CLA ADDER

Introduction

The carry-lookahead adder improves speed by reducing the amount of time required to determine carry bits.

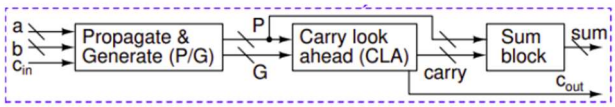

Figure 1: CLA Adder

The propagate and generate bits are given by:

$$p_i = a_i \oplus b_i$$

$$g_i = a_i . b_i$$

$$c_{i+1} = g_i + (p_i . c_i)$$

$$sum_i = p_i \oplus c_i$$

We will be representing c_{i+1} by expanding all the values and assuming $c_{in} = c_0 = 0$ (given). So, the three blocks can be:

- Each propagate and generate block will receive a_i and b_i as input and will give output as p_i and g_i. We shall use 4 such blocks, to simplify our design of the circuit.
- The sum block will consist of 4 XOR gates, it will receive $p_3p_2p_1p_0$ and $c_3c_2c_1c_0$ as input and give out sum_i.
- Our CLA block is the crucial element in the circuit, it will give out $c_3c_2c_1c_0$ and $c_{out} = c_4$ with inputs as P and G.

$$c_0 = 0$$
$$c_1 = g_0 + p_0 . c_0$$
$$c_2 = g_1 + p_1 . c_1$$
$$c_3 = g_2 + p_2 . c_2$$
$$c_4 = g_3 + p_3 . c_3$$

Substituting in all the carry values will give us:

$$c_0 = 0$$
$$c_1 = g_0$$
$$c_2 = g_1 + p_1 . g_0$$
$$c_3 = g_2 + p_2 . (g_1 + p_1 . g_0) = g_2 + g_1 . p_2 + g_0 . p_1 . p_2$$
$$c_4 = g_3 + p_3 . (g_2 + g_1 . p_2 + g_0 . p_1 . p_2) = g_3 + g_2 . p_3 + g_1 . p_2 . p_3 + g_0 . p_1 . p_2 . p_3$$

