Efficient 16x16 Vedic Multiplier using various Adders

A

Dissertation

Submitted in the fulfilment of the requirements

For the award of degree

Of

MASTER OF TECHNOLOGY

In

VLSI Design and Embedded System

By

YATHAM

NAGA SAI

HARSHEESWAR

REDDY

(2K20/VLS/24)

Under the Guidance of

Dr. N.S. RAGHAVA

ELECTRONICS AND COMMUNICATION DEPARTMENT

DELHI TECHNOLOGICAL UNIVERSITY

DELHI-110042

SESSION 2020-2022

ELECTRONICS AND COMMUNICATION DEPARTMENT
DELHI TECHNOLOGICAL UNIVERSITY
DELHI-110042
SESSION 2020-2022

CANDIDATE'S DECLARATION

I hereby declare that the work being presented in this dissertation entitled ***"Efficient 16*16 Vedic Multiplier using various Adders"*** submitted towards the fulfilment of the Major project requirements for the award of degree, Master of Technology in VLSI Design and Embedded System to the Electronics and Communication Dept., Delhi Technological University, is an authentic record of my work carried out from January 2022 to June 2022, under the guidance of Dr. N.S. RAGHAVA, Electronics and Communication Dept., Delhi Technological University, Delhi.

I have not submitted the matter embodied in the dissertation for the awardof any other degree.

YNS Harsheeswar Reddy

YATHAM NAGA SAI HARSHEESWAR REDDY
2K20/VLS/24
Electronics and Communication Department

Date: 17th May, 2022

ELECTRONICS AND COMMUNICATION DEPARTMENT
DELHI TECHNOLOGICAL UNIVERSITY
DELHI-110042
SESSION 2020-2022

CERTIFICATE

This is to certify that the dissertation entitled ***"Efficient 16*16 Vedic Multiplier using various Adders"*** is the authentic record of work done by **Yatham Naga Sai Harsheeswar Reddy** under my guidance and supervision. This dissertation is being submitted to the ***Delhi Technological University, Delhi*** towards the fulfilment of the requirements for the award of ***degree of Master of Technology in VLSI Design and Embedded System.***

Date: 17th May, 2022

Dr. N.S. Raghava

SUPERVISOR

PROFESSOR

HEAD OF DEPARTMENT

ACKNOWLEDGEMENT

I would like to express my deep gratitude and appreciation to all the people who have helped and supported me in the process of dissertation. Without their help and support, 1 would not have been able to reach this level of satisfaction with what 1 have learnt and accomplished during my Master's dissertation. First and foremost, I would like to express my deep sense of respect and gratitude towards my supervisor Dr. N.S. Raghava, Professor, Head of Department, Electronics and Communication Dept., DTU, for giving me opportunity to do my Major project of master's dissertation under her guidance. I am very thankful for her for givingme the opportunity to choose such an interesting topic by my own. I would also like to thanks the NPTEL Lectures for their valuable thoughts and knowledge, which motivated me to do better. Finally, none of this would have been possiblewithout incredible support of my friends. They were always supporting me and encouraging me with their best wishes.

[illegible]

YATHAM NAGA SAI HARSHEESWAR REDDY

Roll No. 2K20/VLS/24

Electronics and Communication Dept.

ABSTRACT

We live in an era of artificial intelligence which leads to lots of automation for the betterment of life and society. Semiconductor industries, makes the way for bigger solution for people and which brings the technology to audience ranging from common people to stalwarts in the means of smallest components as sensors, mobiles, laptop to bigger components as data centers. As technology is evolving and increasing demand of the technologies and the evolution of the products and the giants of semiconductor industries like Intel, Texas, Samsung, Qualcomm and Western Digital are finding their way in a best possible way to design the product which is user friendly.

There are various constraints which are implemented, which are imposed by these industries like functionality of the electronic device, power dissipation by the product, area occupied and also the reliability of the product. All these constraints, require some special attention and the measurements, which needs to be fulfilled by the design engineers, so that the reputation of the industry, andthe competition in the products will be sustained.

The electronic devices process the digital signals and process here is nothing but the Addition, Subtraction and Multiplication which are some Arithmetic operations. To make devices faster either we can go for better technology in terms of MOS that is having lesser feature size or make the Arithmetic operations more efficient. Here in this thesis we focus on the latter part. In this thesis we look into both Multipliers and Adders, in multipliers we focus on three techniques. While in Adders we look into both logical and circuit design styles of adder and know the best possible combination of the adder we can use.

TABLE OF CONTENTS

Declaration	*i*
Certificate	*ii*
Acknowledgement	*Iii*
Abstract	*iv*
List of Figures	*6*
List of Tables	*9*
List of Abbreviations	***10***
CHAPTER 1	**11-15**
Introduction	
1.1 Vedic Mathematics	11
1.2 Vedic Multiplier	13
CHAPTER 2	**17-20**
Types of Adders	16
2.1 Ripple Carry Adder	18
2.2 Carry Look Ahead	19
2.3 Carry Select Adder	20
CHAPTER 3	**21-23**
CIRCUIT DESIGN STYLES OF ADDER	
3.1 Static CMOS Full Adder	21
3.2 DCVSL based Full Adder	22
3.3 NORA based Full Adder	23
CHAPTER 4	**24-69**
RESULTS AND DISCUSSION	
CHAPTER 5	**70**
CONCLUSION AND FUTURE SCOPE	
REFERENCES	

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
1.1	Decimal Multiplication Using Urdhva Triyagbhyam	13
1.2	Decimal Multiplication of Nikhilam Navatashcaramam Dashatah	14
2.1	32-Bit Ripple Carry Adder	18
2.2	4-Bit Carry Look Ahead Adder	19
2.3	4-Bit Carry Select Adder	20
3.1	1-Bit Static CMOS Full Adder	21
3.2	1-Bit Differential Cascode Switch logic based Full Adder	22
3.3	1-Bit Dynamic CMOS logic Full Adder	23
4.1	Simulation of 2*2 Vedic Multiplier	24
4.2	Netlist of 2*2 Vedic Multiplier	24
4.3	Simulation of 4*4 Vedic Multiplier	25
4.4	Netlist of 4*4 Vedic Multiplier	25
4.5	Simulation of 8*8 Vedic Multiplier	26
4.6	Netlist of 8*8 Vedic Multiplier	26
4.7	Simulation of 16*16 Vedic Multiplier	27
4.8	Netlist of 16*16 Vedic Multiplier	27
4.9	Simulation of 4-Bit Carry Select Adder	28

4.10	Netlist of 4-Bit Carry Select Adder	28
4.11	Simulation of 8-Bit Carry Select Adder	29
4.12	Netlist of 8-Bit Carry Select Adder	29
4.13	Simulation of 12-Bit Carry Select Adder	30
4.14	Netlist of 12-Bit Carry Select Adder	30
4.15	Simulation of 16-Bit Carry Select Adder	31
4.16	Netlist of 16-Bit Carry Select Adder	31
4.17	Simulation of 24-Bit Carry Select Adder	32
4.18	Netlist of 24-Bit Carry Select Adder	32
4.19	Schematic of 1-Bit full Adder using Static CMOS	34
4.20	Inputs and Output of 1-Bit Full Adder using Static CMOS logic	35
4.21	Schematic of 4-Bit Ripple Carry Adder using Static CMOS	36
4.22	Inputs and Output of 4-Bit Ripple Carry Adder using Static CMOS logic	37
4.23	Schematic of 8-Bit Ripple Carry Adder using Static CMOS logic	38
4.24	Inputs and Output of 8-Bit Ripple Carry Adder using Static CMOS logic	39
4.25	Schematic of 16-Bit Ripple Carry Adder using Static CMOS logic	40
4.26	Inputs and Output of 16-Bit Ripple Carry Adder using Static CMOS logic	41
4.27	Inputs and Output of 16-Bit Ripple Carry Adder using Static CMOS logic	42
4.28	Schematic of 1-Bit full Adder using Differential Cascode Logic	43
4.29	Inputs and Output of 1-Bit Full Adder using Differential Cascode logic	44
4.30	Schematic of 4-Bit full Adder using Differential Cascode Logic	45
4.31	Inputs and Output of 4-Bit Full Adder using Differential Cascode logic	46

4.32	Schematic of 8-Bit full Adder using Differential Cascode Logic	47
4.33	Inputs and Output of 8-Bit Full Adder using Differential Cascode logic	48
4.34	Schematic of 16-Bit full Adder using Differential Cascode Logic	49
4.35	Inputs and Output of 16-Bit Full Adder using Differential Cascode logic	50
4.36	Inputs and Output of 16-Bit Full Adder using Differential Cascode logic	51
4.37	Schematic of 1-Bit full Adder using NORA (NO Race Around) Logic	52
4.38	Inputs and Output of 1-Bit Full Adder using NORA logic	53
4.39	Schematic of 4-Bit full Adder using NORA (NO Race Around) Logic	54
4.40	Inputs and Output of 4-Bit Full Adder using NORA logic	55
4.41	Schematic of 8-Bit full Adder using NORA (NO Race Around) Logic	56
4.42	Inputs and Output of 8-Bit Full Adder using NORA logic	57
4.43	Schematic of 16-Bit full Adder using NORA (NO Race Around) Logic	58
4.44	Inputs and Output of 16-Bit Full Adder using NORA logic	59
4.45	Inputs and Output of 16-Bit Full Adder using NORA logic	60
4.46	Verilog Code of Half Adder	61
4.47	Verilog Code of 2-Bit Multiplier	61
4.48	Verilog Code of 4-Bit Multiplier	62
4.49	Verilog Code of 8-Bit Multiplier	63
4.50	Verilog Code of 16-Bit Multiplier	64
4.51	Verilog Code of Full Adder	65
4.52	Verilog Code of 2*1 Multiplexer	66
4.53	Verilog Code of 4-Bit Carry Select Adder	67

4.54	Verilog Code of 6-Bit Carry Select Adder	68
4.55	Verilog Code of 8-Bit Carry Select Adder	69
4.56	Verilog Code of 12-Bit Carry Select Adder	70
4.57	Verilog Code of 16-Bit Carry Select Adder	71

LIST OF TABLES

TABLE	TABLE NAME	PAGE NO
Table 5.1	Delays of 16-bit RCA	52

LIST OF ABBREVIATIONS

RTL	Register transfer level
VLSI	Vert Large Scale Integration
GDSII	Graphic design system
ASIC	Application specific integrated circuit
UT	Urdhva Triyagbhyam
SOC	System on chip
FPGA	Field programmable gate array
HDL	Hardware description language
GPDK	Generic process Design Kit
DCVSL	Differential Cascode Voltage Switch Logic
NORA	No Race Around

Chapter 1

INTRODUCTION

1.1 VEDIC MATHEMATICS

Multipliers play an important role in building processors Multipliers play an important role in most of the digital devices and prominently in digital communications and many other applications as well. Multipliers are used to perform multiplication and various kinds of multipliers are present in today's world and used according to requirement. Multiplying the numbers is very expensive and it takes more memory, time and space. Depending upon multipliers it takes different memory, different delay.

Vedic Mathematics is an ancient method which deals in unique way in performing mathematical operations. There are 16 Sutras where Vedic mathematics revolve around. The application of vedic mathematics is used in number of ways and core reason of it is solve the numerical problems faster than the modern calculations. Few benefits of using Vedic sutras are listed below.

1. Calculations become short and easy.
2. Simplifications takes very less time.
3. Complex calculations can be done in efficient manner.
4. Almost every arithmetic operation can be done using just 16 sutras.

All vedic 16 sutras are listed below

1. Chalana-Kalanabyham – The Differences and similarities.
2. Shunyamanyat – If one is in ratio and the other is zero.
3. Ekanyunena Purvena – By one less than the previous one.
4. Ekadhikena Purvena – By one more than the previous one.
5. Gunakasamuchyah – The factors of the sum is equals to the sum of factors.
6. Gunitasamuchyah – The product of the sum is equal to the sum of product.
7. Paraavartya Yojayet -Transpose and Adjust.
8. Puranapuranabyham – By the completion or noncompletion.
9. Sankalana – vyavakalanabhyam – By addition and by subtraction.
10. Shesanyankena Charamena – The remainders by the last digit.
11. Shunyam Saamyasamuccaye –When the sum is the same that the sum is zero
12. Sopaantyadvayamantyam – The ultimate and twice the penultimate.
13. Nikhilam Navatashcaramam Dashatah – All from 9 and last from 10.
14. Urdhva-Triyagbhyam – Vertically and Crosswise.
15. Vyashtisamanstih – Part and Whole.
16. Yaavadunam – Whatever the extent of its deficiency.

These 16 sutras are used for basic to advance arithmetic operations fast and most efficient way possible and can be verified using formal methods. In these sutras only few qould be picked up for vedic Multiplier.

1.2 VEDIC MULTIPLIER

Multiplication in Vedic Mathematics are divided in the sutras as shown below.

1. Urdhva Triyagbhyam (UT)
2. Nikhilam Sutra
3. Ekadhikena Purvena

1. Urdhva Triyagbhyam

This is one of the easy method of multiplication in vedic mathematics. This technique provide easy method to multiply two numbers. It’s method is described with an example.

