VLSI-Self PROJECT

DESIGN AND IMPLEMENTATION OF A 4 BIT CARRY-LOOKAHEAD ADDER

Name:
Vishnu V Saseendran

4 BIT CARRY-LOOKAHEAD ADDER

OBJECTIVE:

The objectives of this project are to design and implement a 4-bit Carry-Lookahead Adder (CLA) using the LT-spice tool, verify its functional correctness using SPICE simulations, discuss the effect of carry propagation and ripple in each case with respect to conventional carry ripple adder.

AIM:

To design and implement a 4-bit carry-lookahead adder

COMPONENTS REQUIRED

- C5NNMOS
- C5NPMOS
- Voltage sources(as input signals)

- Tools Used: LTSpice

METHODOLOGY:

Introduction:

Carry-lookahead adder is a practical design with a reduced delay at the price of more complex hardware. The carry lookahead design can be obtained by a transformation of the ripple carry design into a design in which the carry logic over fixed groups of bits of the adder are reduced to two-level logic. The Addition is the very basic operation in digital electronics, but most of the digital component takes time to propagate the carry signal and hence the use of CLA can reduce the time required for the carry bits.

Carry Look Ahead Adder:

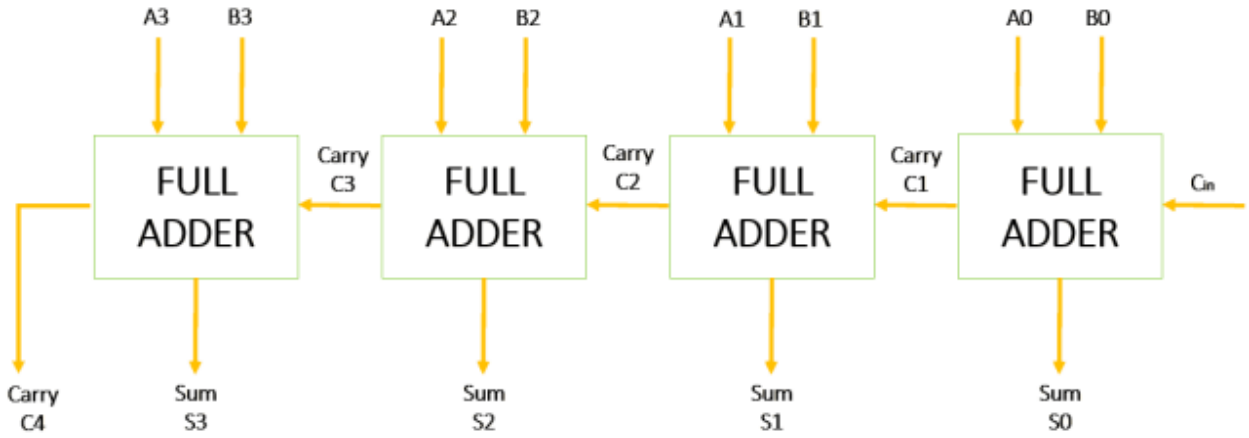
A carry look-ahead adder (CLA) is an electronic adder used for binary addition. Due to the quick additions performed, it is also known as a fast adder. The CLA logic uses the concepts of generating and propagating carries. The CLA adder is faster than the Ripple Carry Adder.

Need of Carry Look Ahead Adder:

In parallel adders, the addition operation occurs when both values are required to perform addition, i.e., the augend and the addend are present at the same time. In parallel adder circuits, the carry output of one stage serves as the carry input of the succeeding stage, thus being called the 'ripple' carry adder.

For example, when we throw a stone in the water, The water around the stone-hit area flows ahead in the form of ripples. Similarly, in the Ripple Carry Adder, the Carry bit 'ripples' forward into the system.

When we consider a 4-bit ripple carry adder, the augend and the addend are readily available. All that is left for the full adder to begin working is the input carry. This carry is given as input to the first full adder. But the remaining full adders require the carry-output of the previous adder to be input in their systems. In other words, a full adder can't generate the sum and carry of the respective block unless the input carry is known.



4-bit Ripple Carry Adder:

In general, let us represent each of these full adder circuit blocks with variables i, i+1, …..

If we analyze the system carefully, the (i+1)th block waits for the ith block to generate a carry. The sum S3 generated depends on the availability of input signals A3 and B3, but the Carry C4 does not get generated until the previous carry C3 is available, which in turn cannot be present without the presence of C2, which is again dependent on the previous carries C1 and Cin. As a result, the entire operation has a lot of delay within the system, which can be called the carry propagation delay.

The total propagation time is equal to the sum of all the individual propagation delays of each Full Adder block present in the system. Undeniably, the propagation time of the system increases if the number of

Full Adder blocks (i, in this case) increases, i.e., if we perform the addition of larger numbers (8-bit or 16-bit).
To overcome this issue, We have two solutions :

1. Introduce faster gates in the system with smaller delays. But we cannot keep adding more and more gates in the circuit; this would increase the size of the overall prototype and, in turn, cause more power dissipation and lead to faster damage of the system.
2. Modify the circuit in such a way that, even though the complexity increases by a small degree, it performs the same function more efficiently and does not depend on the carry of the previous system.

So the second solution leads to the invention of the Carry-ahead adder.

Working of a Carry Look Ahead Adder

In a full adder. We have the input signals A, B, and Cin. If we consider the addition of these three variables in every possible case, we get a truth table like the one below.

A	B	C_{in}	Sum	Carry	Condition
0	0	0	0	0	No Carry Generated
0	0	1	1	0	
0	1	0	1	0	
0	1	1	0	1	Carry Propagated
1	0	0	1	0	No Carry
1	0	1	0	1	Carry Propagated
1	1	0	0	1	Carry Generated
1	1	1	1	1	

Truth Table of Full Adder

On analyzing the truth table, we can see that the Carry is 1 when

1. Either the value of A or B is one, as well as Cin, is 1, or
2. Both A and B have the value 1.

The idea consists of adding two new variables, Carry Generate (Gi) and Carry Propagate (Pi).

For case 1, we see that an output carry is propagated, when we give an input carry. We will refer to this with Pi. So, the mathematical expression of Pi can be represented as :

$$Pi = Ai \oplus Bi$$

While considering case 2, we see that an output carry is generated when both inputs, A and B, are high, regardless of the value of the input carry. We will refer to this output carry as Gi. Thus, we can mathematically express Gi as :

$$Gi = Ai \cdot Bi$$

Originally, for a full adder we have the following equations:

