut (
.a(a),
.b(b),
.clk(clk),
.c_out(c_out),
.sum(sum),
.result(result)
);
always #10
clk = ~clk;
initial begin
$dumpfile("pro.vcd");
$dumpvars(0,pro_tb);
#10;
a = 4'b1100;
b = 4'b0101;
#30;
$finish;
end
endmodule
```

WAVEFORM:


Signals
Waves
Time
a[3:0] =1100
b[3:0] =0101
sum[3:0] =0001
c_out =1
clk =1
10 ns
20 ns
30 ns
xxxx
1100
xxxx
0101
xxxx
0001