Simulating Blocks in NGSPICE

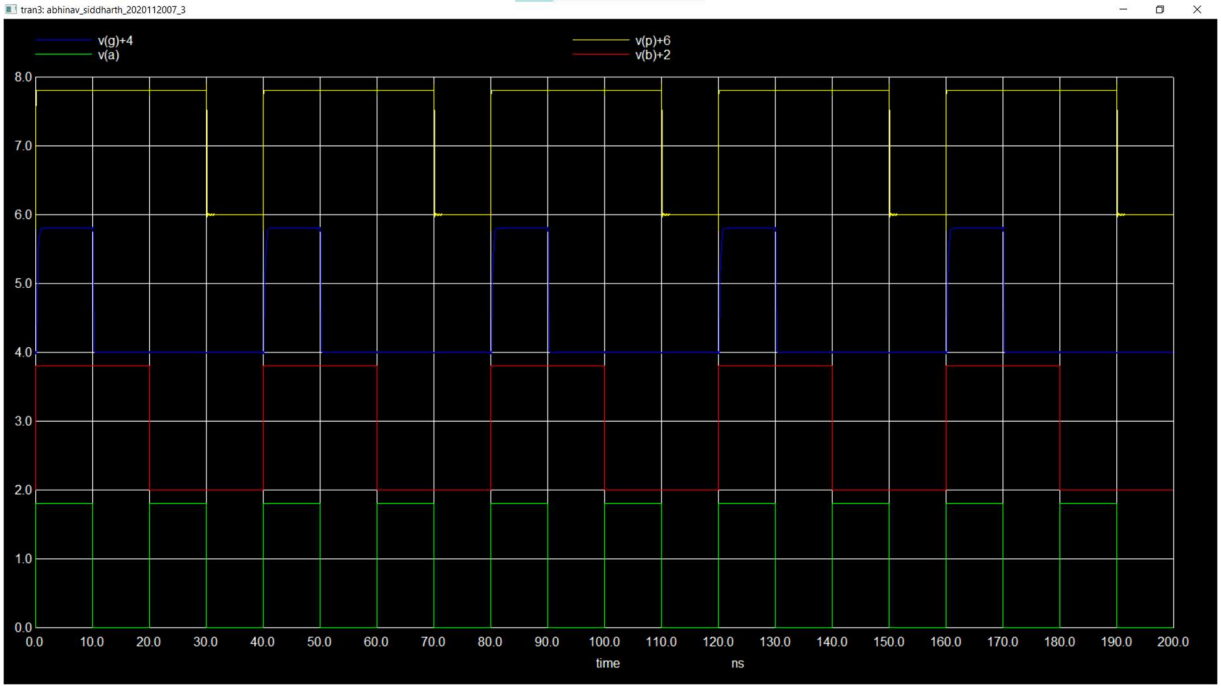

Figure 2: Propagate/Generate block output

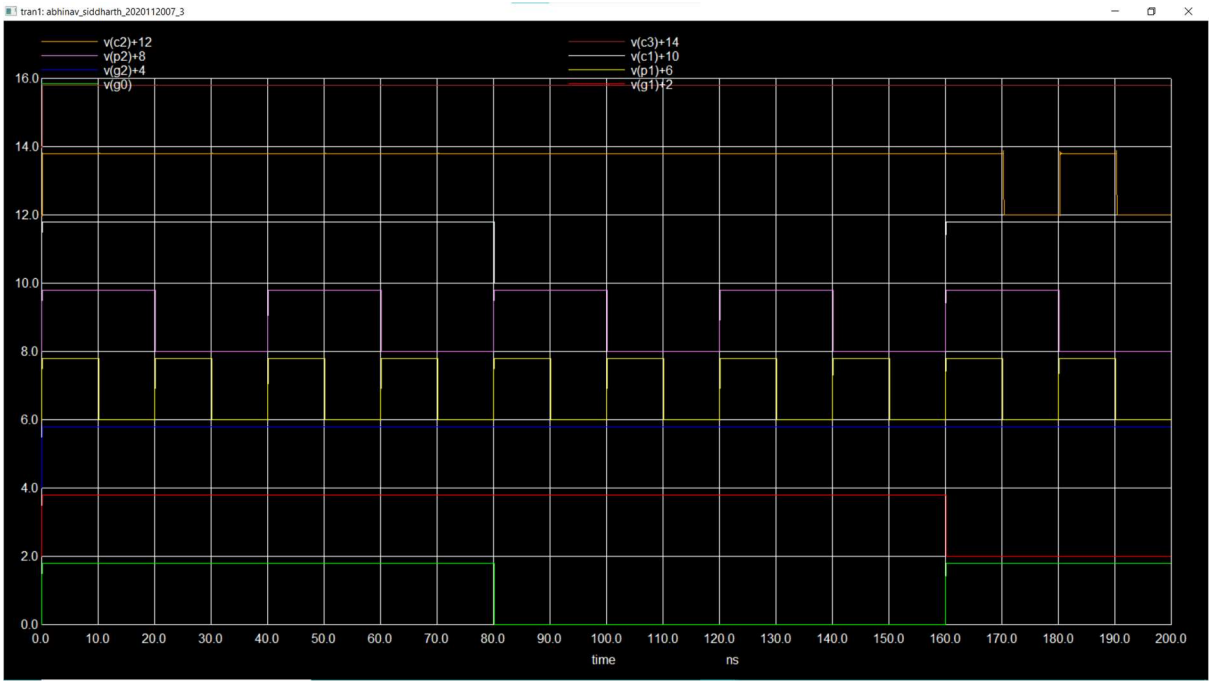

Figure 3: CLA block output

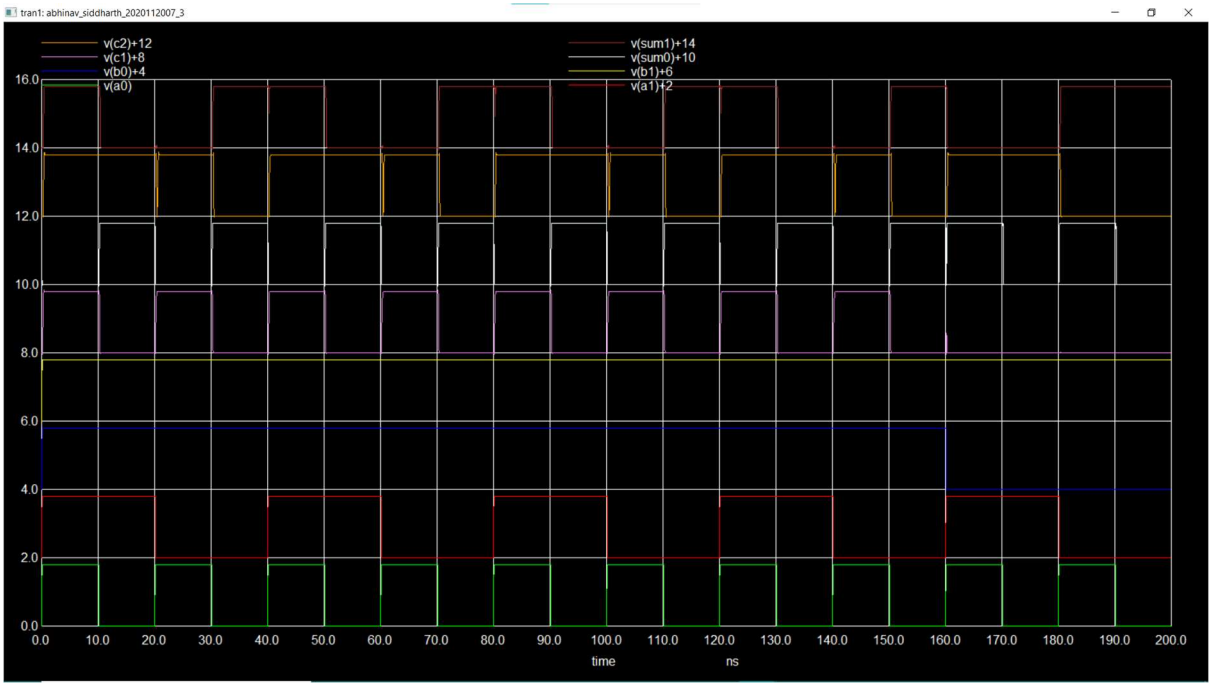

Figure 4: Complete CLA Adder output (c1 sum0 c2 sum1)

We have successfully verified the functionality of our circuit using NGSPICE.

Circuit Layouts

We have previously built AND, OR and XOR gates, we shall be using them in our CLA adder. Our previous XOR gate depended on A' and B' inputs as well, it has been updated with 2 new inverters such that it requires just A and B input.

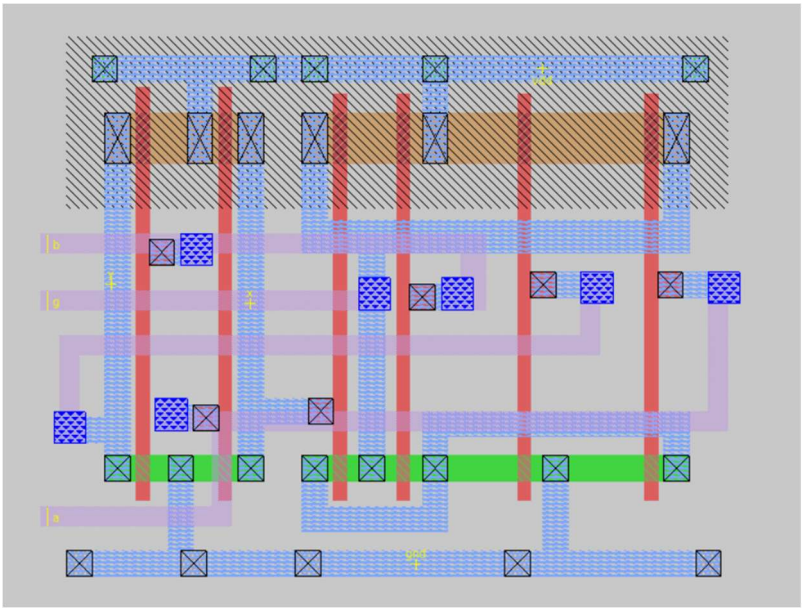

Figure 5: Updated XOR

Now, we can merge the AND and XOR to get one propagate/generate block.