$$Sum = A \oplus B \oplus Ci$$

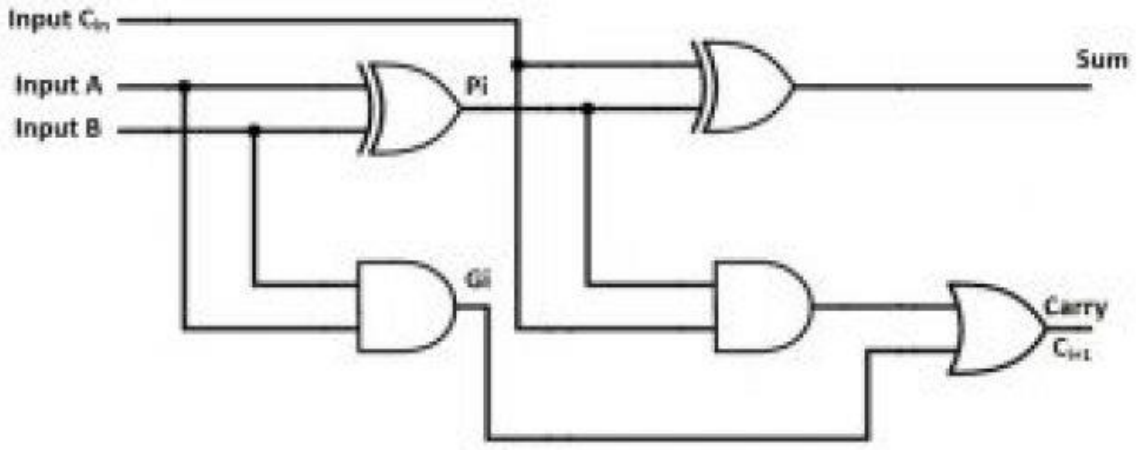
$$Carry = Ci(A \oplus B) + AB$$

Thus, we can rewrite the equations of the full adder in terms of Carry Propagate (Pi) and Carry Generate (Gi) as :

$$Sum = Pi \oplus Ci$$

$$Carry = Pi \cdot Ci + Gi$$

The equations of Sum and Carry can be represented by a logic circuit given below.



Logic Circuit

Principle of Operation of the CLA:

The CLA is a very fast adder designed to minimize the delay caused by carry propagation in basic adders. The CLA makes use of the fact that, at each bit position in the addition, it can be determined if a carry will be generated at that bit, or if a carry will be propagated through that bit. The CLA uses two signals; Propagate (P) and Generate (G). The truth table for the binary addition is shown in Table 1.

Table 1: Truth Table for Binary addition

Ai	Bi	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Consider the full adder circuit shown above with the corresponding truth table. We define two variables as 'carry generate' G_i and 'carry propagate' $P_i$then,

$$P_i = A_i \otimes B_i$$
$$G_i = A_i * B_i$$

where produces the carry when both G_i, are 1 regardless of the input carry. P_i is associated with the propagation of carry from C_i to C_{i+1}

The carry output Boolean function of each stage in a 4 stage carry look-ahead adder can be expressed as

$$C_1 = G_0 + P_0 C_{in}$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

From the above Boolean equations, we can observe that C_4 does not have to wait for $C_3$and C_2 to propagate but actually $C_4$is propagated at the same time as $C_3$and C_2. Since the Boolean expression for each carry, the output is the sum of products, so these can be implemented with one level of AND gates followed by an OR gate

We can see that there is no dependency on any intermediate Carry values in any of the equations. On solving the equations, we see that only the input Carry Cin is required to calculate all the Sum and Output

Carry values. This resolves the issue faced by the Ripple Carry Adder's dependency on the intermediate carry values.

Thus, the entire operation works faster for higher-order bits, when compared to the Ripple Carry Adder. This is the reason why the CLA Adder is also called a Fast Adder.

Cascading Carry Look Ahead Adders

Cascading CLA Adders can make the addition of higher-order bits possible. To construct 8 bit, 16 bit, and 32-bit parallel adders, we can cascade multiple 4-bit Carry Look Ahead Adders with the carry logic. A 16 bit CLA adder can be constructed by cascading four 4 bit adders with two extra gate delays, while a 32 bit CLA adder is formed when two 16 bit adders are cascaded to form one system.

Advantages of Carry Look Ahead Adders

- CLA Adders generate the carry-in for each full adder simultaneously, by using simplified equations involving Pi, Gi, and Cin.
- This system reduces the propagation delay. This is because the output carry at any stage is dependent only on the first Carry signal given at the input.
- It is the fastest adder when compared to other addition mechanisms.

Disadvantages of Carry Look Ahead Adders

- The carry-lookahead adder circuit gets more complicated as the number of variables increases.
- The circuit for a carry-lookahead adder is expensive as it involves more hardware.
- As the number of variables increases, the circuit implements more hardware. Thus, when the carry-lookahead adder is implemented as an IC, the area is bound to increase.

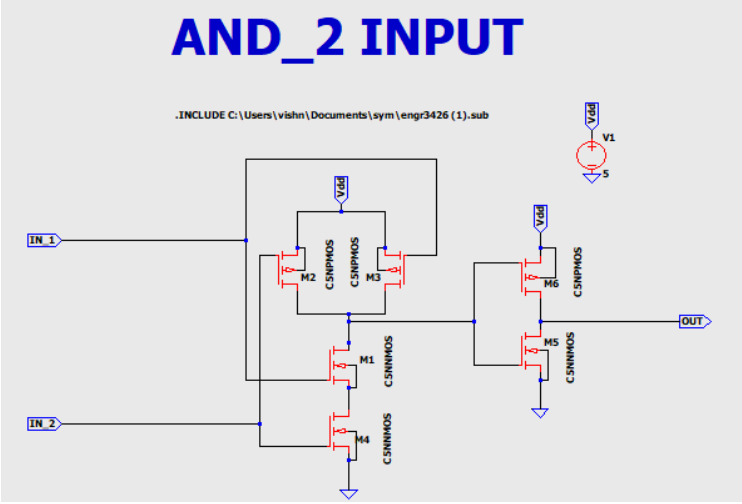
Uses of Carry Look Ahead Adders

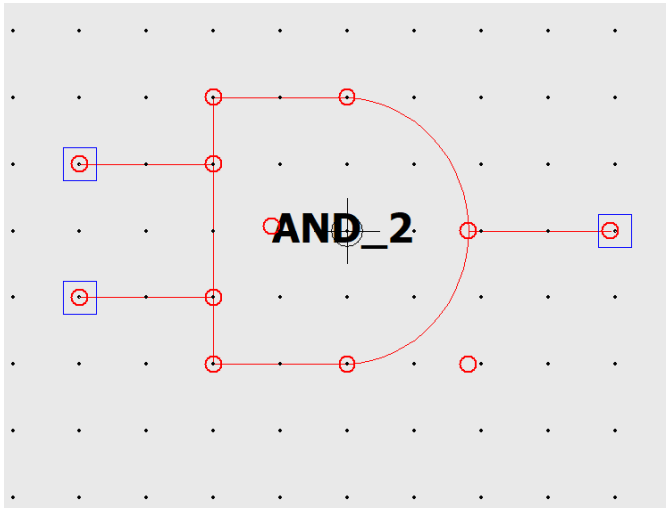
Different bit-adder configurations of the fast-response carry-lookahead adders are manufactured on integrated circuitry nowadays. Apart from the circuits, there are also individual carry generator ICs that are manufactured, and we need to make the required connections to perform the quick addition function.