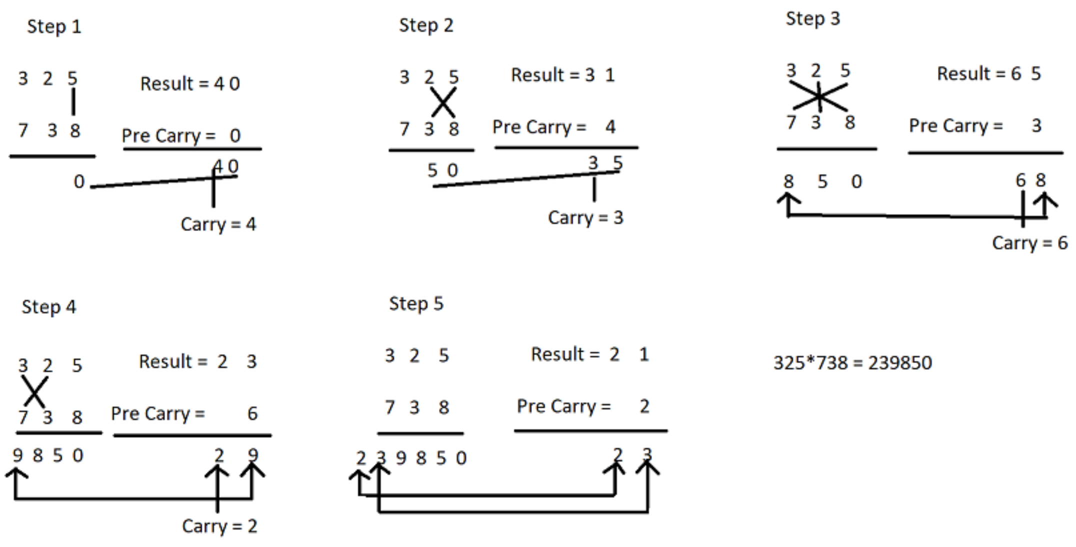


Fig.1.1 Decimal Multiplication Using Urdhva Triyagbhyam

2. Nikhilam Navatashcaramam Dashatah

Nikhilam Navatashcaramam Dashatah is one of the 16 sutras in vedic mathematics. The speciality of this method is to break down the large number multiplication into small digit multiplication by efficiently using Addition and Subtraction and Shift operation. This is one of the fastest multiplication method used in vedic mathematics. Three cases would be discussed here:

1. Numbers nearest and greater than the powers of 10. Example: 101*102, 1003*1001, 10005*10006.
2. Numbers nearest and lesser than the powers of 10. Example: 96*98, 998*999, 9992*9999
3. Numbers nearest and be either sides of power of 10. Example: 96*101, 998*1004, 10002*9997.

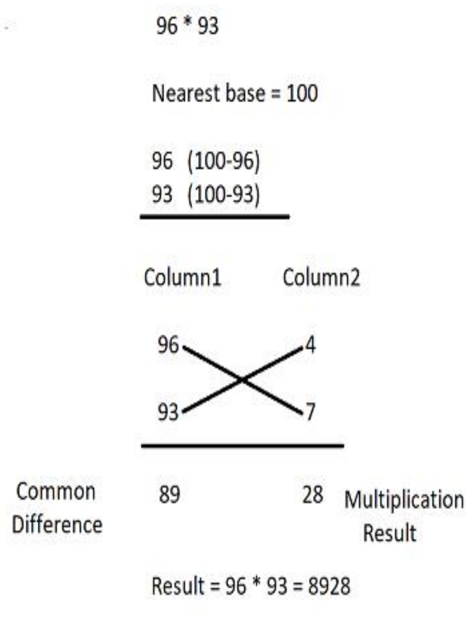


Fig.1.2 Decimal Multiplication of Nikhilam Navatashcaramam Dashatah

3. Ekadhikena Purvena

Ekadhikena Purvena is one of the 16 sutras in vedic mathematics. The name Ekadhikena Purveena in Sanskrit mean, add one to the previous one to get next number. This technique is generally used to get fast multiplication when digits of two number are in this fashion i.e AD and AE. Here certain conditions has to be followed to implement Ekadhikena Purvena

- First digits should be of same number (Here A=A).
- Adding both second digits should be 10. (Here D+E=10).

1. With an example of multiply two numbers 57 and 53.
2. Above two conditions are followed, first digit is 5 and adding both second digit results in 10.
3. By applying Ekadhikena Purvena sutra to above example, i.e. increment it by one which becomes 6.
4. Having the product of 6*5 results in 30. This 30 is the first two digits of the final result.
5. Having the product of other two digits is 7*3 is 21. So final result will be 3021.

THESIS ORGANIZATION

The Thesis is organized in 3 chapters which are as follows:

- **Chapter 1** introduces the aspect of Vedic Mathematics which gives the coverage of all 16 sutras and insights of Vedic Multiplier and particular sutras which supports for multiplication. And the objective of the thesis which is showing the motivation while taking this title.

- **Chapter 2** helps in understanding the Adders in detail, which includes the various types of Adders and different logic styles of Adders.

- **Chapter 3** includes includes Circuit Design Styles of Adder in detail, which adder would be good in terms of delay.

- **Chapter 4** includes Results and the discussion of the codes and explanation is presented in the chapter.

- **Chapter 5** includes Conclusion and Future Scope what can be done and further improvements of the design.

CHAPTER 2

ADDERS

Adders play an important role in Arithmetic operations. In almost every digital component there would be operations as Addition and Subtraction and even Subtraction operations are done by using Adders. So optimization of Adders is useful as it widely used. The choice of adders are done by following particular requisite:

- Lesser Area
- Lower Power Consumption
- Lower Power Dissipation
- High Speed

Depending upon the above nececssities adders would be chosen. As a particular adder cannot achieve all the above characterisics, the requirement demands which adder need to be used. Few Adders which gonna be discussed in this paper are:

1. Ripple Carry Adder
2. Carry Look Ahead Adder
3. Carry Select Adder

In Multiplying two numbers require addition after generating partial products. There are various kinds of adders such as Ripple Carry Adder, Carry Look–Ahead Adder, Carry Save Adder, Carry Select Adder. Each and every adder has its own advantages and disadvantages as in Ripple carry adder gives the most compact design at the same time computation time is high.

Types of Adders

2.1 Ripple Carry Adder

Ripple Carry adders is a basic adder. Here the word itself says the 'Ripple' mean the output of previous is transferred to the current cell. Ripple Carry Adder contains the series of Adders output of one Adder is connected to input of next adder serially. Here Carry of previous adder is connected to another input to the next respective adder cells. Here the time taken to complete the addition gets high until and unless carry gets generated from previous cell it the current adder cell cannot proceed which is one of the major disadvantage of this adder. The Architecture of 32- bit Full Adder can be seen below. Here in Ripple Carry Adder the Area consumption is less the propagation delay here is very high .

Sum = A XOR B XOR C

Carry = AB + BCin + ACin

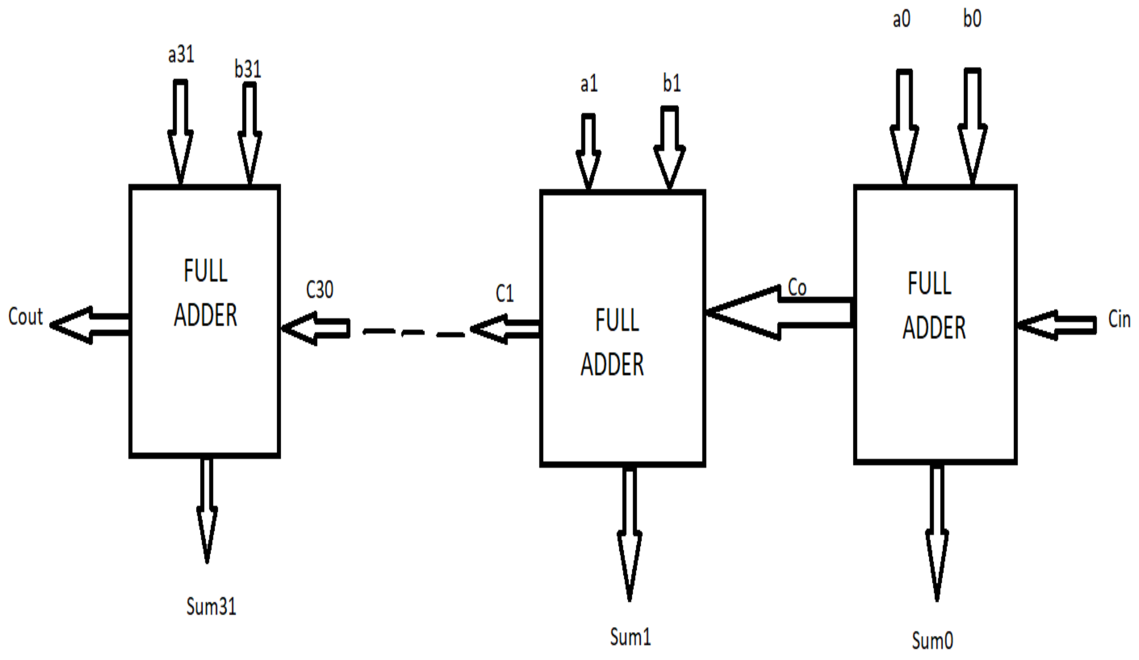


Fig.2.1.1 32-bit Ripple Carry Adder

\

2.2 Carry Look Ahead Adder

Carry Look Ahead Adder is better than RCA in obtaining the result. The carry gets calculated in the intermediate stages using Carry generate and Carry propagate whatever would be the input carry. That is how the name Carry Look Ahead name is given. Here two terms are predominantly important are i.e Carry Generate and Carry Propagate. Carry Propagate will be propagated to further stages of Adder. Carry generate work is to prior generation of carry whatever the input carry would be.

Equations of Carry Look Ahead Adder are:

$$P(i) = A(i) \text{ xor } B(i)$$

$$G(i) = A(i) \text{ and } B(i)$$

$$S(i) = P(i) \text{ xor } C(i)$$

Here i can be the values of i= 1,2,3,4…. Depending upon the number of bit of Carry Look Ahead Adder. For a 4-bit Carry Look Ahead Adder i would be the value of 3, i.e for N-bit Carry Look Ahead Adder the value of i would be taken as N-1.

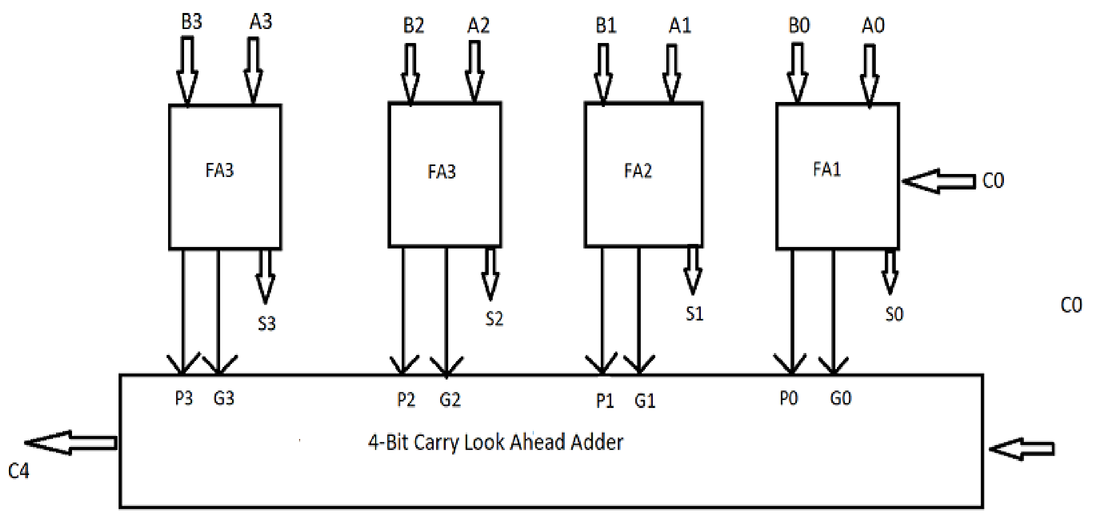


Fig.2.1.2 4-Bit Carry Look Ahead Adder

2.3 Carry Select Adder

Carry Select Adder is the application of Ripple Carry Adder as it comprises of two RCA and a multiplexer. Carry Select adder gives the output for possible values of carry input which mean output for Cin = 0, Cin = 1 both values. Carry Propagate will be propagated to further stages of Adder. Carry generate work is to prior generation of carry whatever the input carry would be.