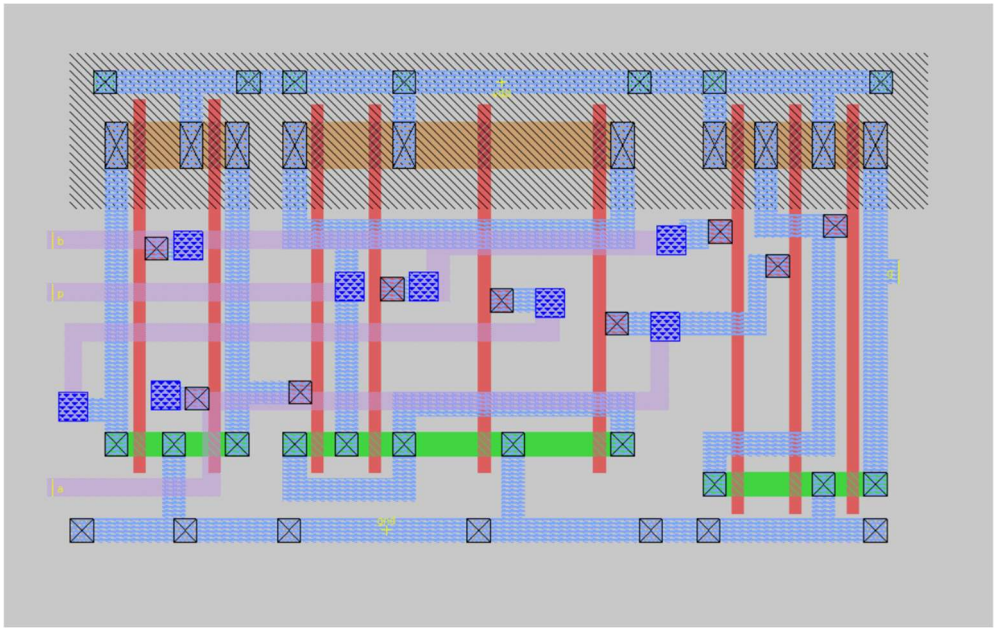

Figure 6: Single Propagate/Generate block

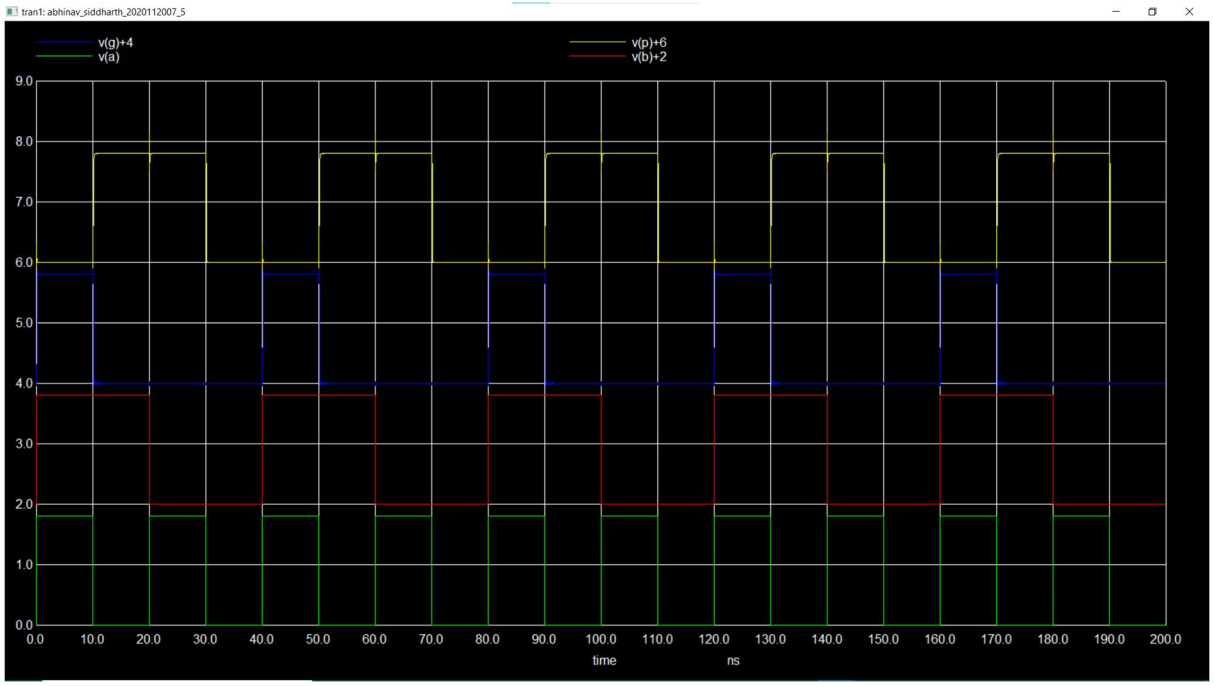

Figure 7: P/G block output

Now, we can merge 4 such instances to get the complete P/G block.

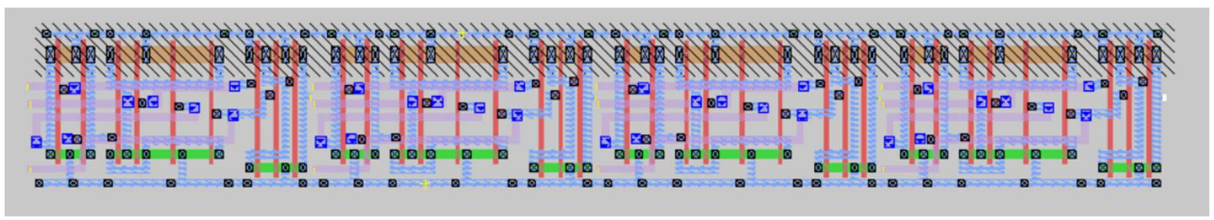

Figure 8: Complete P/G block

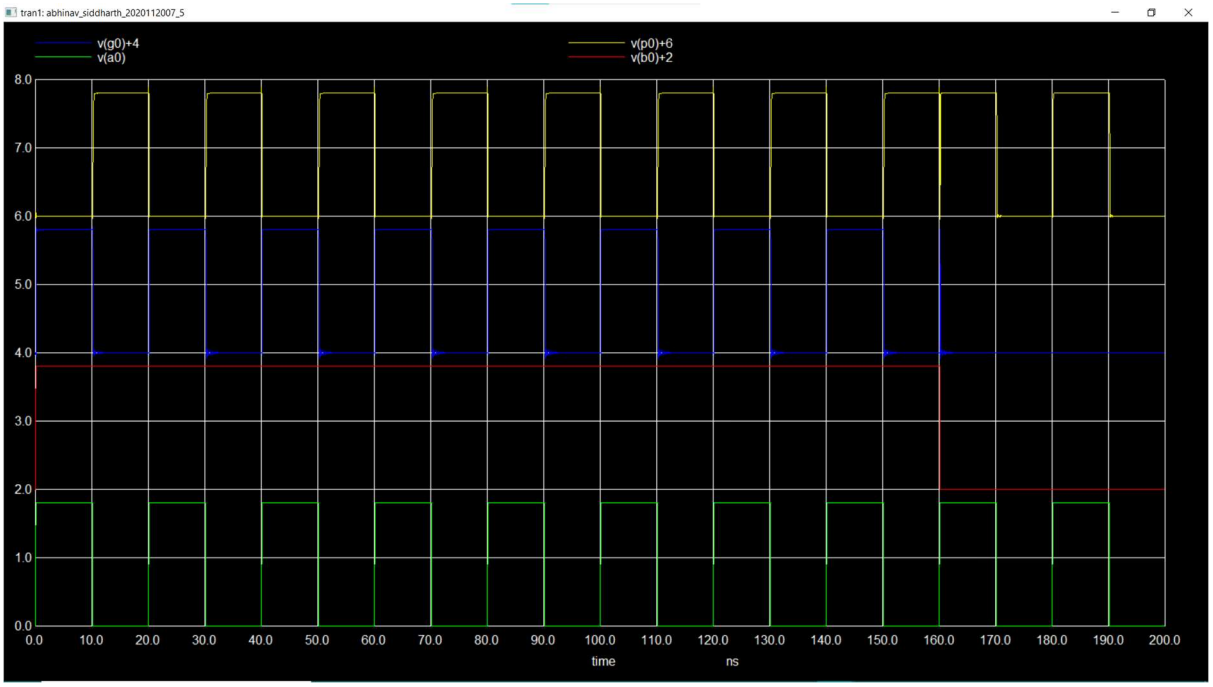

Figure 9: Single test case for complete P/G block

We can check that all our outputs are as desired and we have completed designing the P/G block. Now, moving on to the CLA block:

$$c_1 = g_0$$
$$c_2 = g_1 + p_1.g_0$$
$$c_3 = g_2 + g_1.p_2 + g_0.p_1.p_2$$
$$c_4 = g_3 + g_2.p_3 + g_1.p_2.p_3 + g_0.p_1.p_2.p_3$$

So, we will be needing a 3-input AND, 3-input OR, 4-input AND, finally a 4-input OR gate.