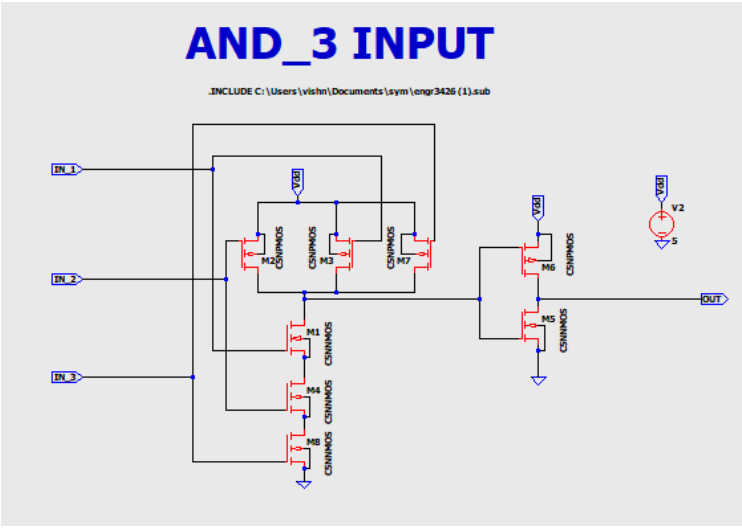
Software Implementation of the 4-Bit CLA

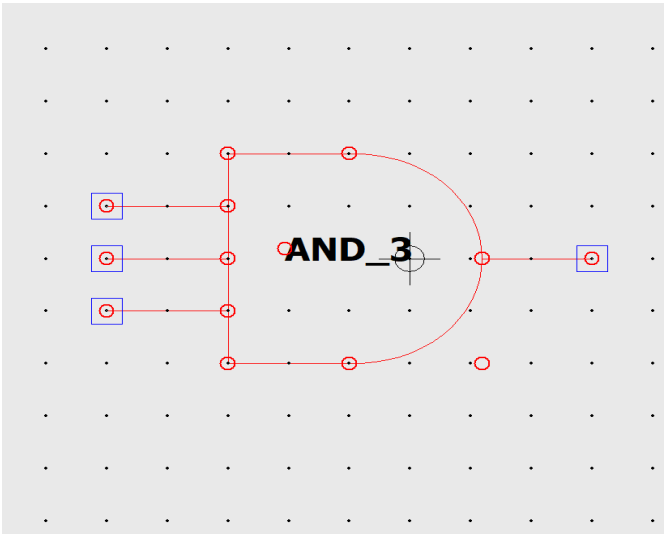
Based on the equations and principles outlined in the previous section, the 4-bit CLA was implemented in Electric software integrated with LTSPICE. First, schematics of logic gates required for the implementation of the design were made in electric software. The figures below represent the basic logic gates implemented in LtSpice software for this project.

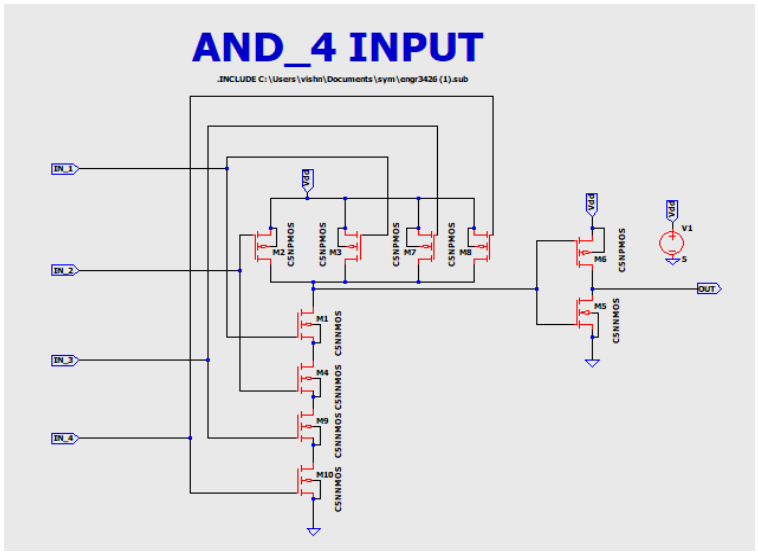
For this project, we implemented each basic gate using CMOS (C5NNMOS, C5NPMOS)and saved it in ".asc " format inside a folder Mysymbols at location "\Documents\LTspiceXVII\lib\sym\Mysymbols"(This is the default location of symbols in LTSpice software). And after that, we added a ".sym" file containing the gate symbol that was also saved at the same location with the same name as that of the asc file . That was how we created all the necessary gates needed.