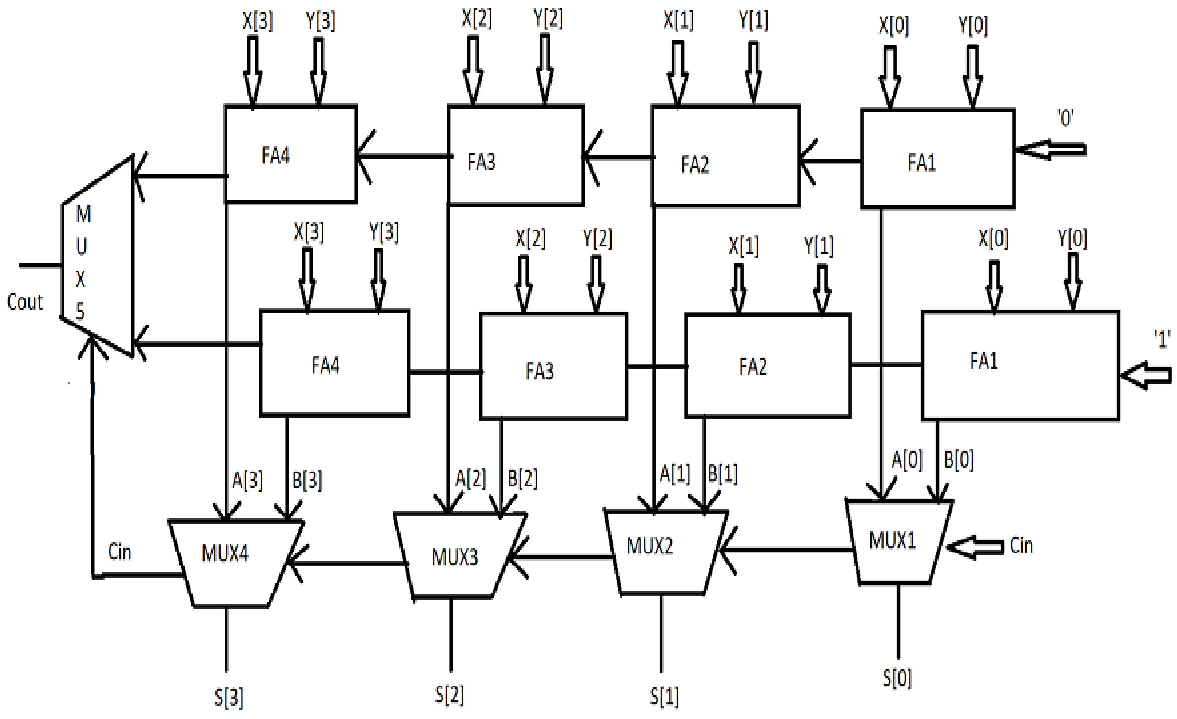


Fig.2.1.3 4-bit Carry Select Adder

CHAPTER 3

CIRCUIT DESIGN STYLES OF ADDER

The Full Adders is the basis for almost every Arithmetic units. There will be a numerous CMOS circuit design styles which includes both static and dynamic in nature. Here we discuss about few circuit design styles of Adders which are given below:

The physical design steps comprise of:

1) Static CMOS

2) Differential Cascode Voltage Switch Logic

3) NoRa Logic

3.1 Static CMOS Full Adder

A Full Adder designed using Static CMOS logic which employs both P-type and N-type logic. The Upper half of circuit allows the output to be charged high upto VDD and lower half of circuit allows the output to be discharged to ground.

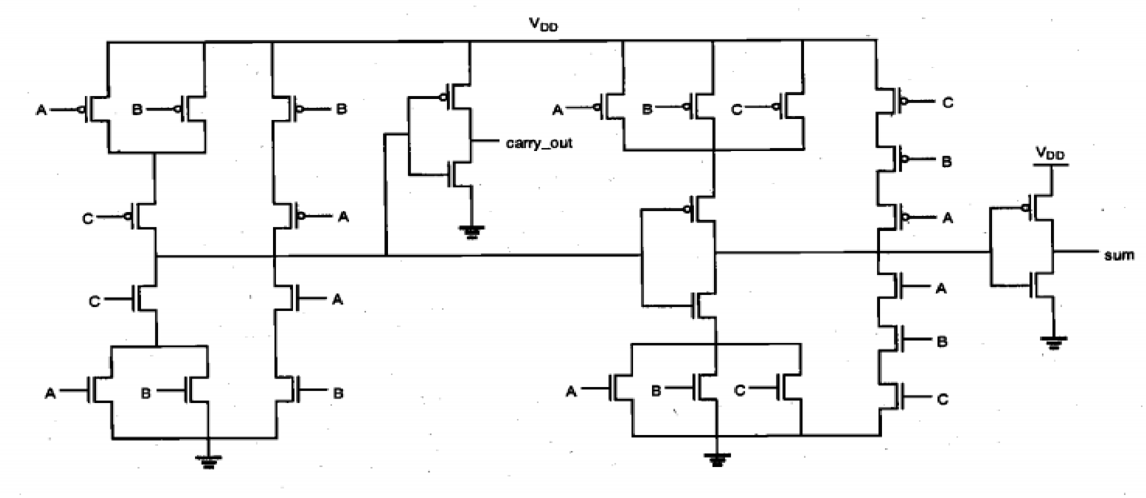


Fig.3.1 1-Bit Static CMOS Full Adder

3.2 Diffential Cascode Switch Logic Full Adder

A cross coupled P-type transistor in Cascode Voltage Switch Logic with a cross coupled pair of P-transistors results in static version of that logic is known as Differential Cascode Switch logic (DCVSL). The cross coupled P-transistors acts as differential pair. Here when an output of one side gets low the opposite P-transistor gets on and high.

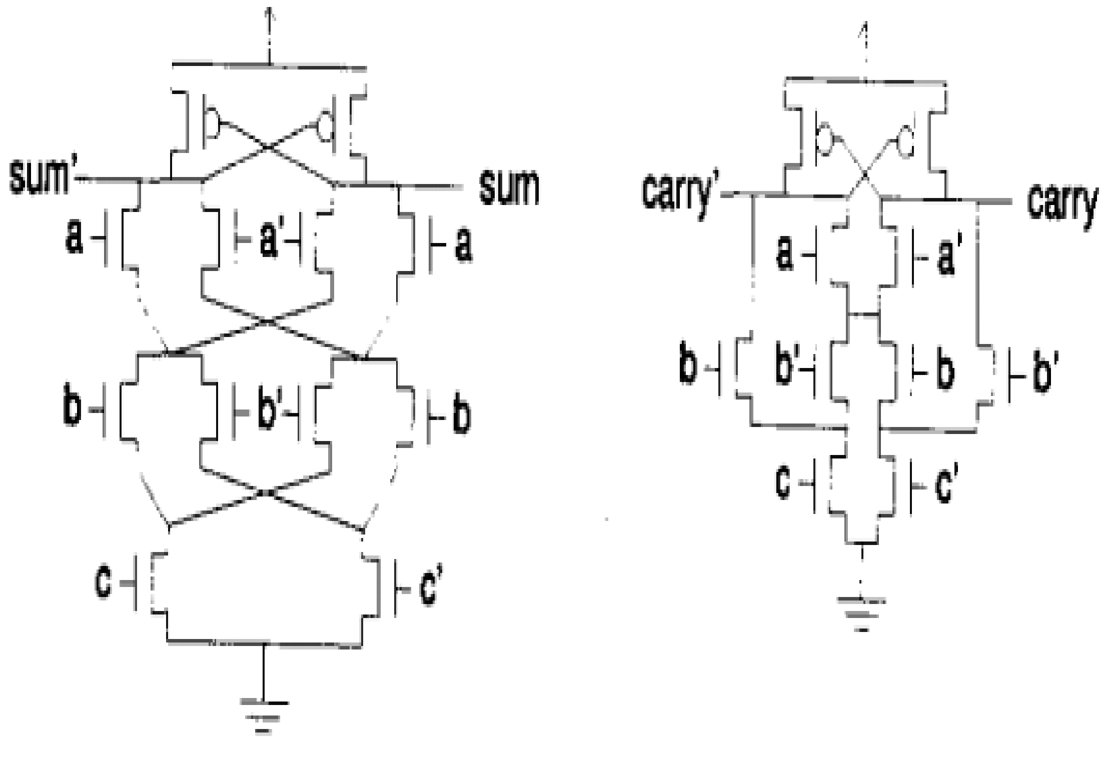


Fig.3.2 1-Bit Differential Cascode Switch Logic (DCVSL) based Full Adder

3.3 NO – RACE(NORA) Dynamic CMOS Full Adder

A Full Adder made using No Race Dynamic Logic (NORA) has an alternating stages of P and N-type logic trees to get Sum and Carry outputs. The P-type stage that forms the carry output is dynamically charged high and while the N-type transistor that evaluates the Sum output is dynamically pre-charged to low. Here is the logic we require two phase clocking named as phi and phi'.