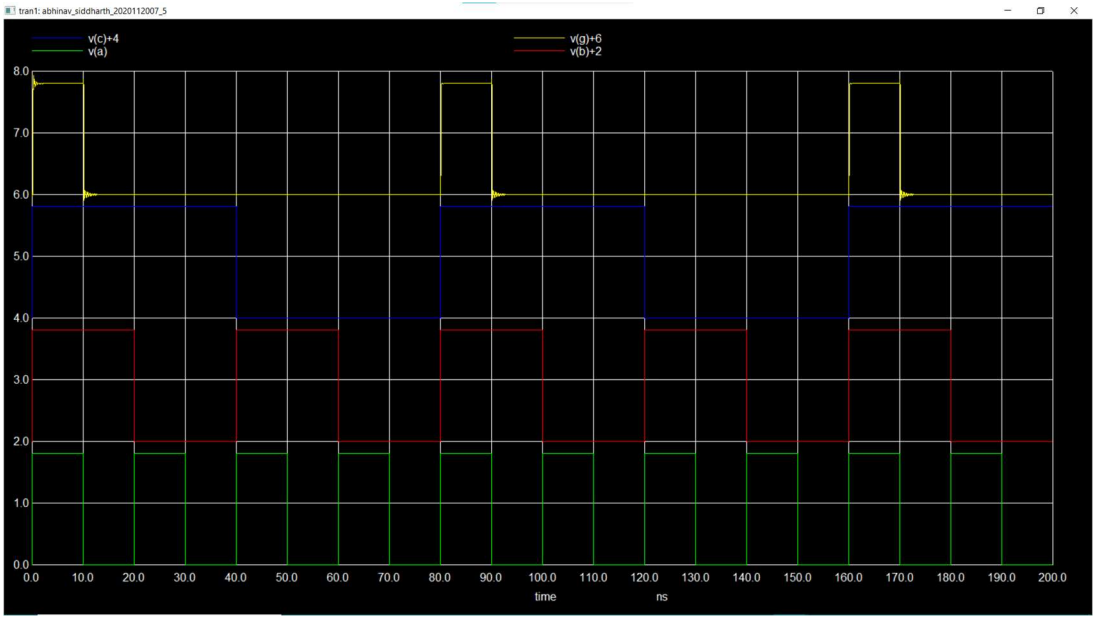

Figure 10: 3-input AND gate

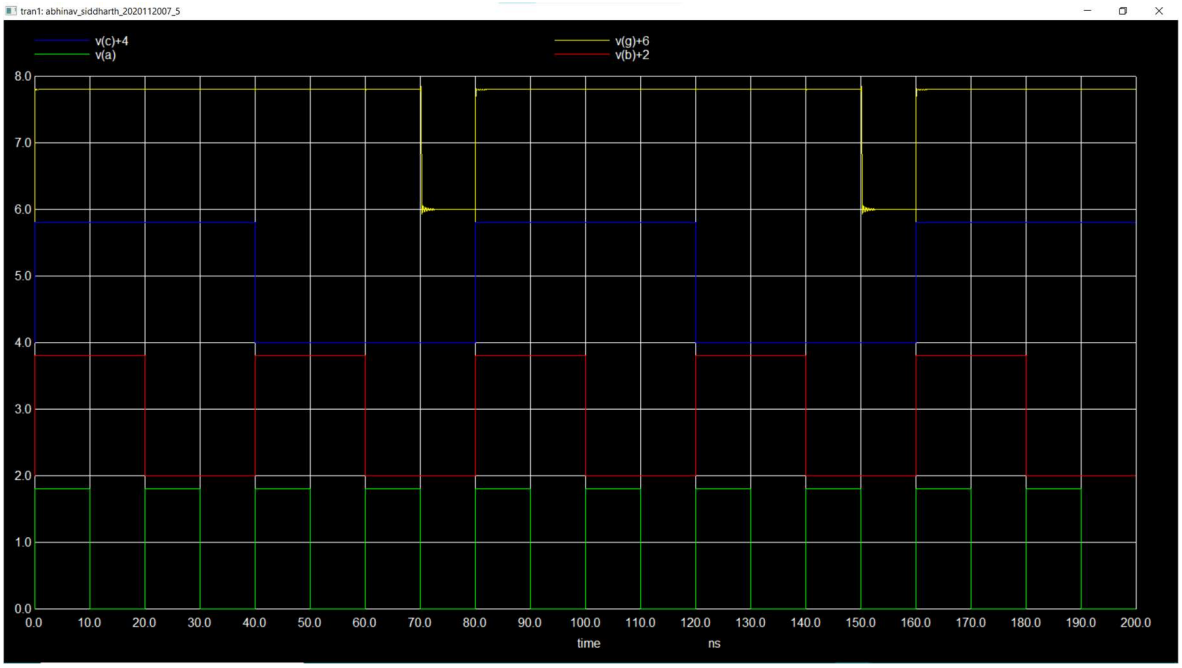

Figure 11: 3-input OR Gate

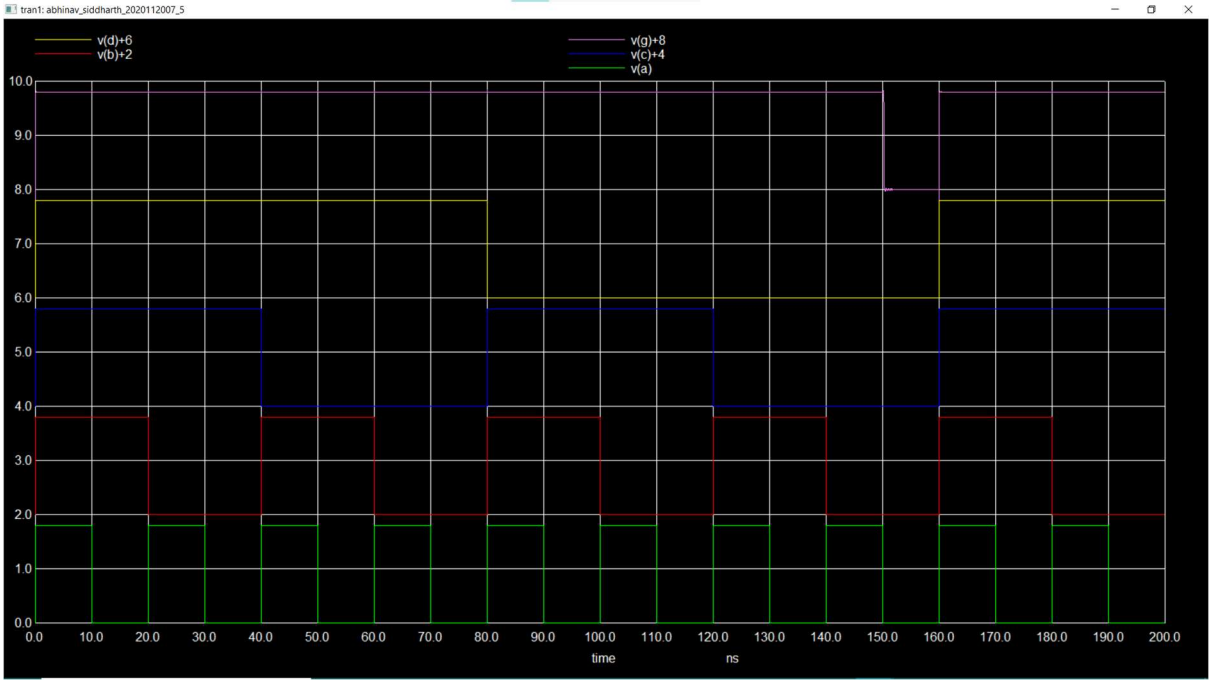

Figure 12: 4-input OR gate

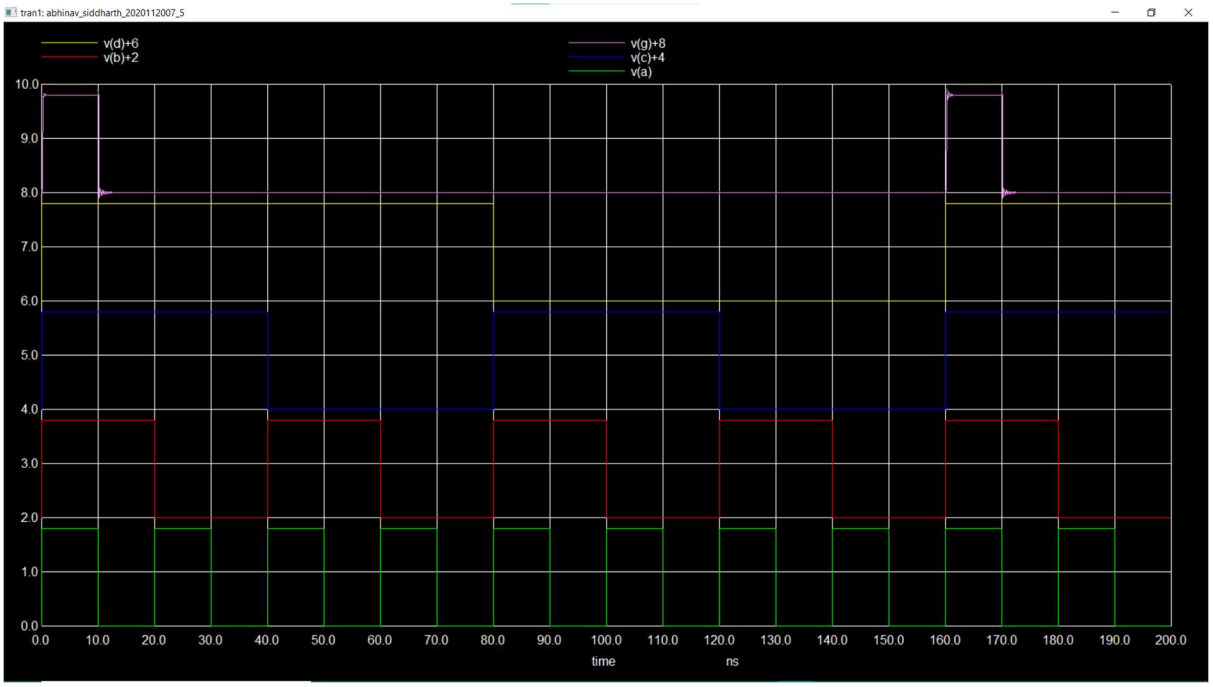

Figure 13: 4-input AND gate

We can now begin to merge our gates.