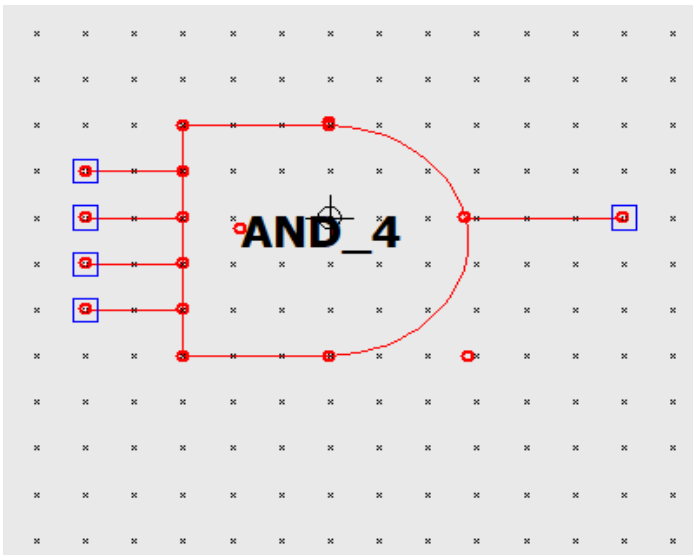


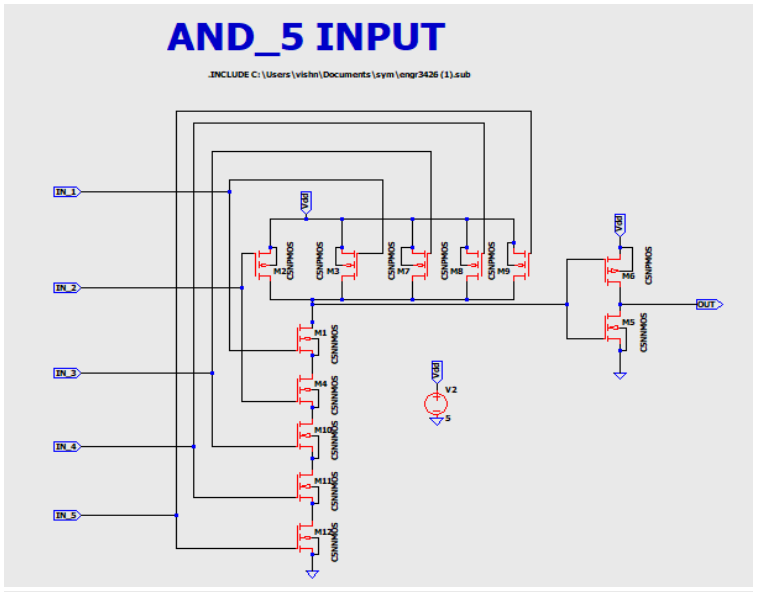


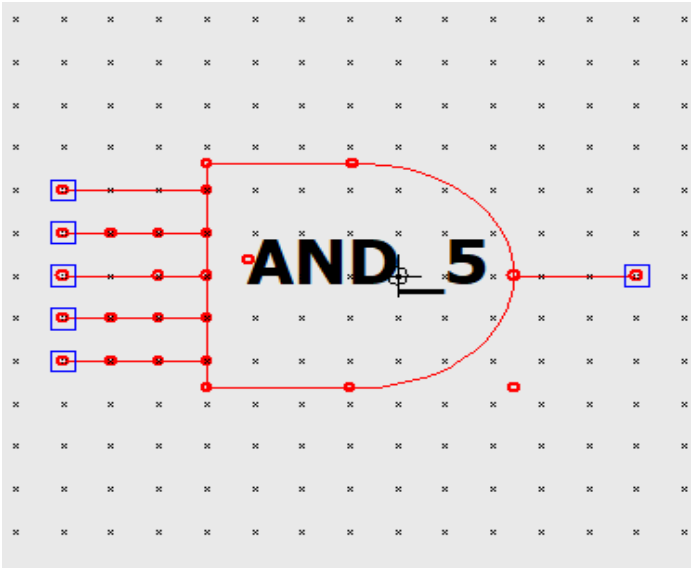


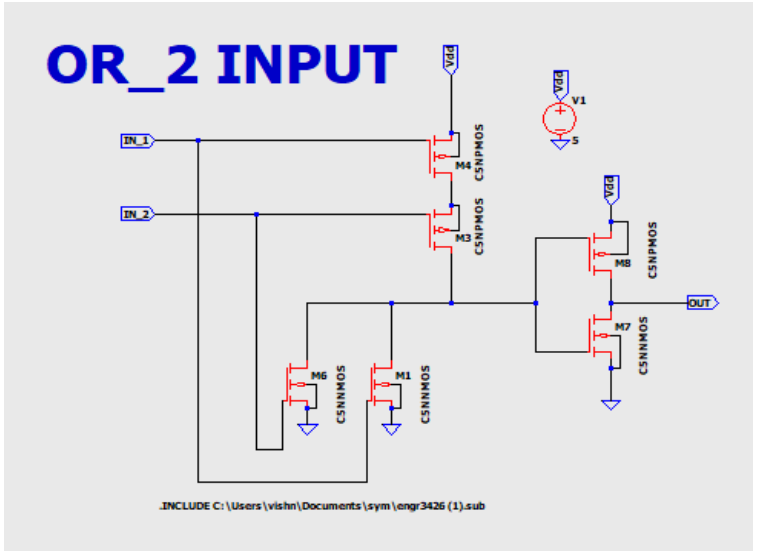


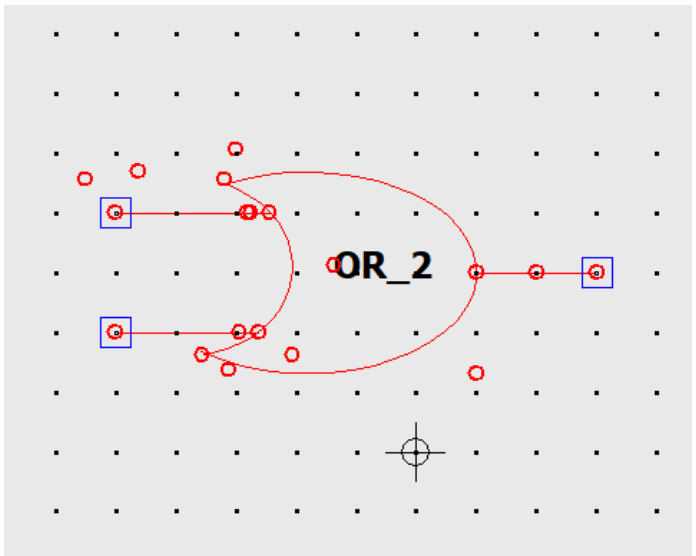

AND_4 INPUT
.INCLUDE C:\Users\vishn\Documents\sym\engr3426 (1).sub
IN_1
IN_2
IN_3
IN_4
Vdd
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
CSNPMOS
CSNNMOS
V1
5
OUT

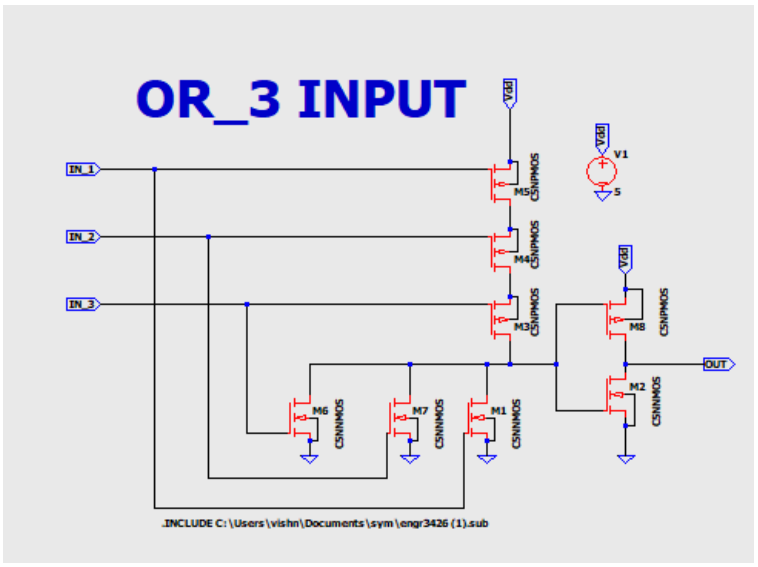

AND_4

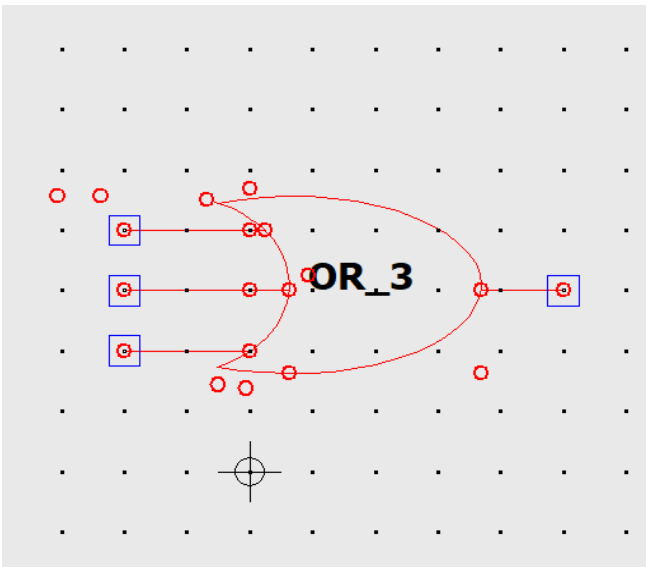

AND_5 INPUT
.INCLUDE C:\Users\vishn\Documents\sym\engr3426 (1).sub
IN_1
IN_2
IN_3
IN_4
IN_5
Vdd
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
M11
M12
CSNPMOS
CSNNMOS
V2
5
OUT