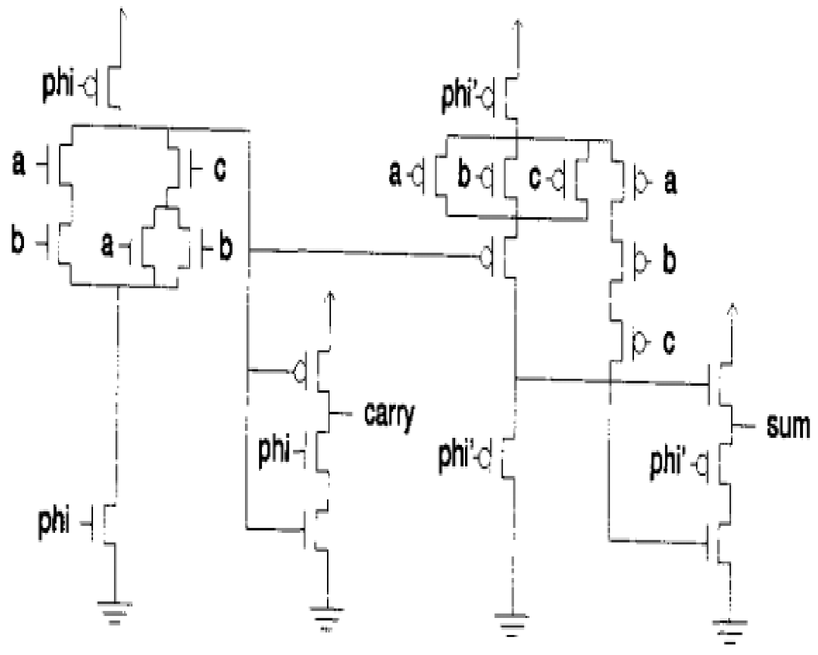


Fig.3.3 1-Bit No-Race Dynamic CMOS Logic Full Adder

CHAPTER 4

RESULTS AND DISCUSSIONS

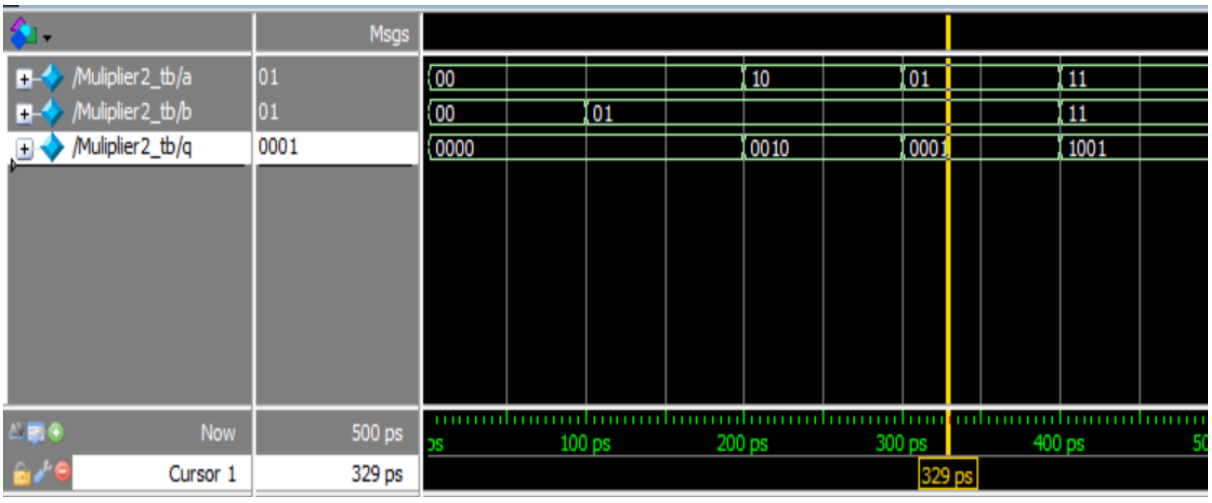


Fig.4.1 Simulation of 2*2 Vedic Multiplier

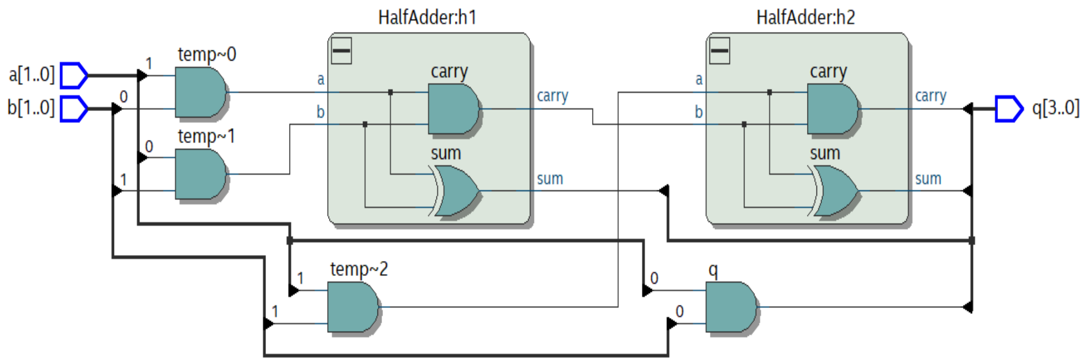


Fig.4.2 Netlist of 2*2 Vedic Multiplier

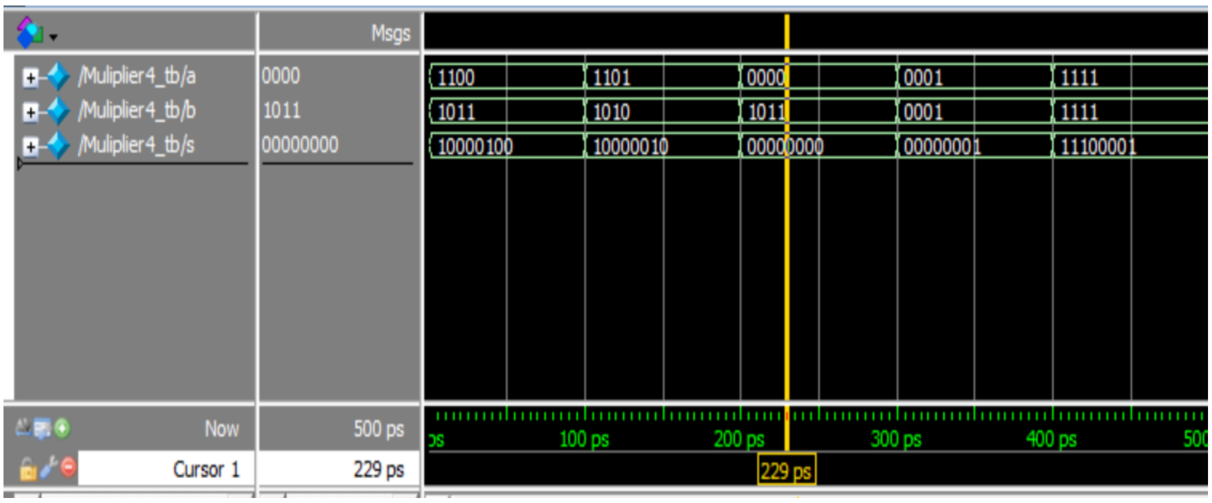


Fig.4.3 Simulation 4*4 Vedic Multiplier

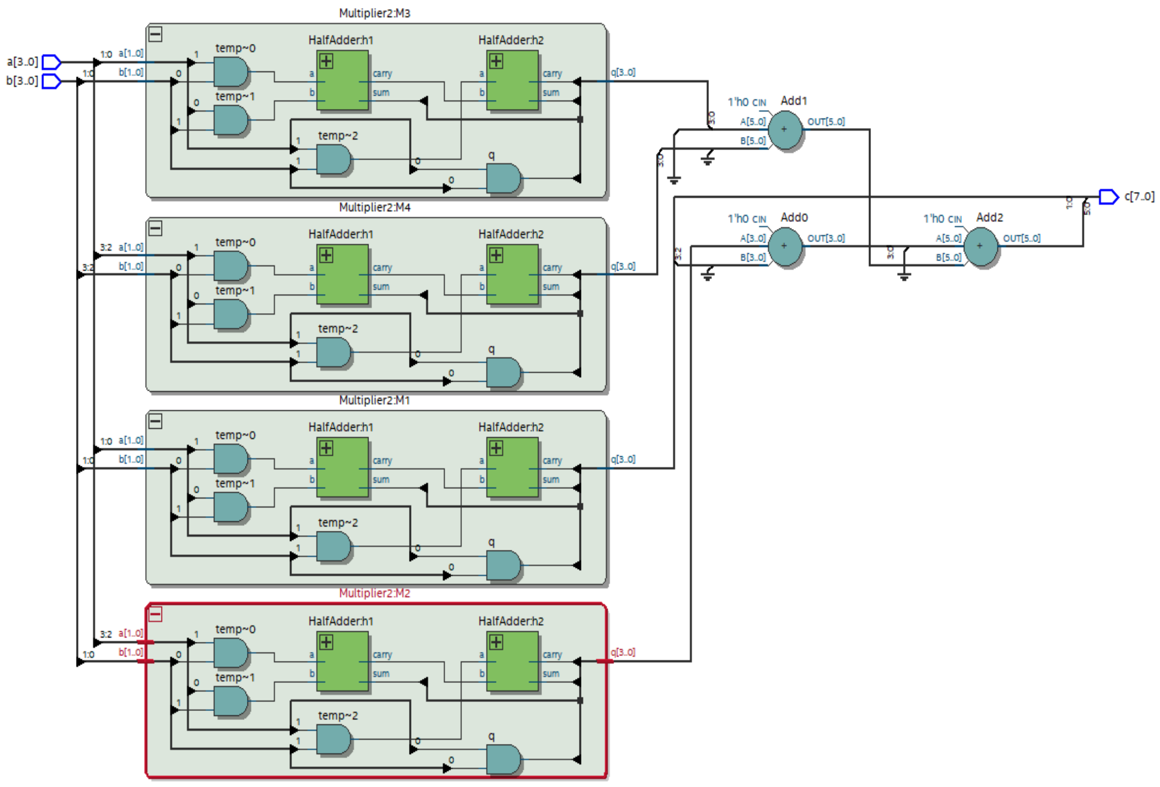
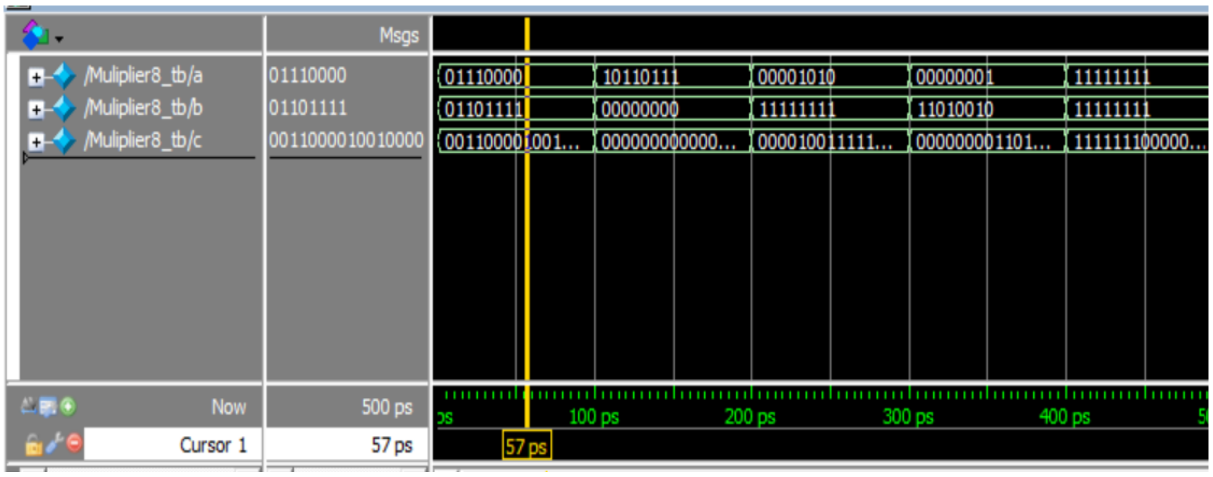


Fig.4.4 Netlist of 4*4 Vedic Multiplier



```
                    Msgs
/Muliplier8_tb/a    01110000            01110000        10110111        00001010        00000001        11111111
/Muliplier8_tb/b    01101111            01101111        00000000        11111111        11010010        11111111
/Muliplier8_tb/c    0011000010010000    00110000 1001... 000000000000... 000010011111... 00000001101... 111111100000...

Now                 500 ps              ps   100 ps   200 ps   300 ps   400 ps   5
Cursor 1            57 ps               57 ps
```