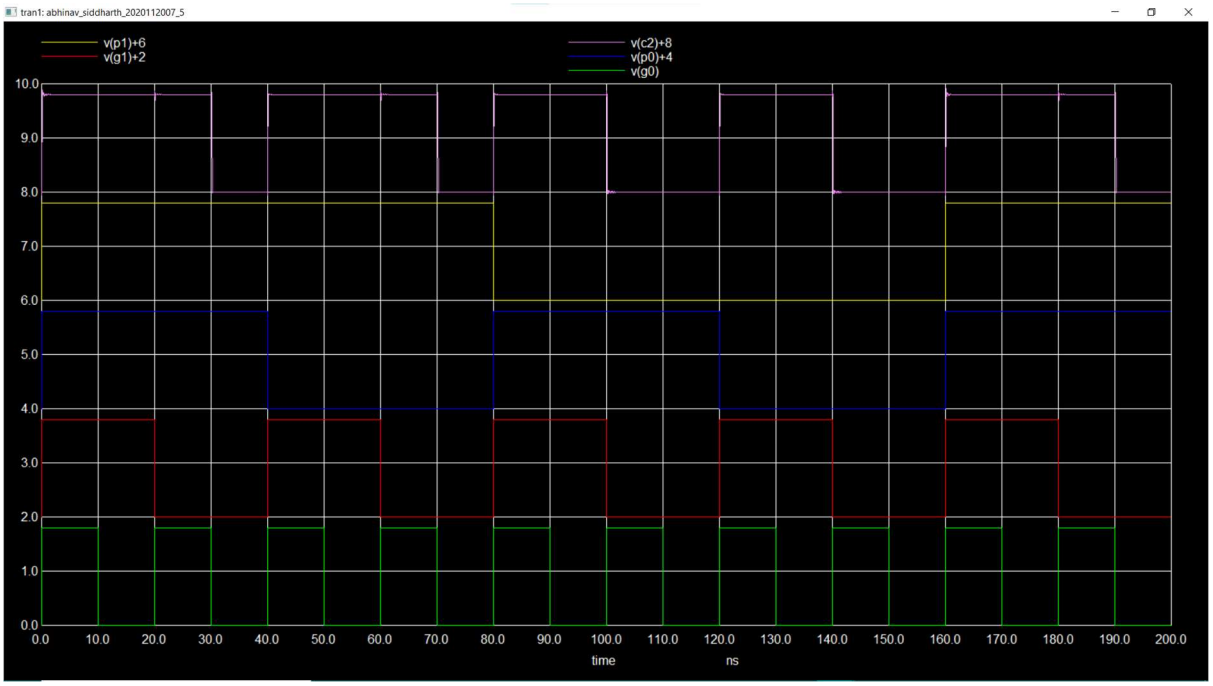

Figure 14: CLA for c2

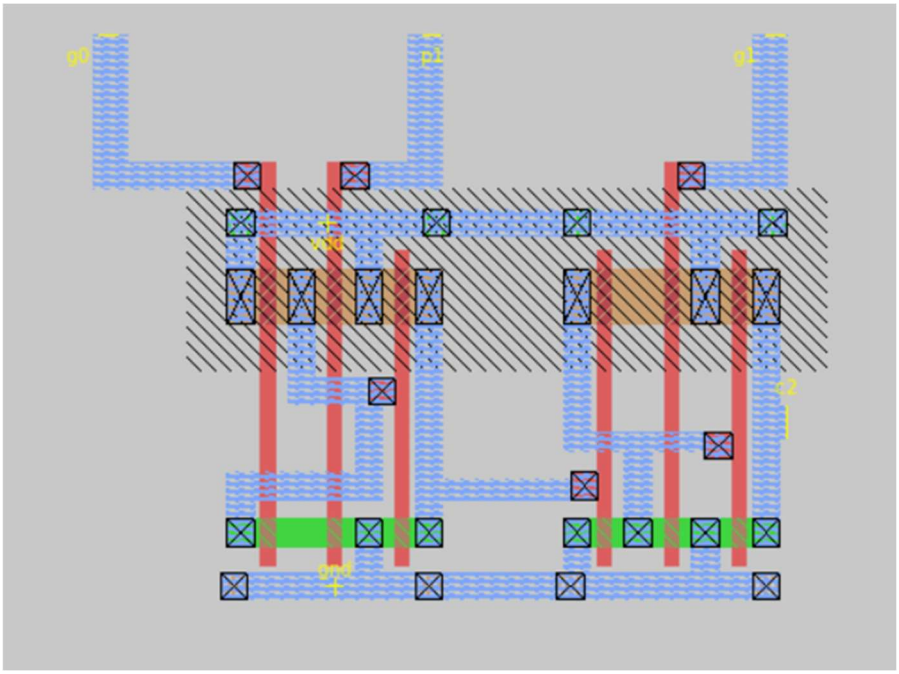

Figure 15: Circuit for c2

$$c_2 = g_1 + p_1.g_0$$

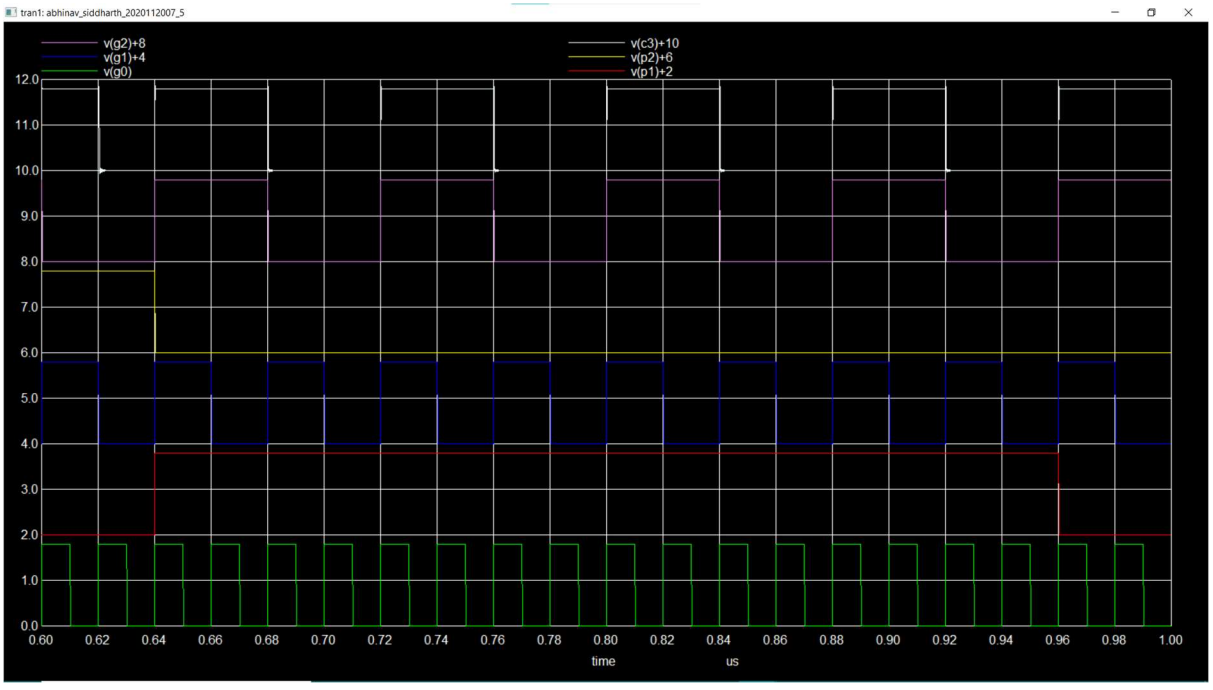

Figure 16: CLA for c3

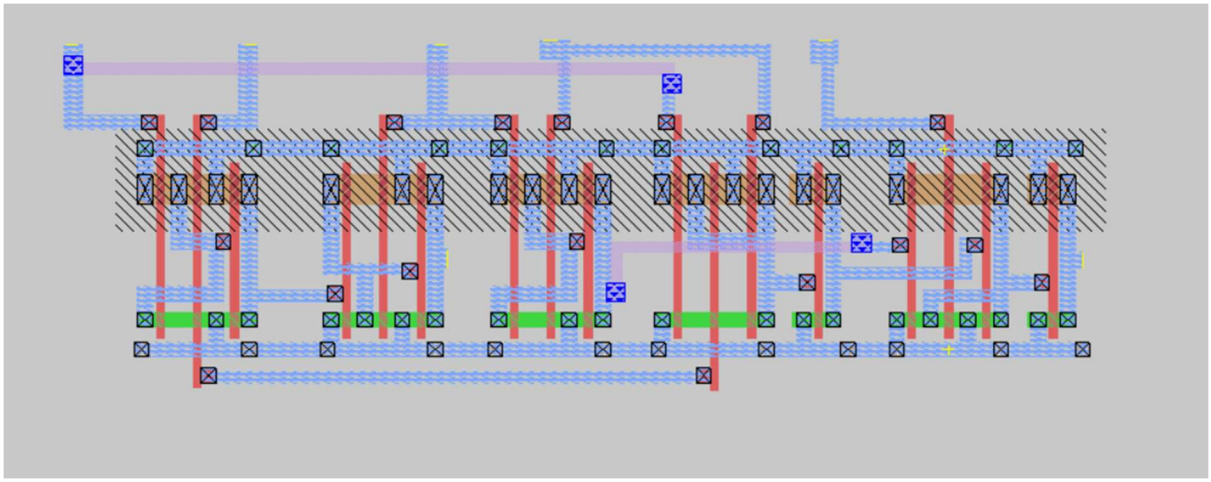

Figure 17: Circuit for c2 and c3

$$c_3 = g_2 + g_1.p_2 + g_0.p_1.p_2$$

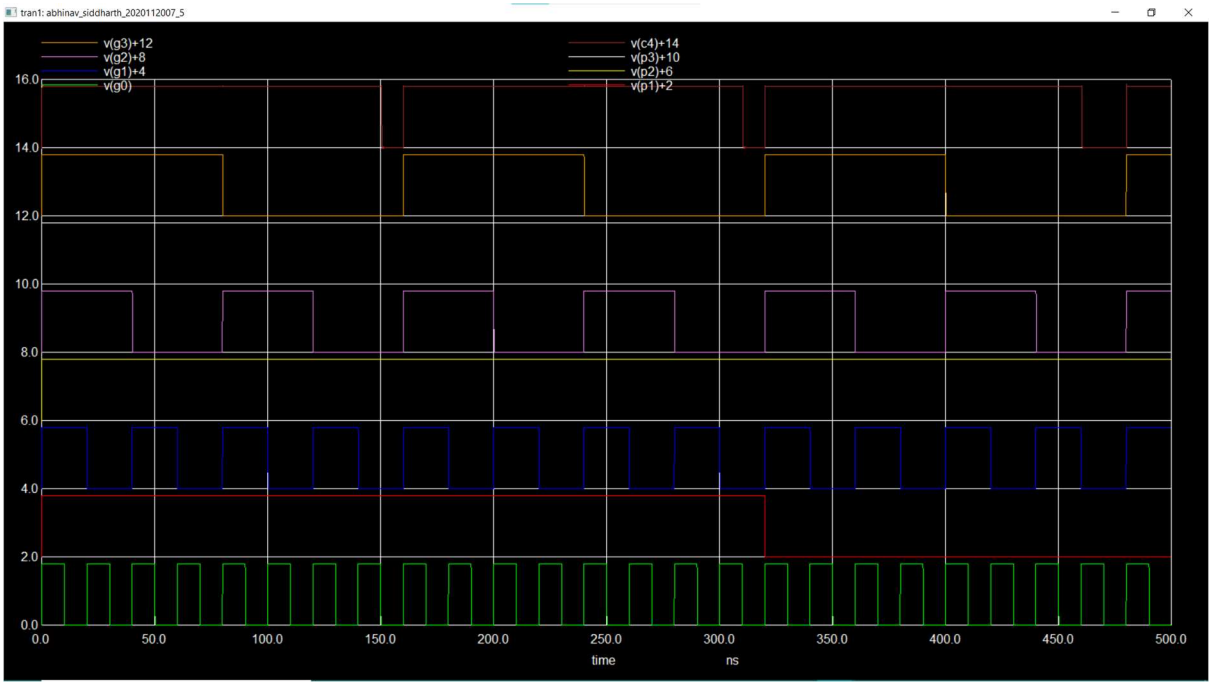

Figure 18: CLA for c4

$$c_4 = g_3 + g_2.p_3 + g_1.p_2.p_3 + g_0.p_1.p_2.p_3$$

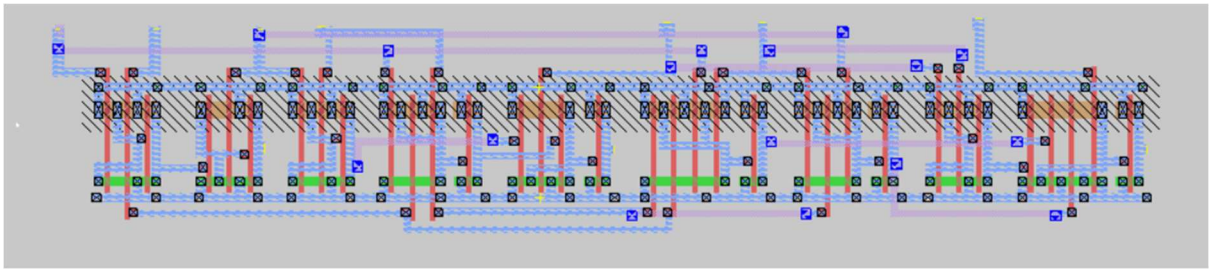

Figure 19: Full CLA block