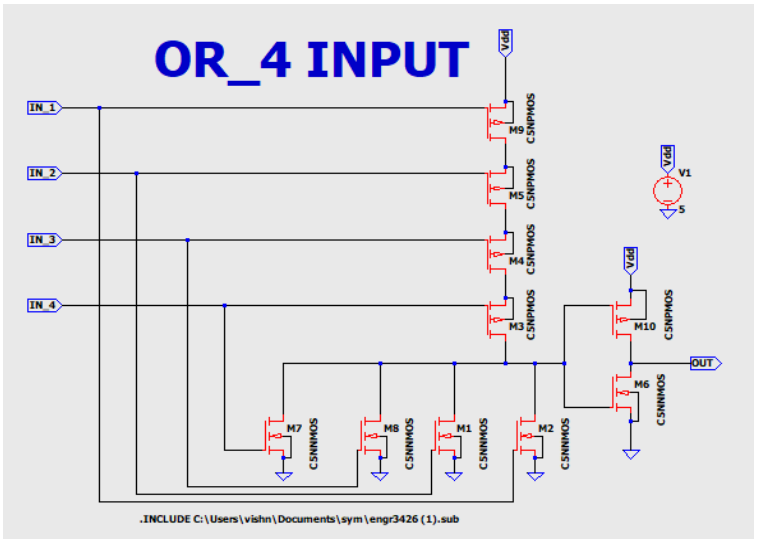

AND_5

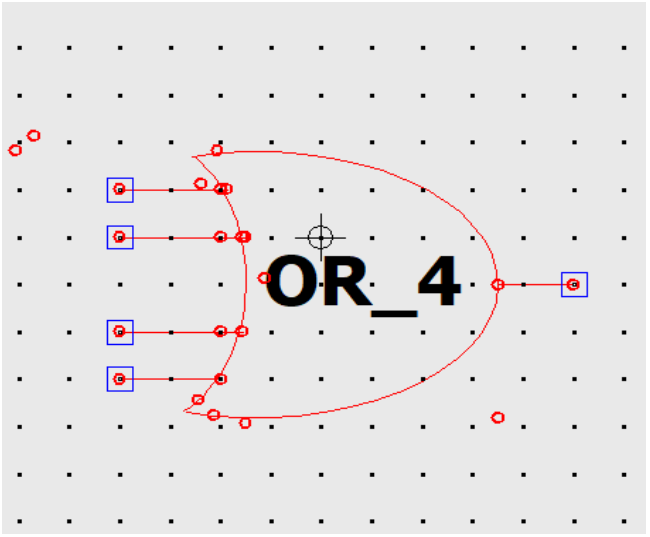

OR_2 INPUT
IN_1
IN_2
Vdd
M1
M3
M4
M6
M7
M8
CSNPMOS
CSNNMOS
V1
5
OUT
.INCLUDE C:\Users\vishn\Documents\sym\engr3426 (1).sub

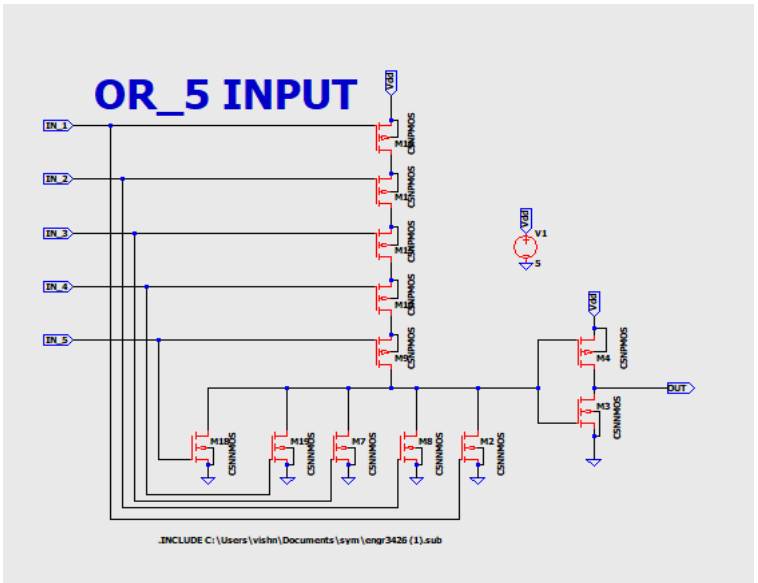

OR_2

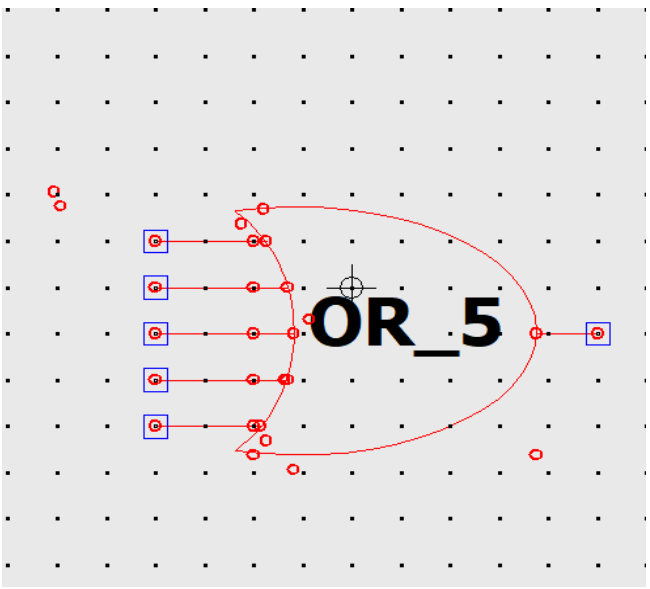

OR_3 INPUT
IN_1
IN_2
IN_3
Vdd
V1
5
M5
M4
M3
M8
M6
M7
M1
M2
CSNPMOS
CSNNMOS
OUT
.INCLUDE C:\Users\vishn\Documents\sym\engr3426 (1).sub