Fig.4.5   Simulation of 8*8 Vedic Multiplier

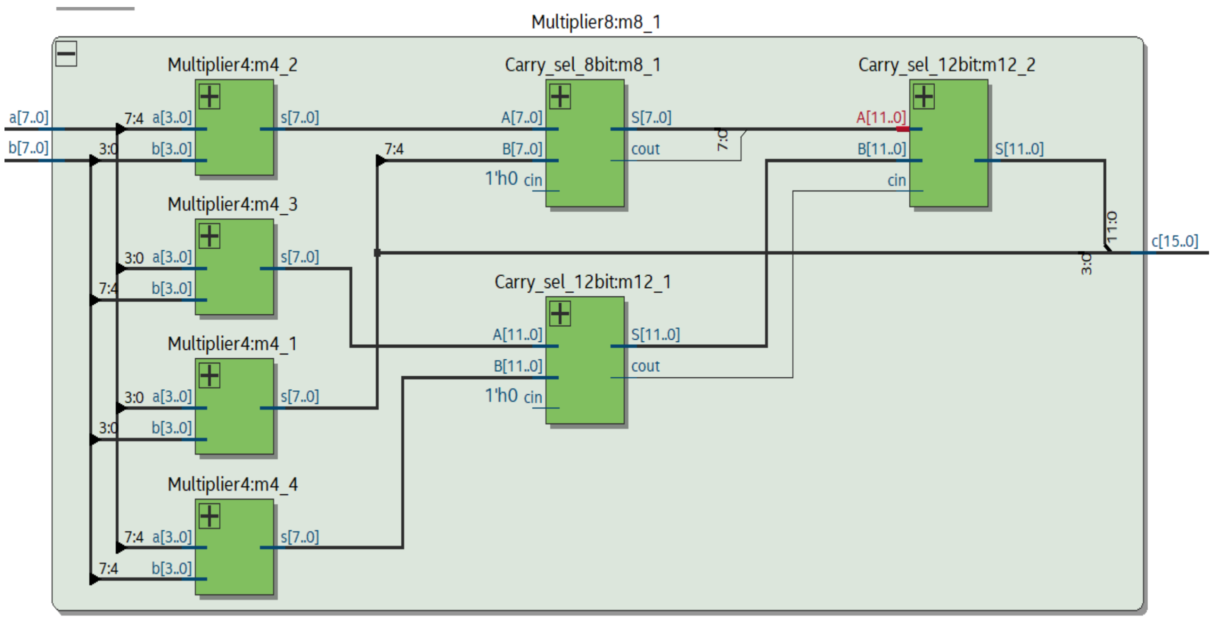


Fig.4.6   Netlist of 8*8 Vedic Multiplier

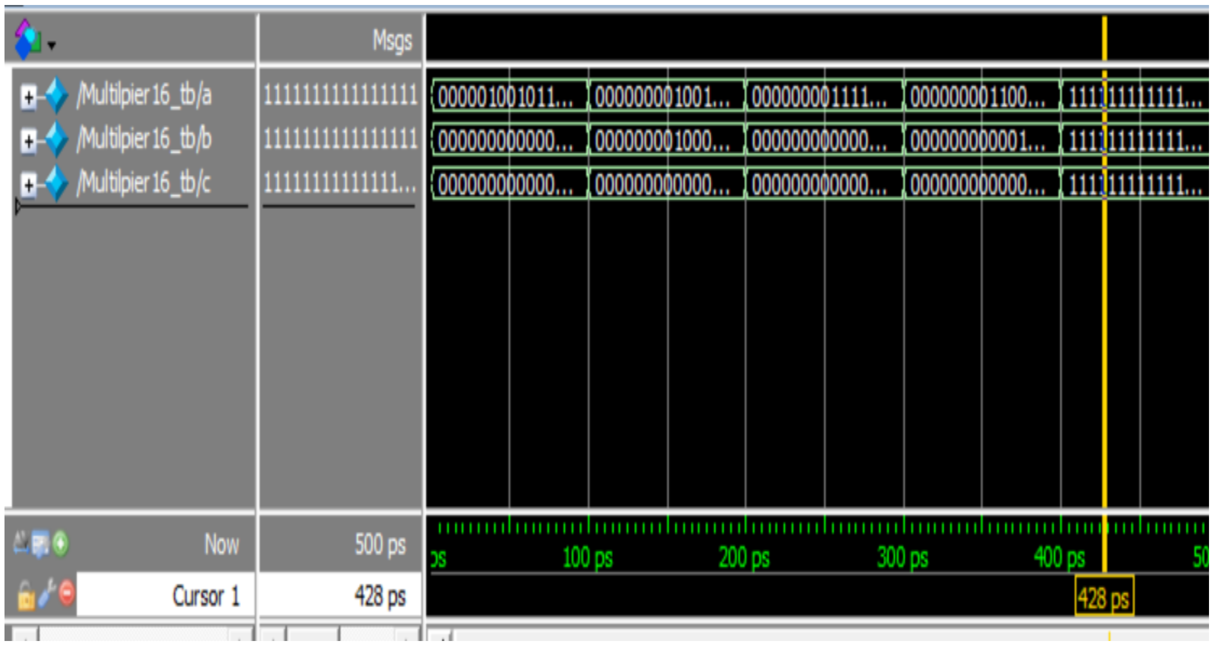


Fig.4.7 Simulation of 16*16 Vedic Multiplier

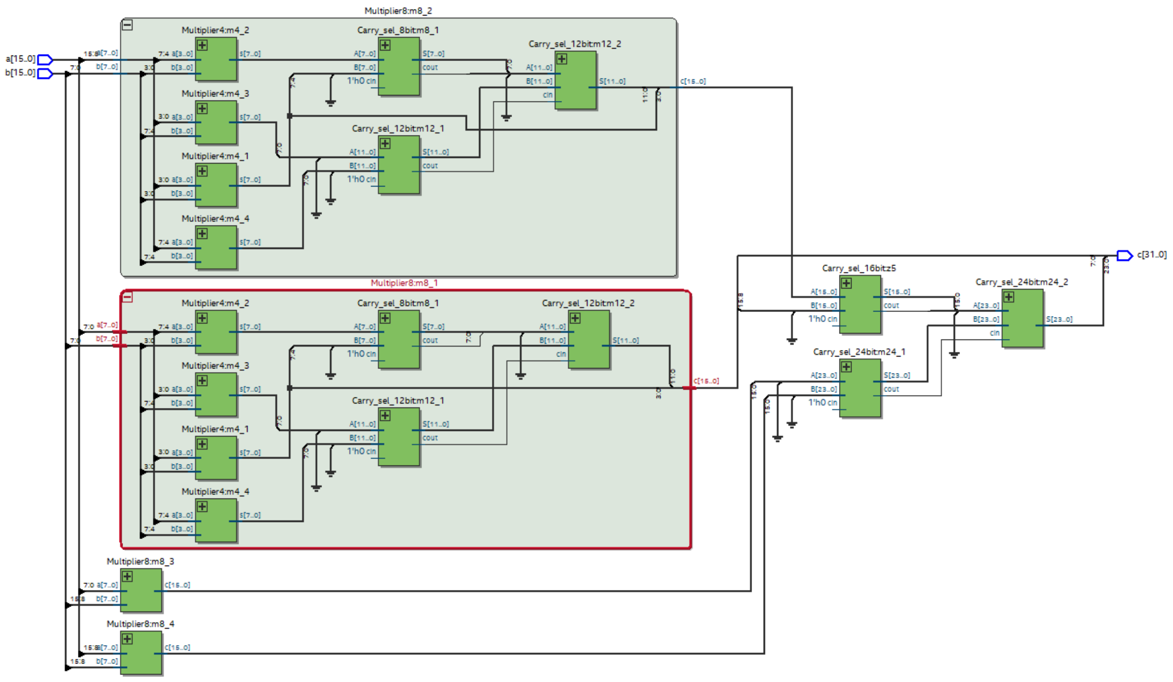


Fig.4.8 Netlist of 16*16 Vedic Multiplier

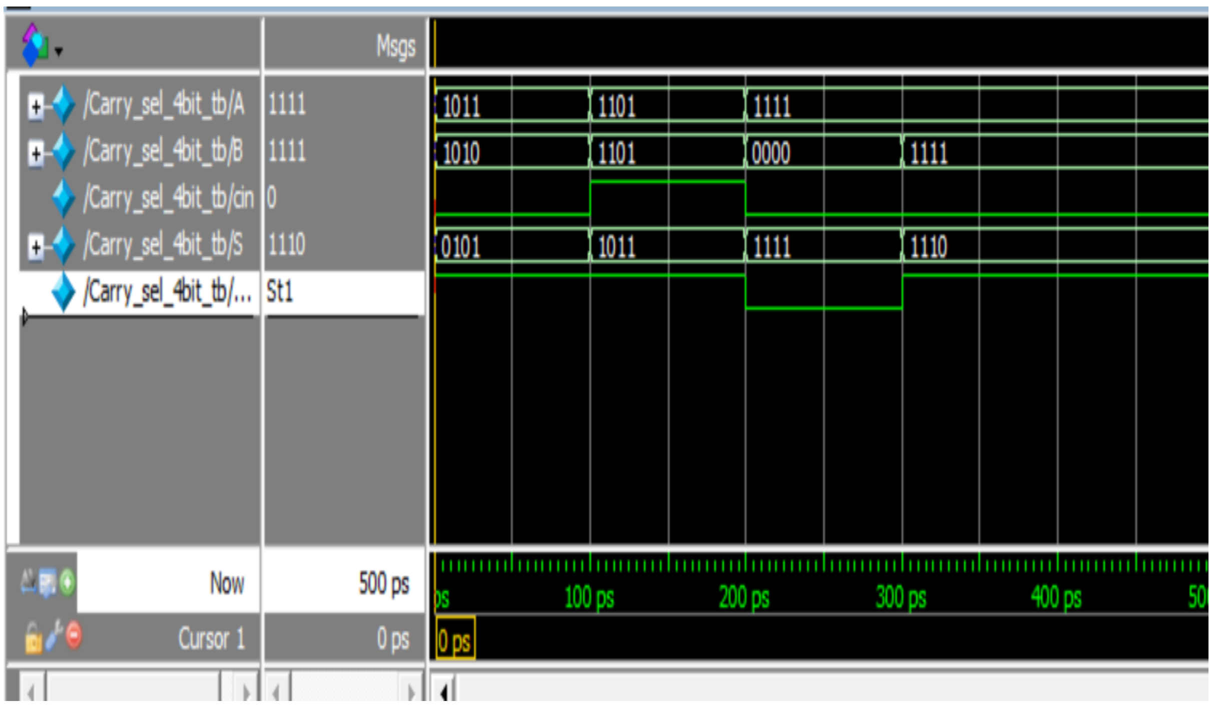


Fig.4.9 Simulation of 4-Bit Carry Select Adder

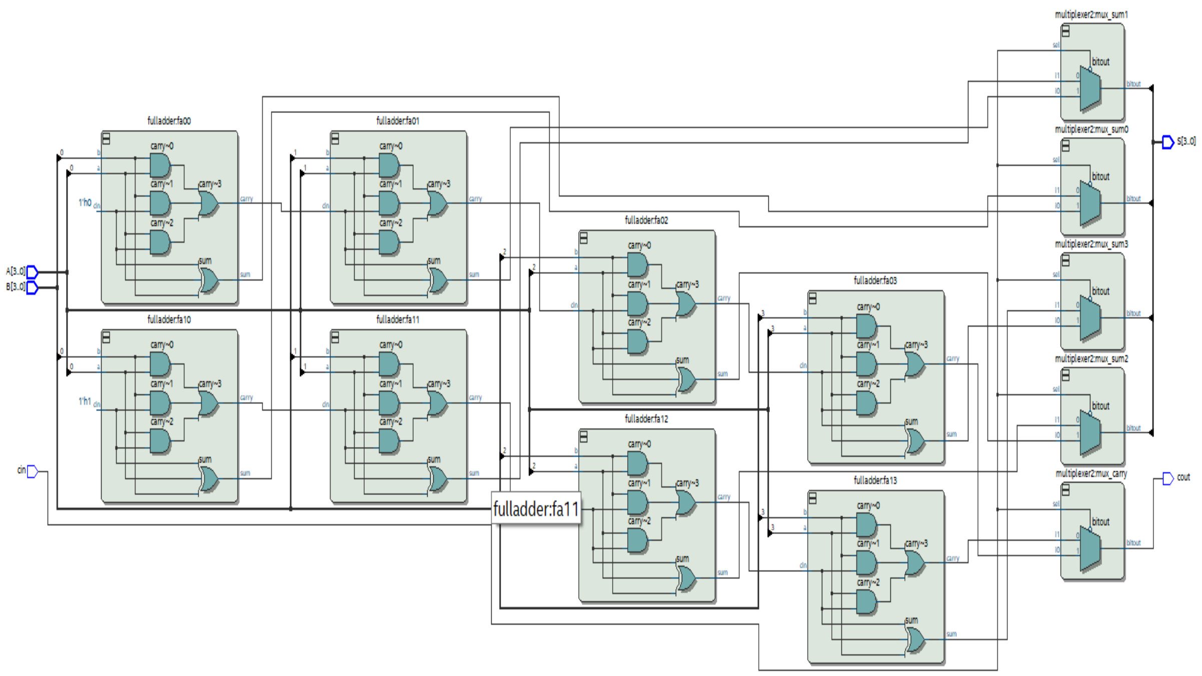


Fig.4.10 Netlist of 4-Bit Carry Select Adder

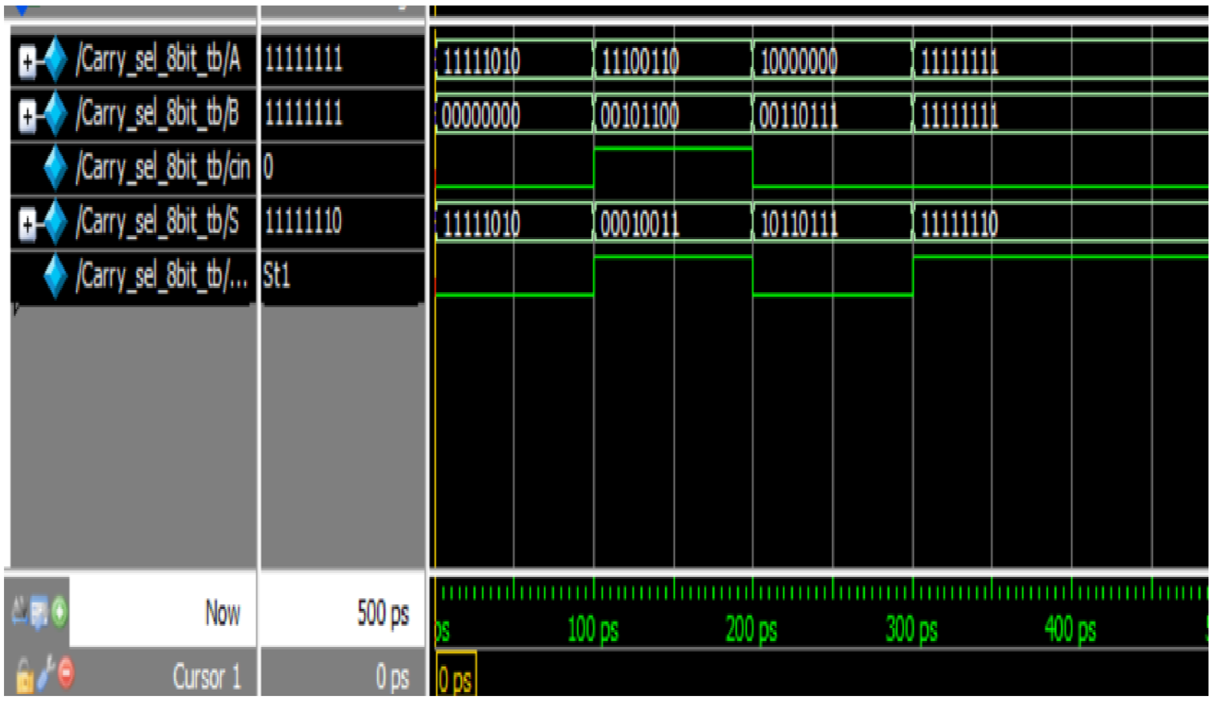


Fig.4.11 Simulation of 8-Bit Carry Select Adder

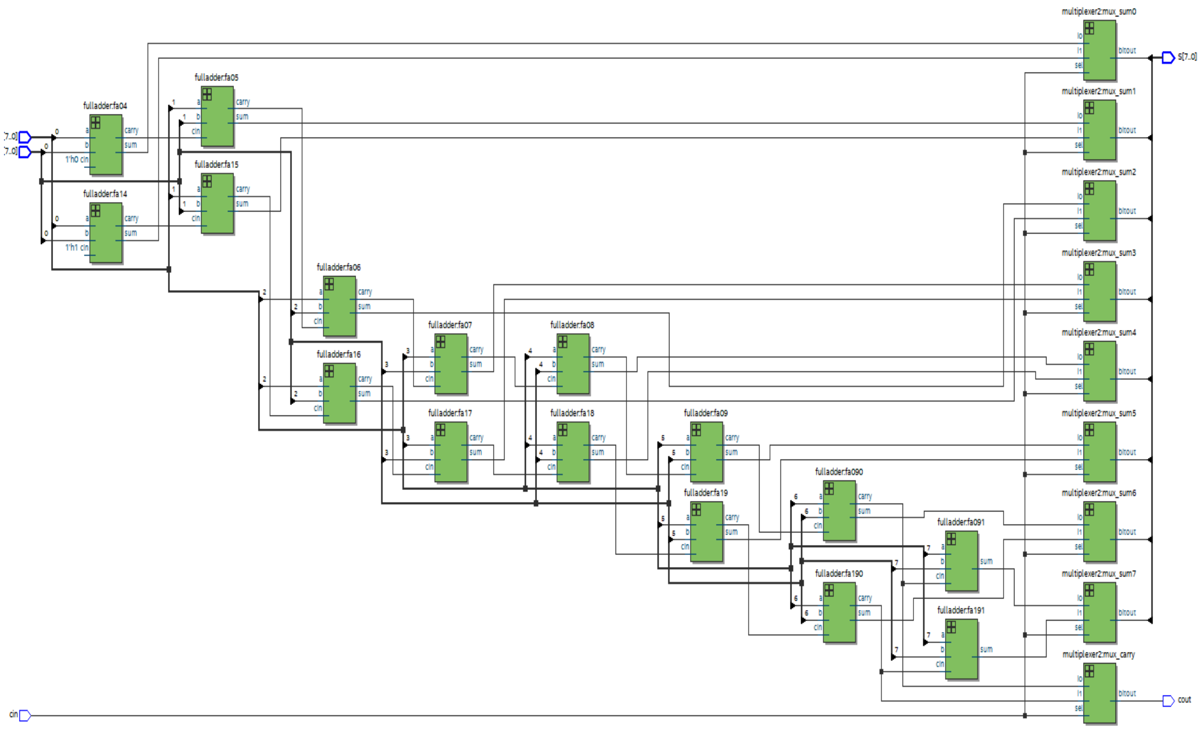
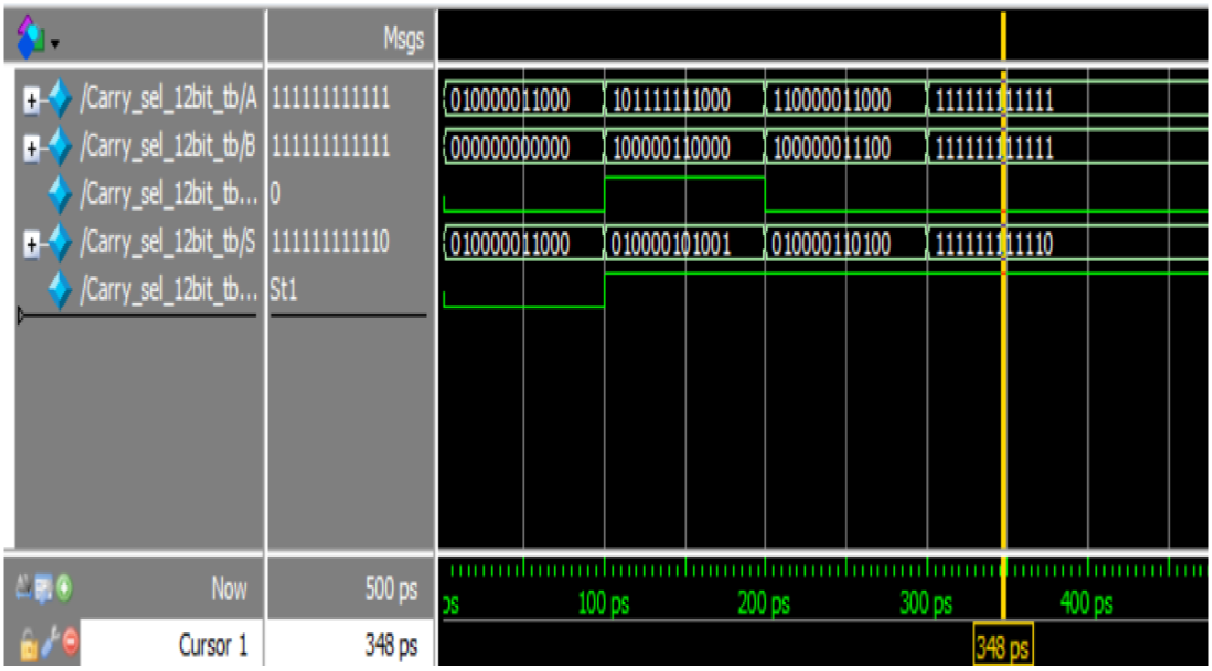