Now, we have successfully tested all our carry outputs, and can verify that our CLA block is working as intended. We are now left with the last sum block to complete our CLA adder. Let us first merge both our PG and CLA blocks first to properly connect the inputs.

$$sum_i = p_i \oplus c_i$$

$$sum_0 = p_0$$

$$sum_1 = p_1 \oplus g_0$$

$$sum_2 = p_2 \oplus c_2$$

$$sum_3 = p_3 \oplus c_3$$

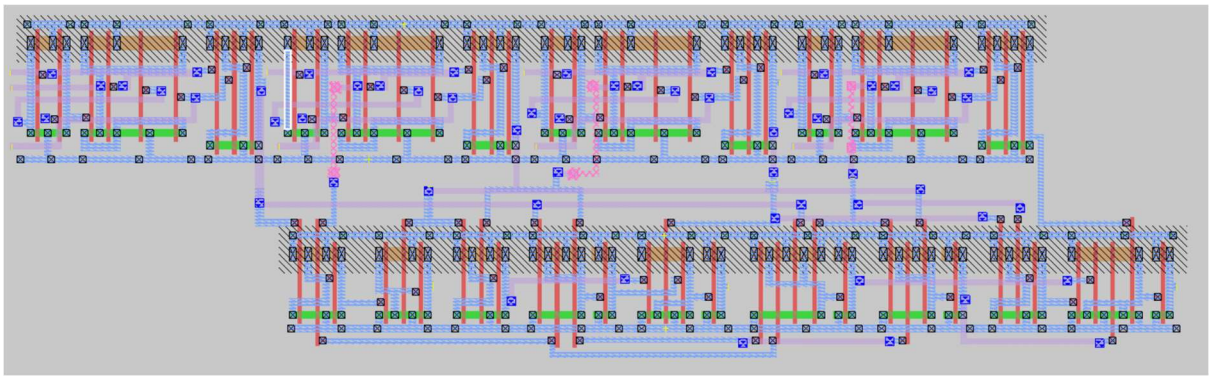

Figure 20: PG and CLA block combined

Now, all we must do is add our XOR blocks and our CLA Adder will be complete.