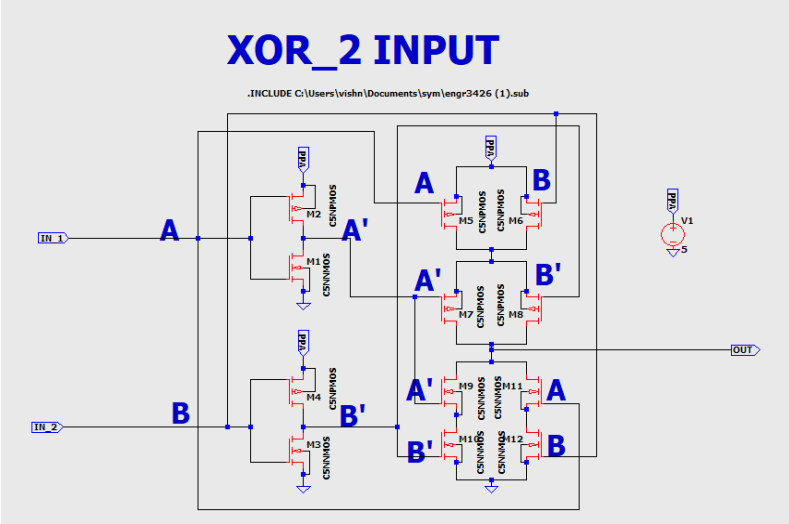

OR_3

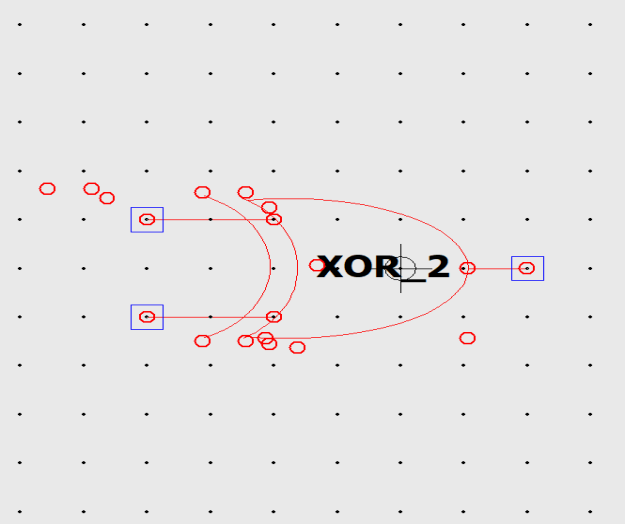

OR_4 INPUT
IN_1
IN_2
IN_3
IN_4
Vdd
V1
5
M9
M5
M4
M3
M10
M6
M7
M8
M1
M2
CSNPMOS
CSNNMOS
OUT
.INCLUDE C:\Users\vishn\Documents\sym\engr3426 (1).sub


OR_4

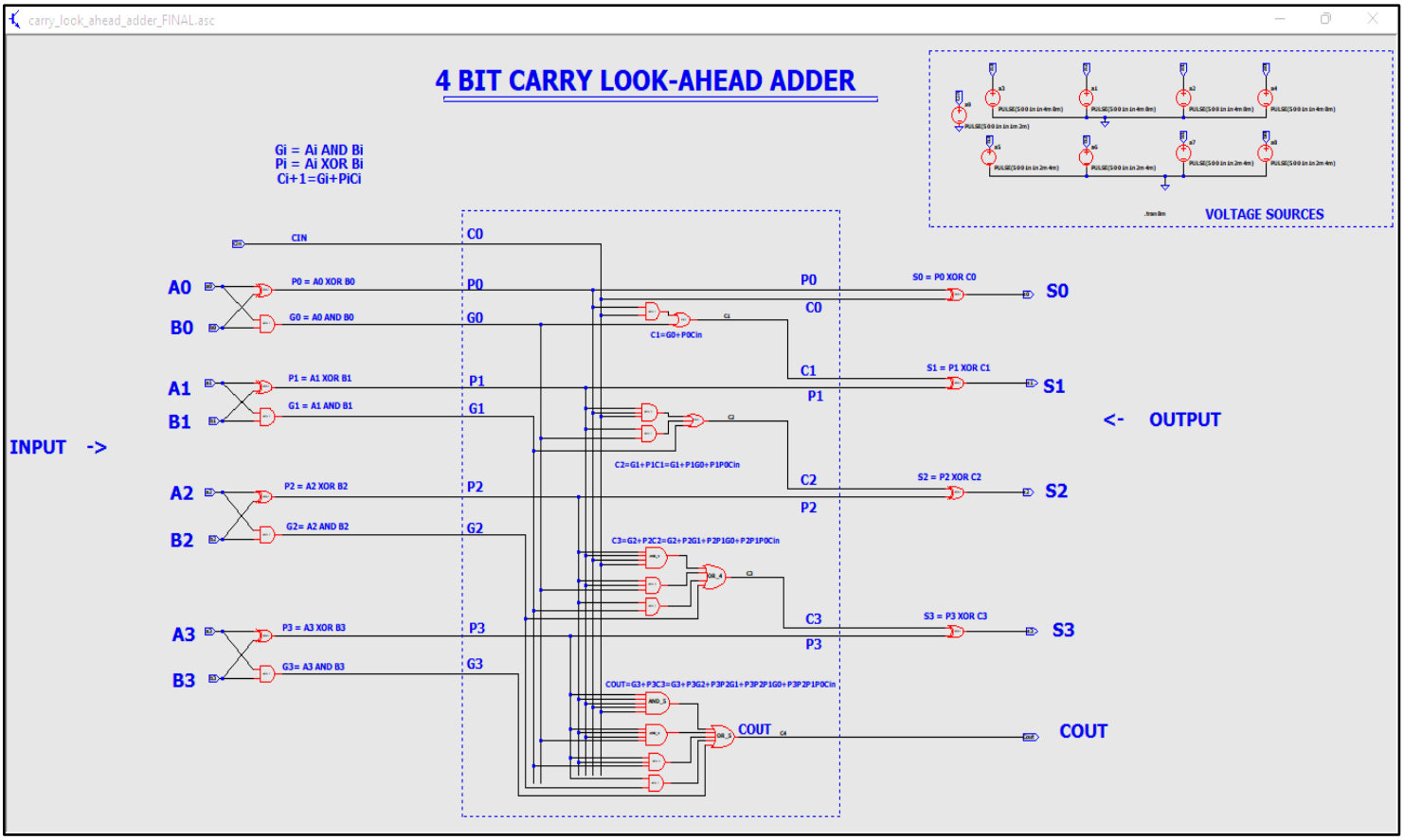

OR_5 INPUT
IN_1
IN_2
IN_3
IN_4
IN_5
Vdd
V1
5
M4
M3
M7
M8
M2
CSNPMOS
CSNNMOS
OUT
.INCLUDE C:\Users\vishn\Documents\sym\engr3426 (1).sub


OR_5

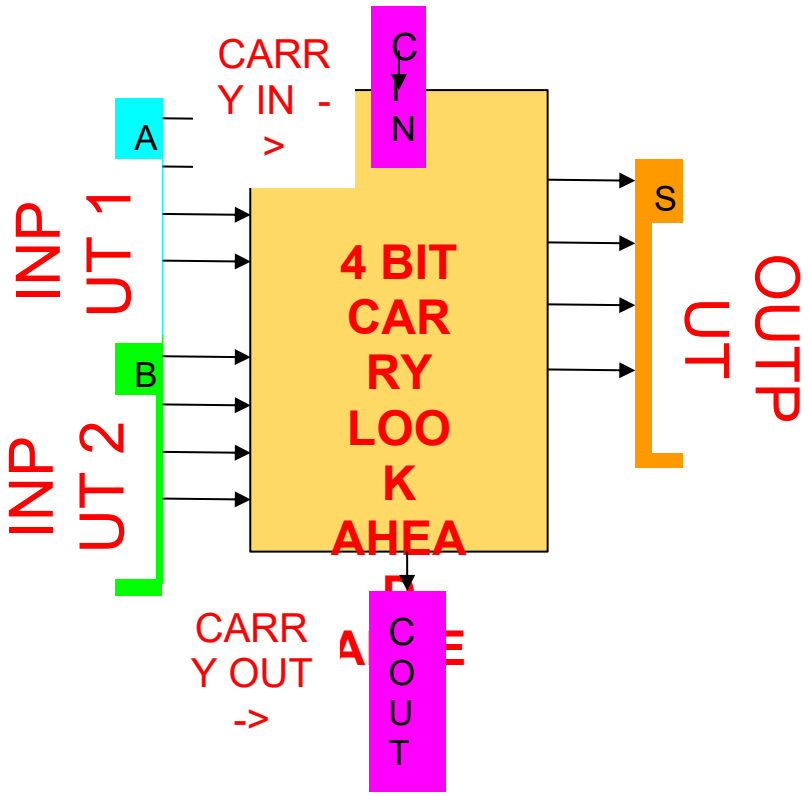




Circuit Diagram:

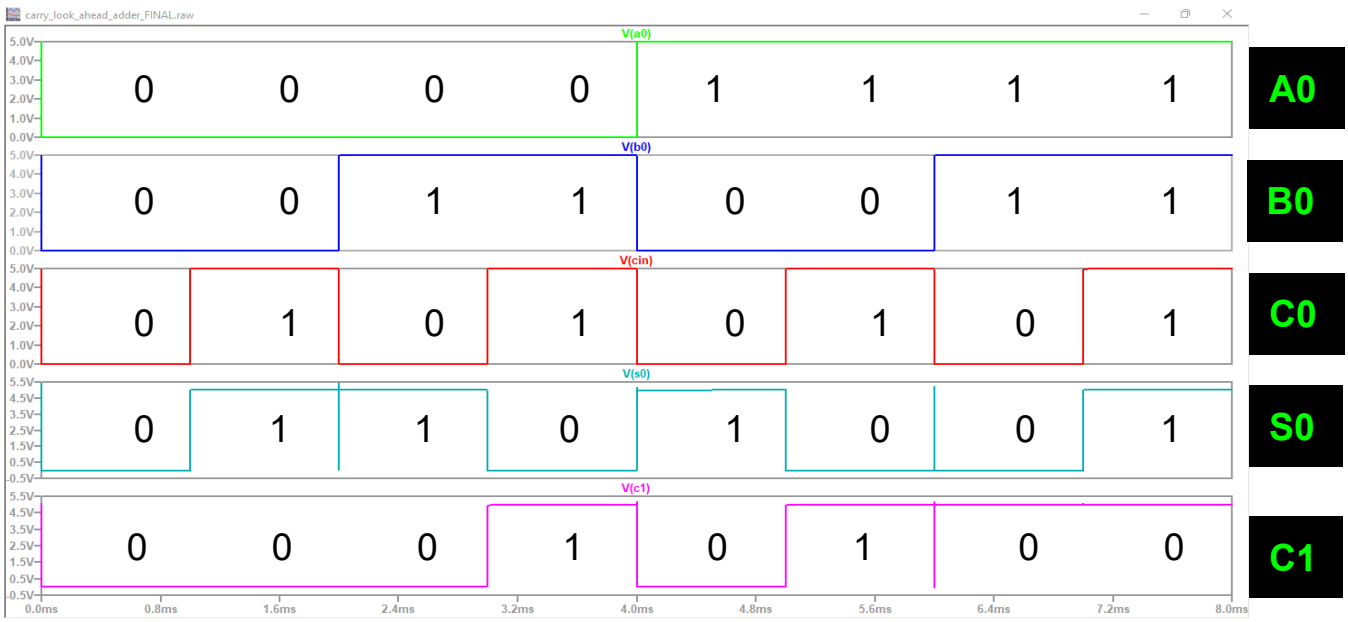


Simulation Results:

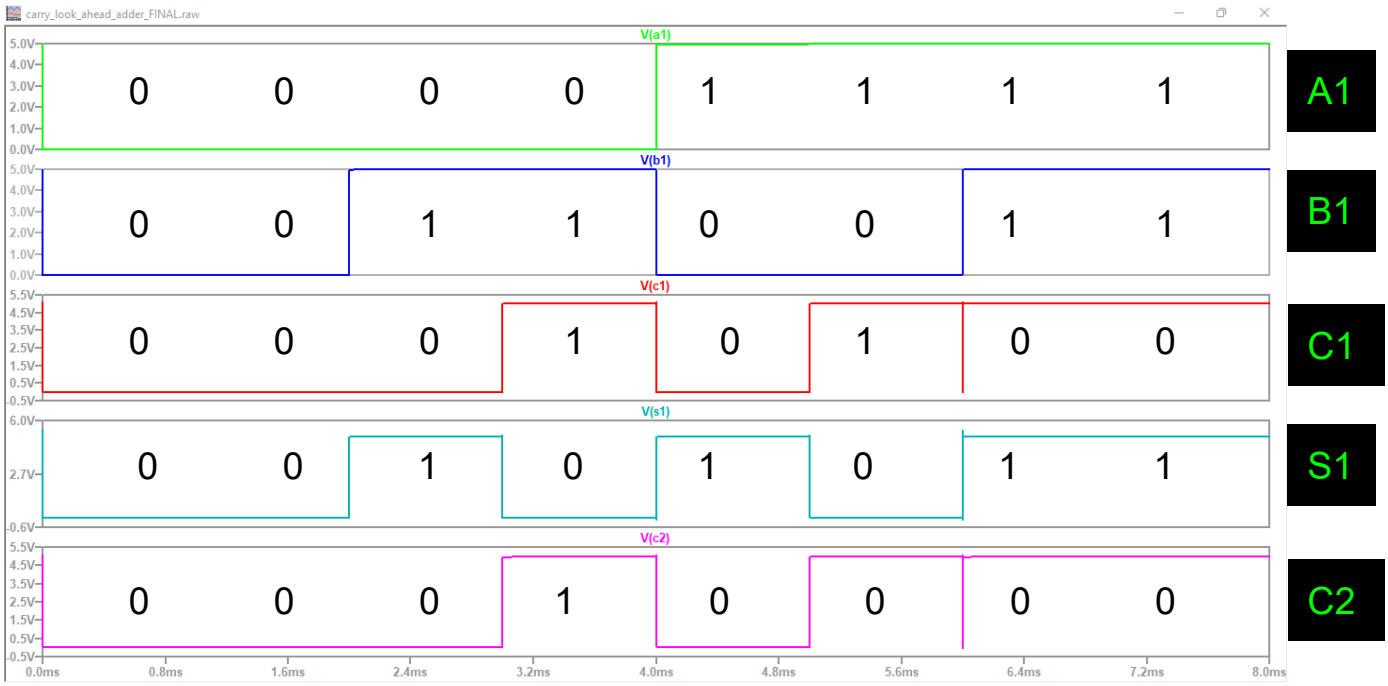


A square wave is used as inputs that transit from 5V to 0v. 5V which is treated as binary "1" and 0V which is treated as binary "0".A and B four-bit numbers. A from **A0** to **A3** and B from **B0** to **B3**. And one bit carry. In each plot, we have plotted the sum of bits from the least significant bit to the most significant bit of A and B respectively along with the sum of the carry from the previous level bit sum. To show maximum possible cases we have chosen the time period of Ai, Bi and C0 in such a way that shows maximum possible cases.