Fig.4.12 Netlist of 8-Bit Carry Select Adder



```
                          Msgs
/Carry_sel_12bit_tb/A   111111111111   010000011000  101111111000  110000011000  111111111111
/Carry_sel_12bit_tb/B   111111111111   000000000000  100000110000  100000011100  111111111111
/Carry_sel_12bit_tb...  0
/Carry_sel_12bit_tb/S   111111111110   010000011000  010000101001  010000110100  111111111110
/Carry_sel_12bit_tb...  St1

Now        500 ps      ps   100 ps   200 ps   300 ps   400 ps
Cursor 1   348 ps                                348 ps
```

Fig.4.13 Simulation of 12-Bit Carry Select Adder

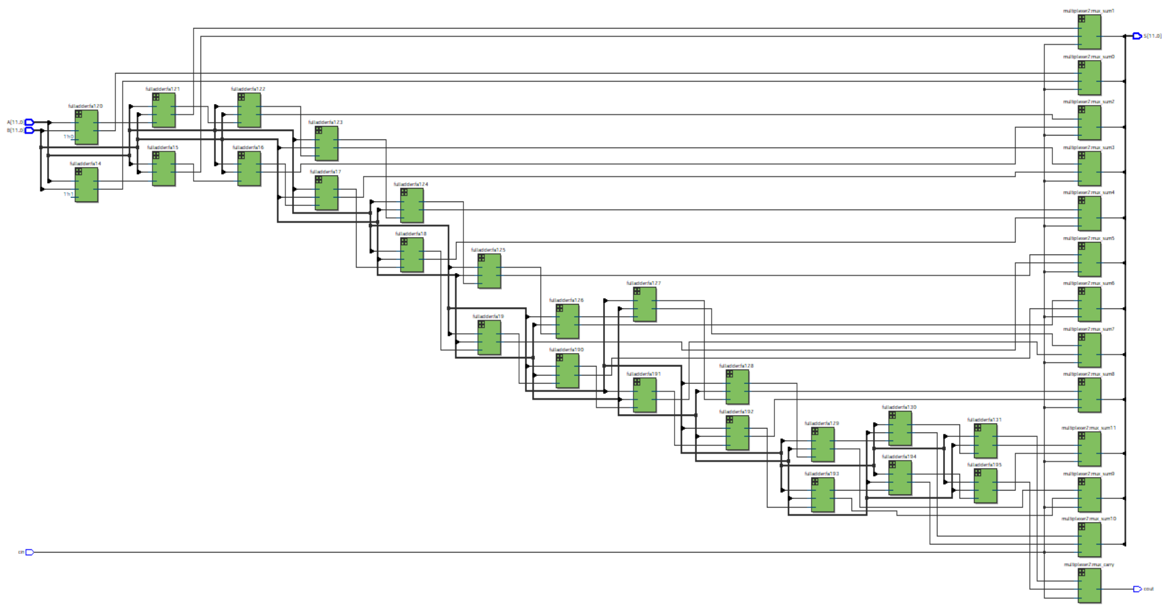


Fig.4.14 Netlist of 12-Bit Carry Select Adder

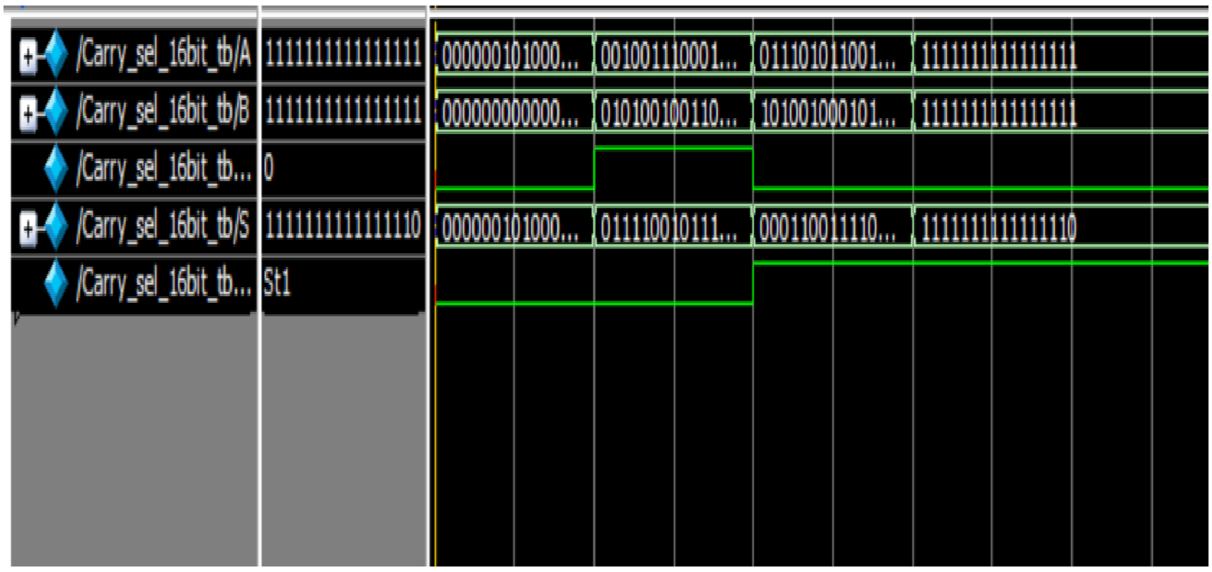


Fig.4.15 Simulation of 16-Bit Carry Select Adder

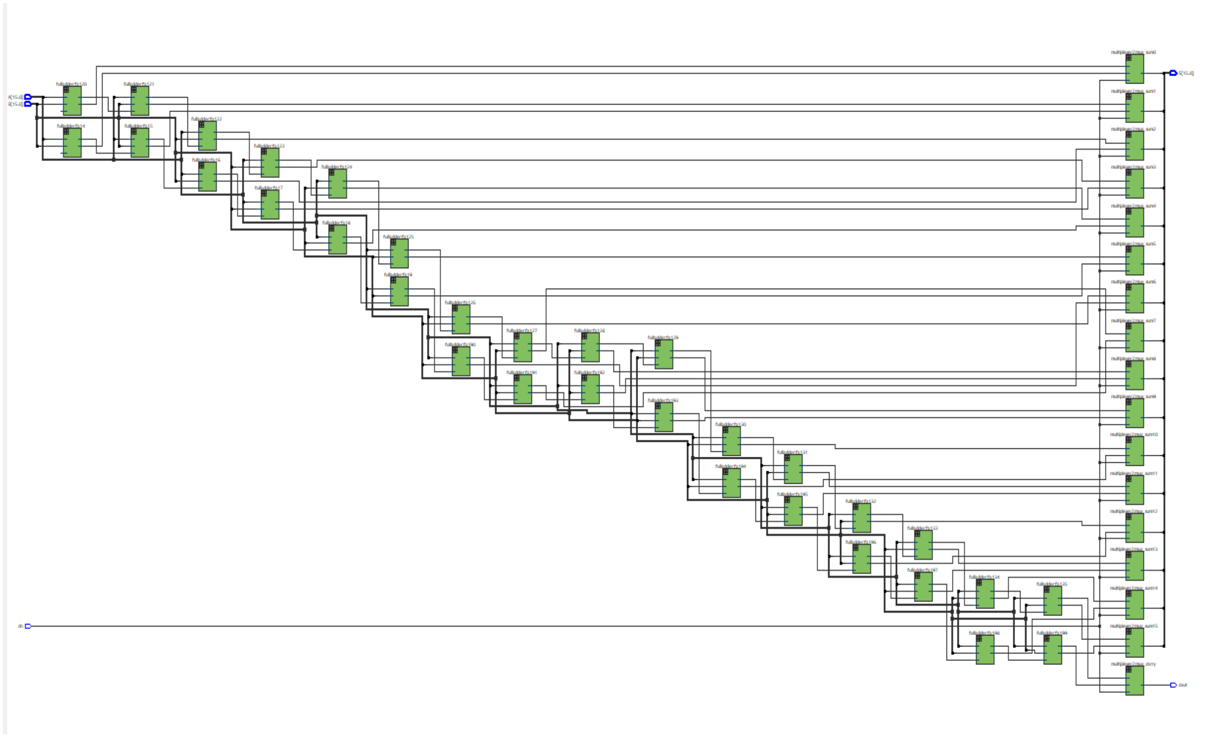


Fig.4.16 Netlist of 16-Bit Carry Select Adder

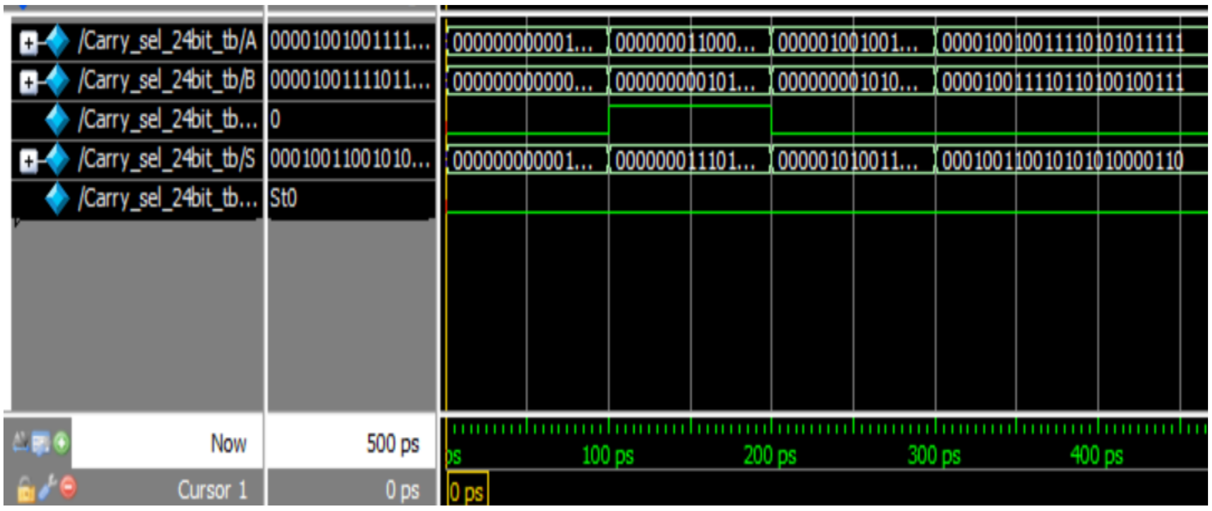

```
/Carry_sel_24bit_tb/A  00001001001111...  000000000001...  000000011000...  000001001001...  000010010011110101011111
/Carry_sel_24bit_tb/B  00001001111011...  000000000000...  000000000101...  000000001010...  000010011110110100100111
/Carry_sel_24bit_tb... 0
/Carry_sel_24bit_tb/S  00010011001010...  000000000001...  000000011101...  000001010011...  000100110010101010000110
/Carry_sel_24bit_tb... St0
Now       500 ps     ps   100 ps   200 ps   300 ps   400 ps
Cursor 1  0 ps       0 ps
```