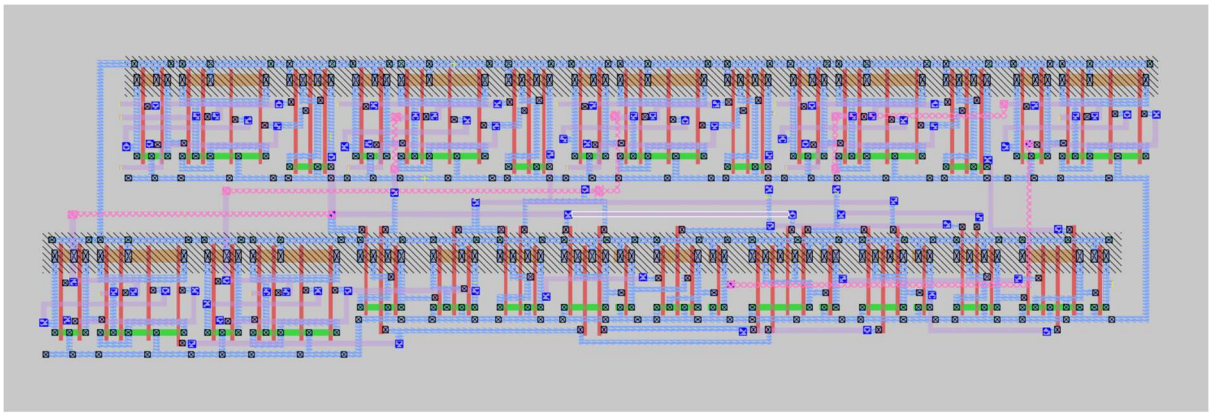

Figure 21: CLA Adder

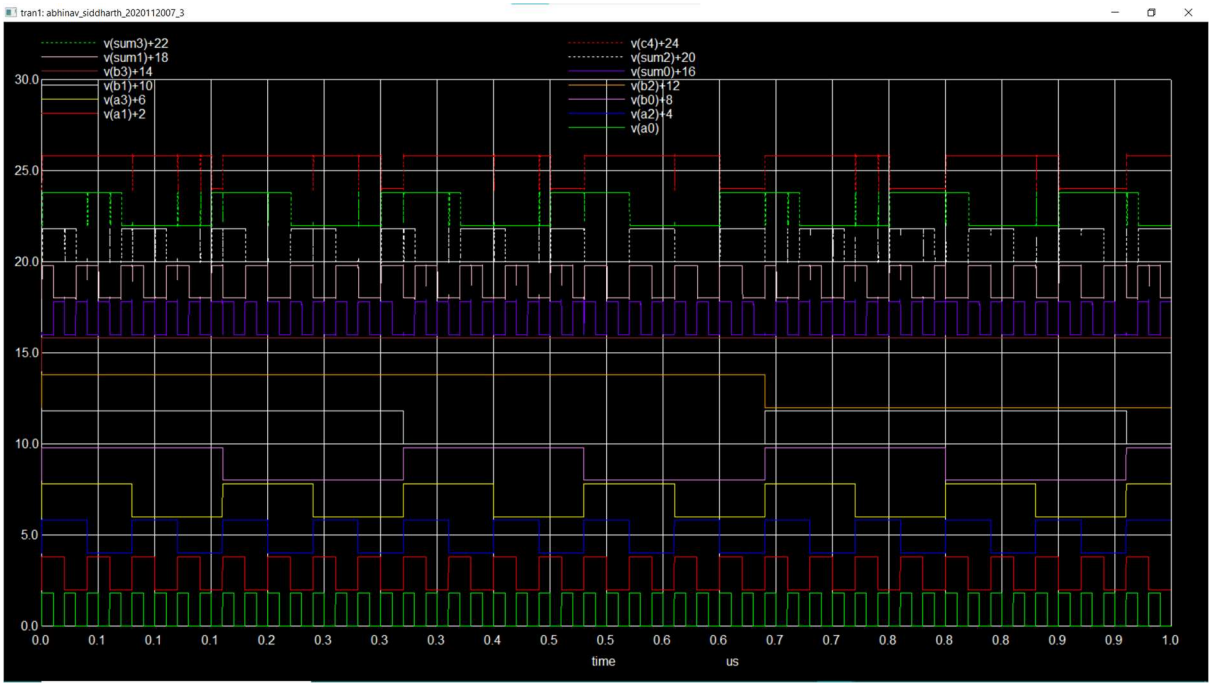

Figure 22: Pre-layout simulation

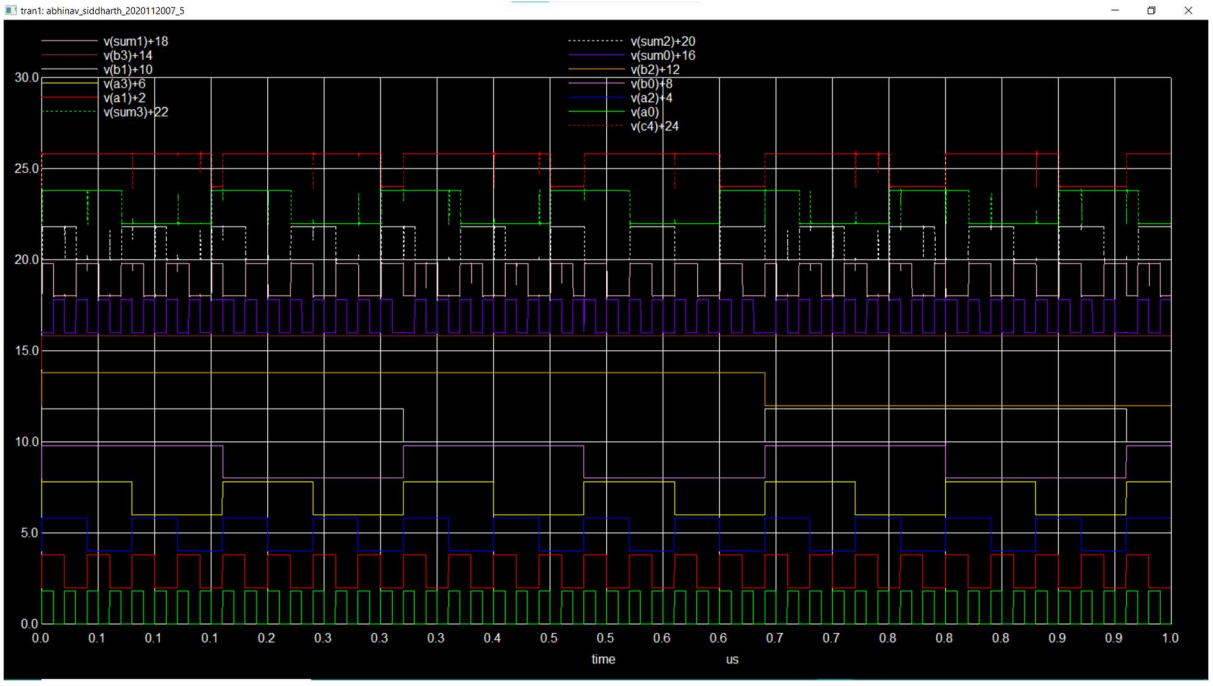

Figure 23: Layout simulation

Both the pre-layout and the MAGIC simulation give same results, hence our CLA adder is working correctly.

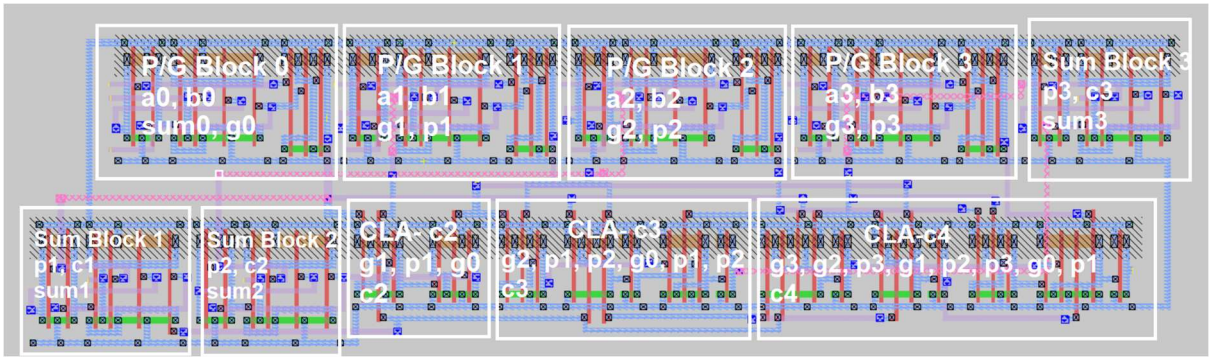

Figure 24: Circuit

Note: c1 = g0.