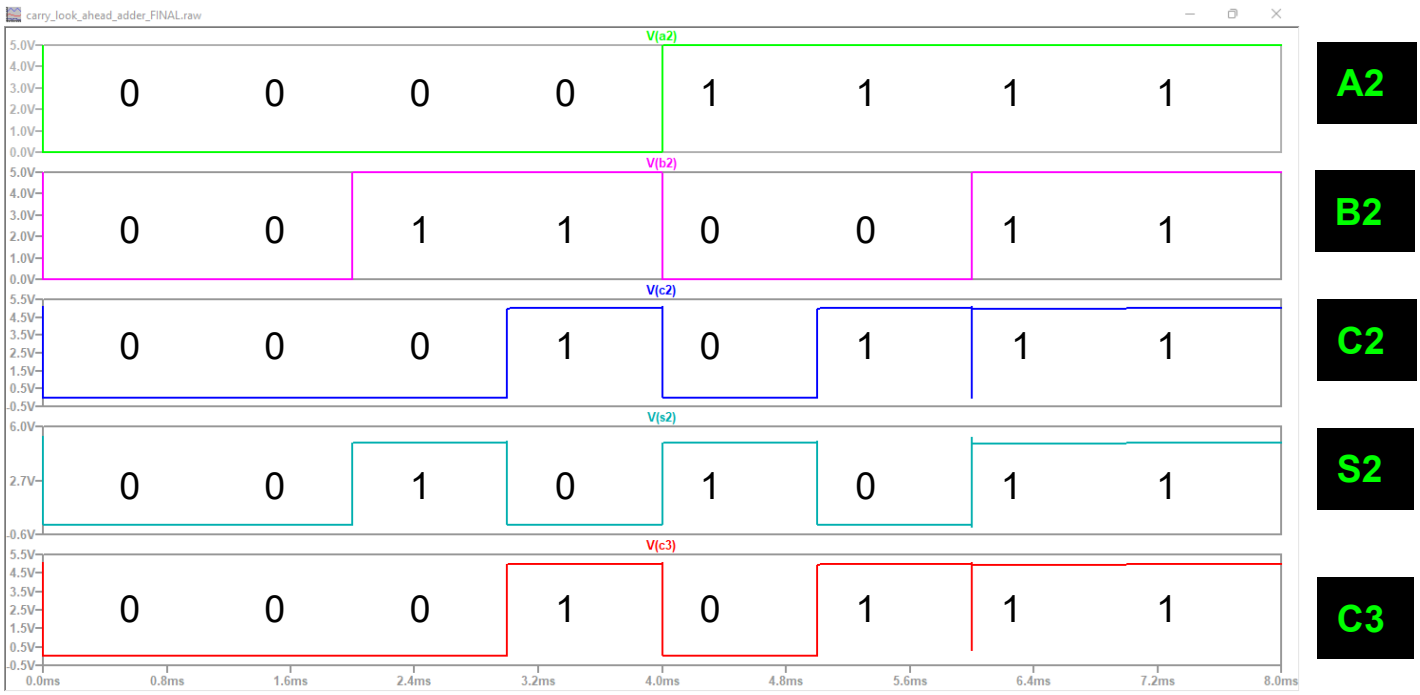
Outputs :- S0 & C1 :: inputs :- A0, B0 & C0



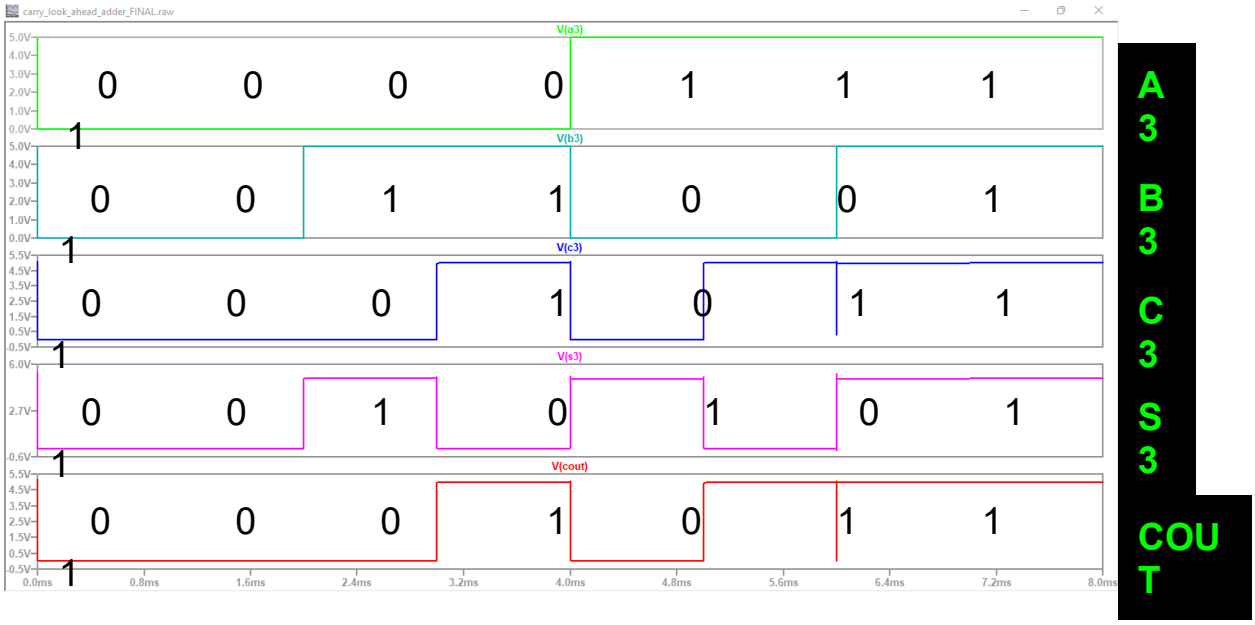
Outputs :- S1 & C2 :: inputs :- A1, B1 & C1



Outputs :- S2 & C3 :: inputs :- A2, B2 & C2



Outputs :- S3 & COUT :: inputs :- A3, B3 & C3



CONCLUSIONS

In conclusion, a schematic diagram of a 4-bit carry-lookahead adder was designed and its functional correctness was verified using simulations in Electric software integrated with LTSPICE. It was realized that the principle of the carry-lookahead adder (CLA) solves the carry delay problem by calculating the carry signals in advance,

based on the input signals. It is based on the fact that a carry signal will be generated when both bits Ai and Bi are 1, or when one of the two bits is 1 and the carry-in is 1.

Again, the correctness of the designed 4-bit adder was demonstrated by using three additions with carries from C0(Cin) to C4(final carry). In each waveform, the three-bit addition conformed exactly with that of the fundamental full adder truth table proving that the design was correct.

It was realized that the carry-lookahead adder is faster than the ripple carry adder (also known as carrying propagation adder) since all its inputs were given by G(x) and P(x) generator functions are calculated simultaneously, while for carry ripple adder, each carry needs to ripple through to produce the next carry out and hence higher delay. It was then decided that there is always the tradeoff between greater complexity and speed of circuitry in choosing between CLA and CRA.

REFERENCES:

[1] Banerjee, Sumitabha. "Carry Look-Ahead Adder." *GeeksforGeeks* https://www.geeksforgeeks.org/carry-look-ahead-adder/

[2] *Wikimedia Commons*, https://en.wikipedia.org/wiki/File:4-bit_carry_lookahead

[3]Aboah Boateng, Emmanuel. (2019). Design and Implementation of a 16 Bit Carry- Lookahead Adder. https://www.researchgate.net/publication/343982099_Design_and_Implementation_of_a_16_Bit_Carry-lLookahead_Adder