Fig.4.17 Simulation of 24-Bit Carry Select Adder

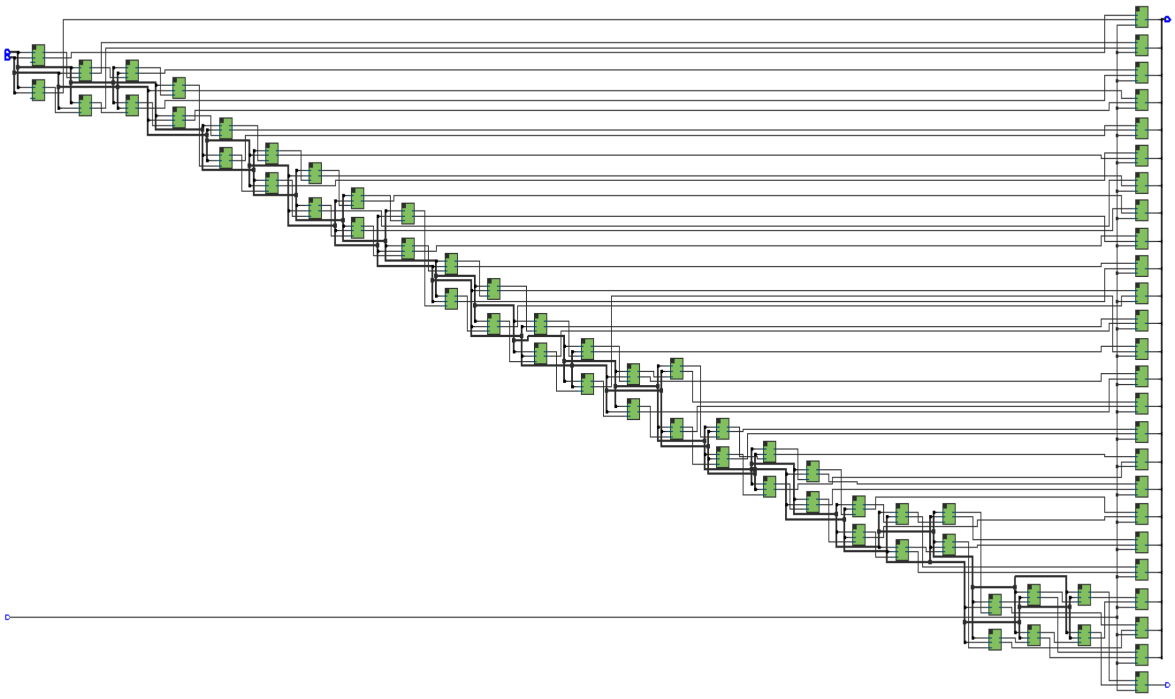


Fig.4.18 Netlist of 24-Bit Carry Select Adder

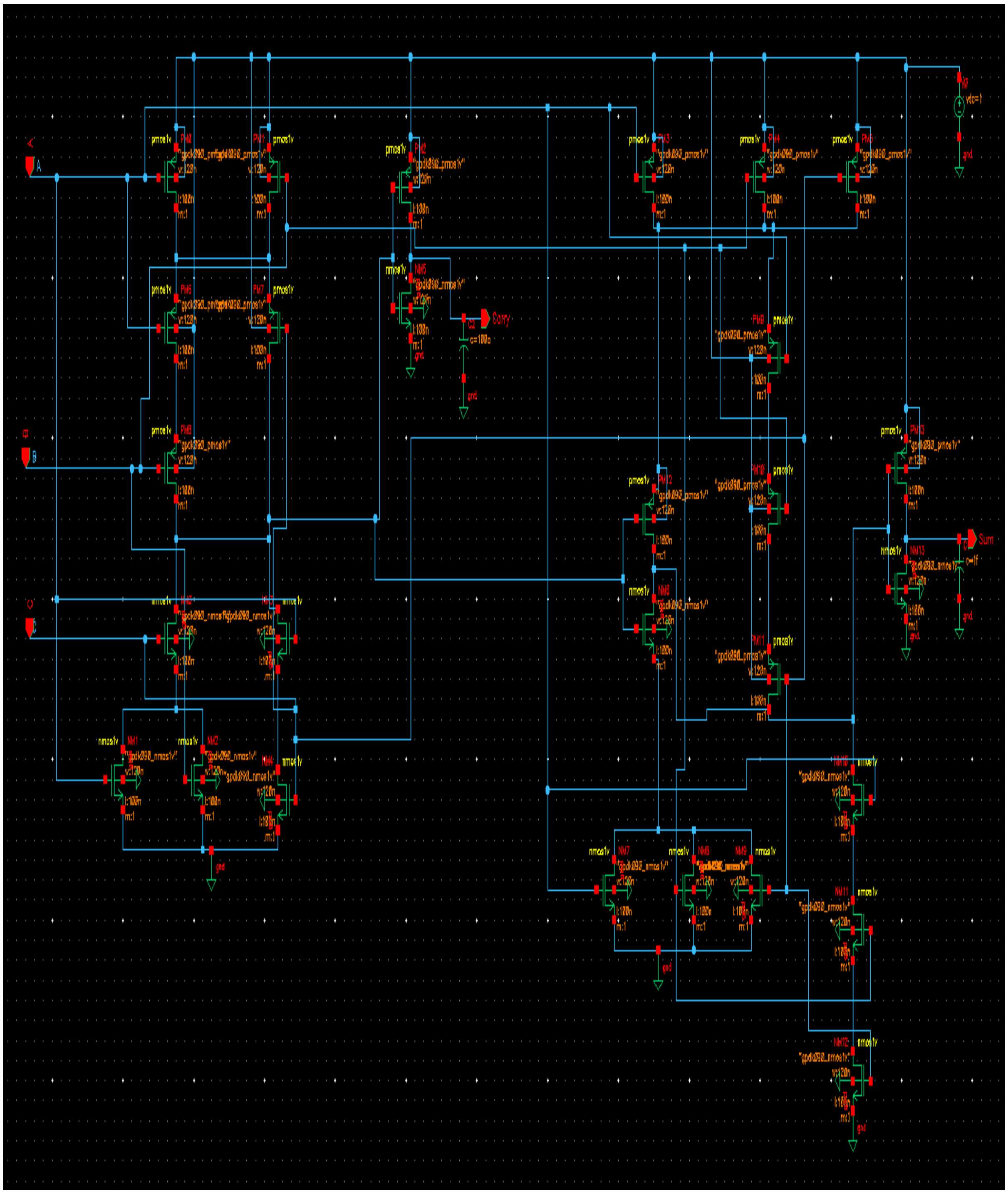


Fig.4.19 Schematic of 1-Bit full Adder using Static CMOS.