Verilog

```
module PG(p, g, a, b);
        input a, b;
        output p, g;
        and G1 (g, a, b);
        xor G2 (p, a, b);
endmodule

module CLA(g0, p0, g1, p1, g2, p2, g3, p3, c0, c1, c2, c3, c4);
        input g0, p0, g1, p1, g2, p2, g3, p3, c0;
        output c1, c2, c3, c4;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10;
        and G1(t1, p0, c0);
        or G2(c1, t1, g0);

        and G3(t2, c0, p0, p1);
        and G4(t3, g0, p1);
        or G5(c2, g1, t2, t3);

        and G6(t4, c0, p0, p1, p2);
        and G7(t5, g0, p1, p2);
        and G8(t6, g1, p2);
        or G9(c3, g2, t4, t5, t6);

        and G10(t7, c0, p0, p1, p2, p3);
        and G11(t8, g0, p1, p2, p3);
        and G12(t9, g1, p2, p3);
        and G13(t10, g2, p3);
        or G14(c4, g3, t7, t8, t9, t10);
endmodule

module SUM(p0, c0, p1, c1, p2, c2, p3, c3, sum0, sum1, sum2, sum3);
        input p0, c0, p1, c1, p2, c2, p3, c3;
        output sum0, sum1, sum2, sum3;
        xor G1(sum0, p0, c0);
        xor G2(sum1, p1, c1);
        xor G3(sum2, p2, c2);
        xor G4(sum3, p3, c3);
endmodule

module CLA_FULL(a3, b3, a2, b2, a1, b1, a0, b0, c0, c4, sum3, sum2, sum1, sum0);
        input a3, b3, a2, b2, a1, b1, a0, b0, c0;
        output c4, sum3, sum2, sum1, sum0;
        wire g0, p0, g1, p1, g2, p2, g3, p3, c1, c2, c3;
        PG B1(p0, g0, a0, b0);
        PG B2(p1, g1, a1, b1);
        PG B3(p2, g2, a2, b2);
        PG B4(p3, g3, a3, b3);
        CLA B5(g0, p0, g1, p1, g2, p2, g3, p3, c0, c1, c2, c3, c4);
        SUM B6(p0, c0, p1, c1, p2, c2, p3, c3, sum0, sum1, sum2, sum3);
endmodule
```

Figure 25: Verilog Structural Code

```
PS C:\Users\abhin\Documents\ngspice\bin> iverilog -o hdl_cla hdl_cla_tb.v hdl_cla.v
PS C:\Users\abhin\Documents\ngspice\bin> vvp hdl_cla
VCD info: dumpfile hdl_cla.vcd opened for output.
                   0ns: a = 1111 b = 1111 c4+sum=11110
                  10ns: a = 1110 b = 1111 c4+sum=11101
                  20ns: a = 1101 b = 1111 c4+sum=11100
                  30ns: a = 1100 b = 1111 c4+sum=11011
                  40ns: a = 1011 b = 1111 c4+sum=11010
                  50ns: a = 1010 b = 1111 c4+sum=11001
                  60ns: a = 1001 b = 1111 c4+sum=11000
                  70ns: a = 1000 b = 1111 c4+sum=10111
                  80ns: a = 0111 b = 1111 c4+sum=10110
                  90ns: a = 0110 b = 1111 c4+sum=10101
                 100ns: a = 0101 b = 1111 c4+sum=10100
                 110ns: a = 0100 b = 1111 c4+sum=10011
                 120ns: a = 0011 b = 1111 c4+sum=10010
                 130ns: a = 0010 b = 1111 c4+sum=10001
                 140ns: a = 0001 b = 1111 c4+sum=10000
                 150ns: a = 0000 b = 1111 c4+sum=01111
                 160ns: a = 1111 b = 1111 c4+sum=11110
                 170ns: a = 1110 b = 1110 c4+sum=11100
                 180ns: a = 1101 b = 1110 c4+sum=11011
                 190ns: a = 1100 b = 1110 c4+sum=11010
                 200ns: a = 1011 b = 1110 c4+sum=11001
                 210ns: a = 1010 b = 1110 c4+sum=11000
                 220ns: a = 1001 b = 1110 c4+sum=10111
                 230ns: a = 1000 b = 1110 c4+sum=10110
                 240ns: a = 0111 b = 1110 c4+sum=10101
                 250ns: a = 0110 b = 1110 c4+sum=10100
                 260ns: a = 0101 b = 1110 c4+sum=10011
                 270ns: a = 0100 b = 1110 c4+sum=10010
```

Figure 26: Few outputs of the Verilog code

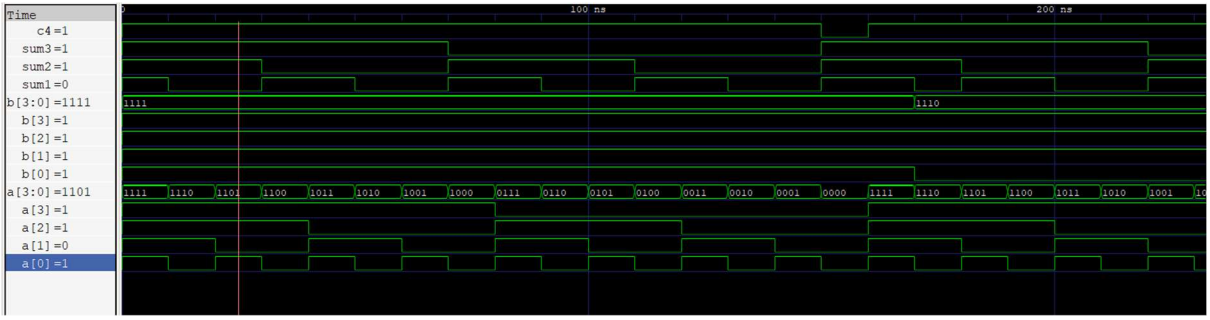

Figure 27: GTKWave output of the Verilog code

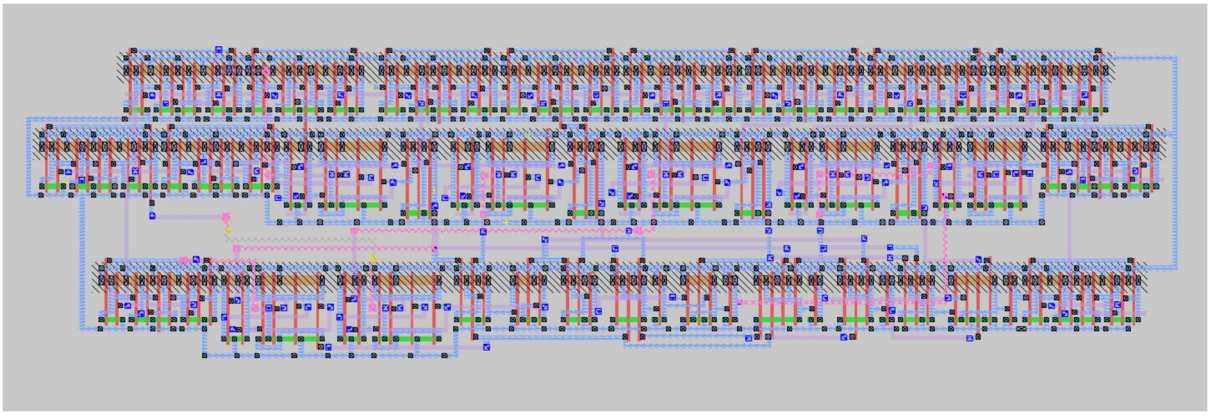

Figure 28: CLA Adder after addition of flip-flops