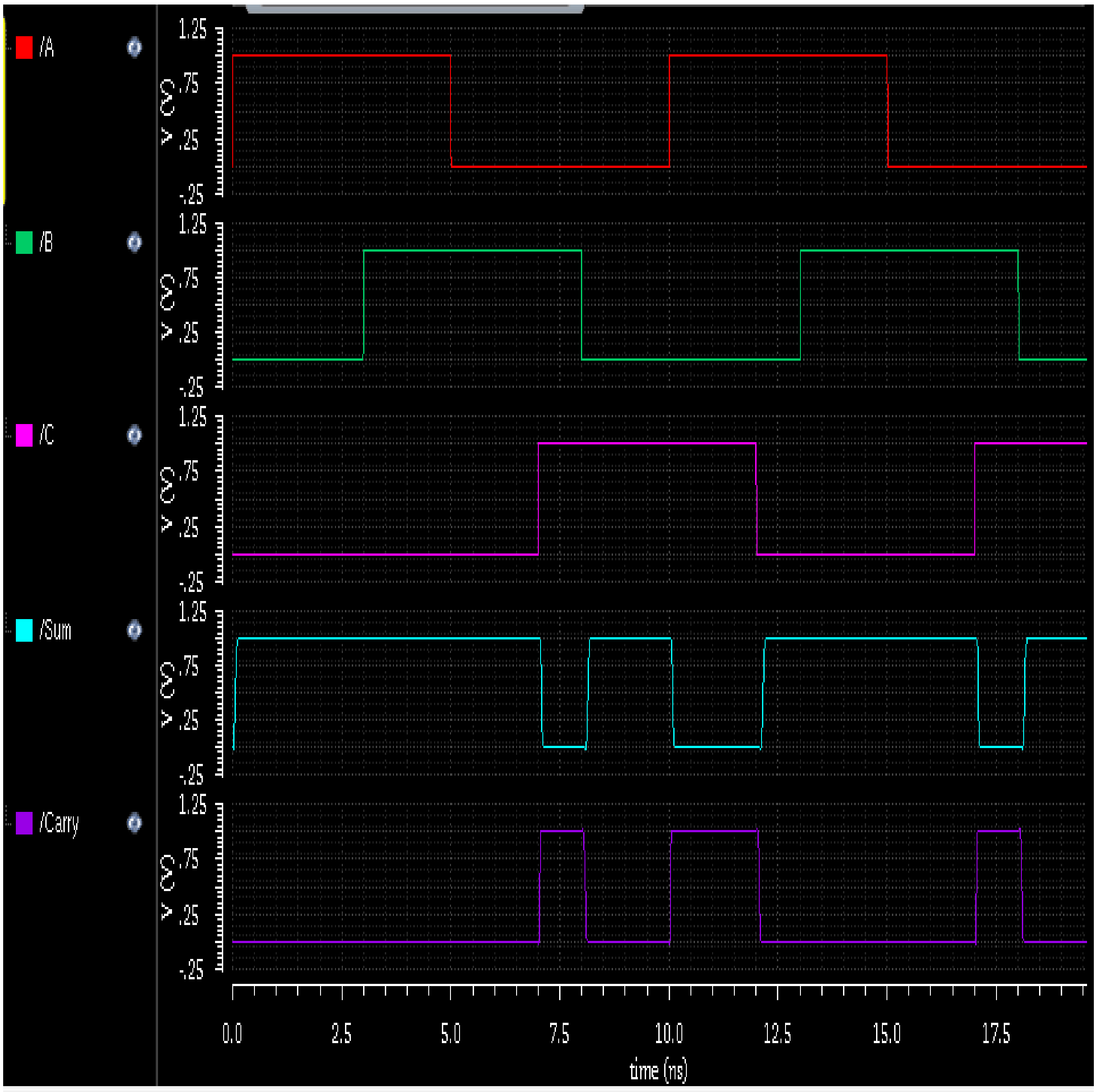


Fig.4.20 Inputs and Output of 1-Bit Full Adder using Static CMOS logic

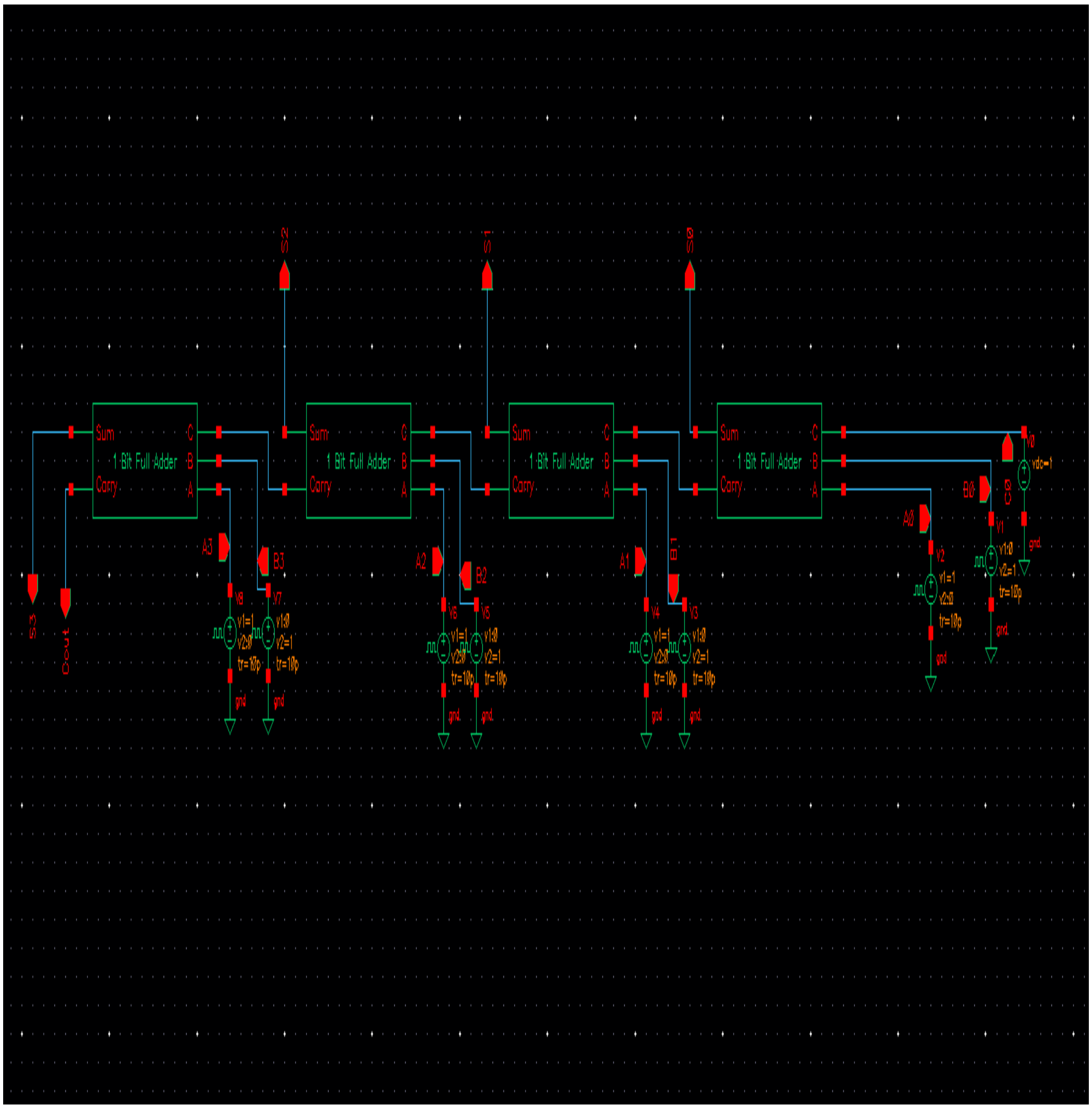


Fig.4.21 Schematic of 4-Bit Ripple Carry Adder using Static CMOS

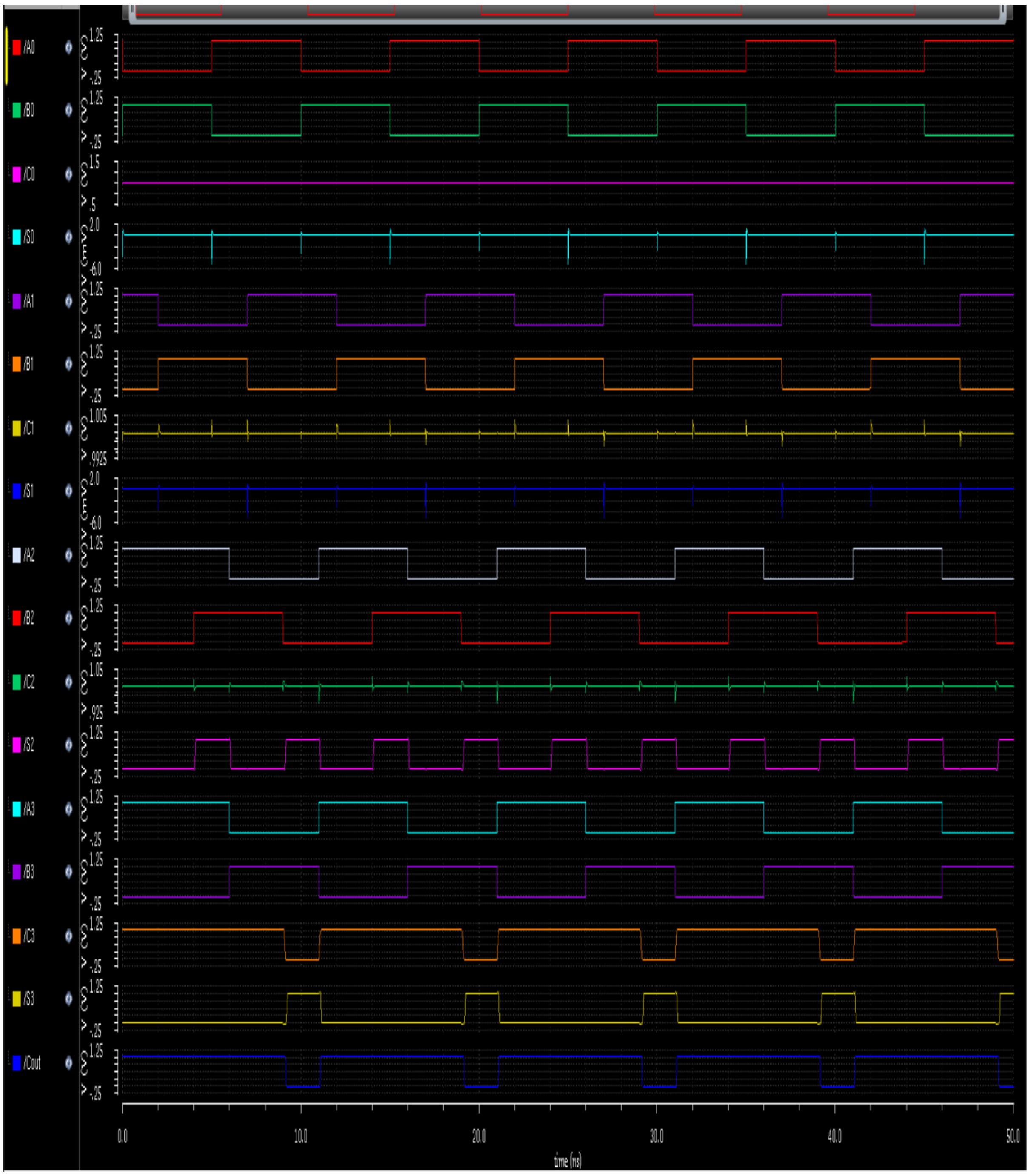


Fig.4.22 Inputs and Output of 4-Bit Ripple Carry Adder using Static CMOS logic

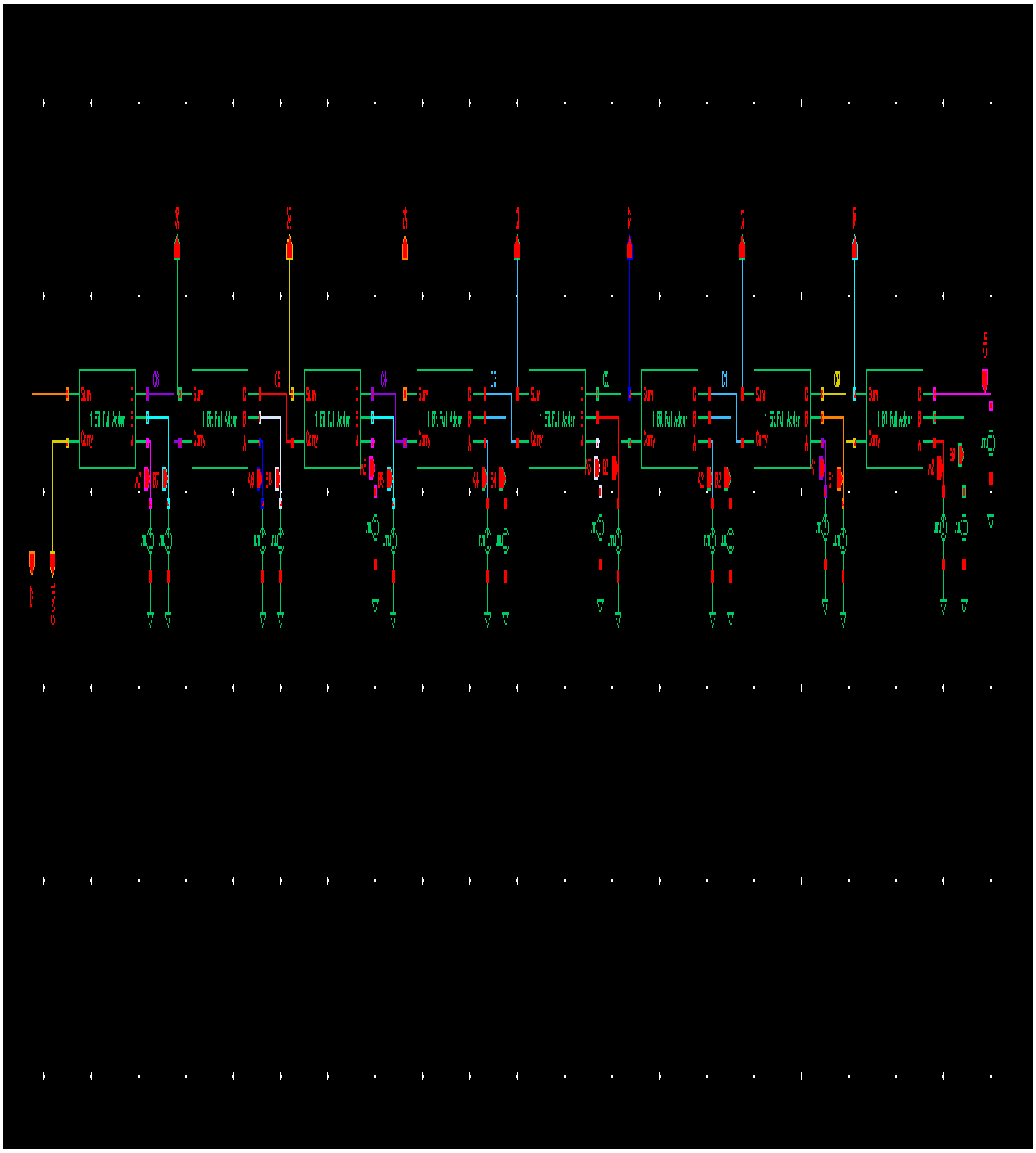


Fig.4.23 Schematic of 8-Bit Ripple Carry Adder using Static CMOS logic

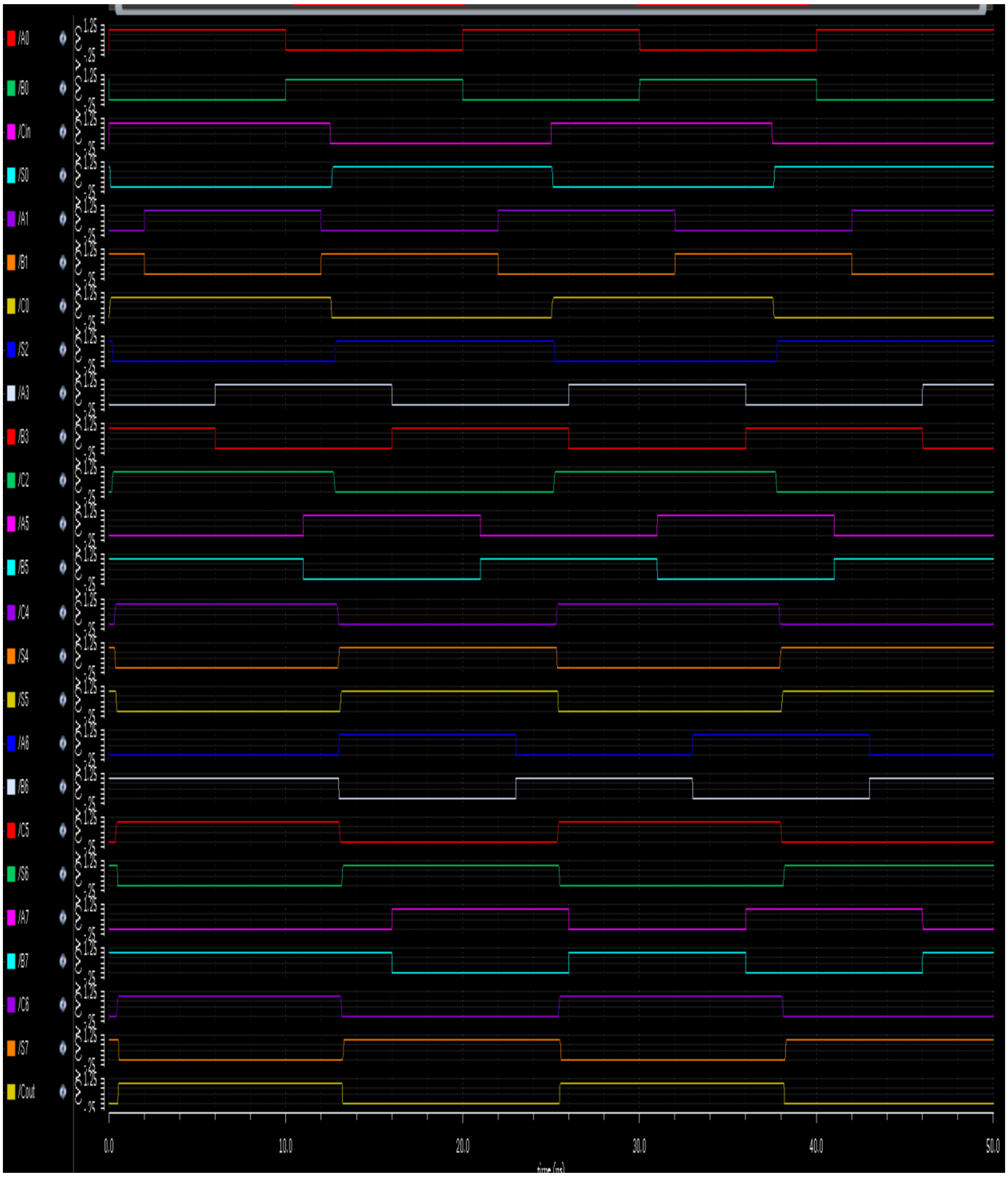


Fig.4.24 Inputs and Output of 8-Bit Ripple Carry Adder using Static CMOS logic

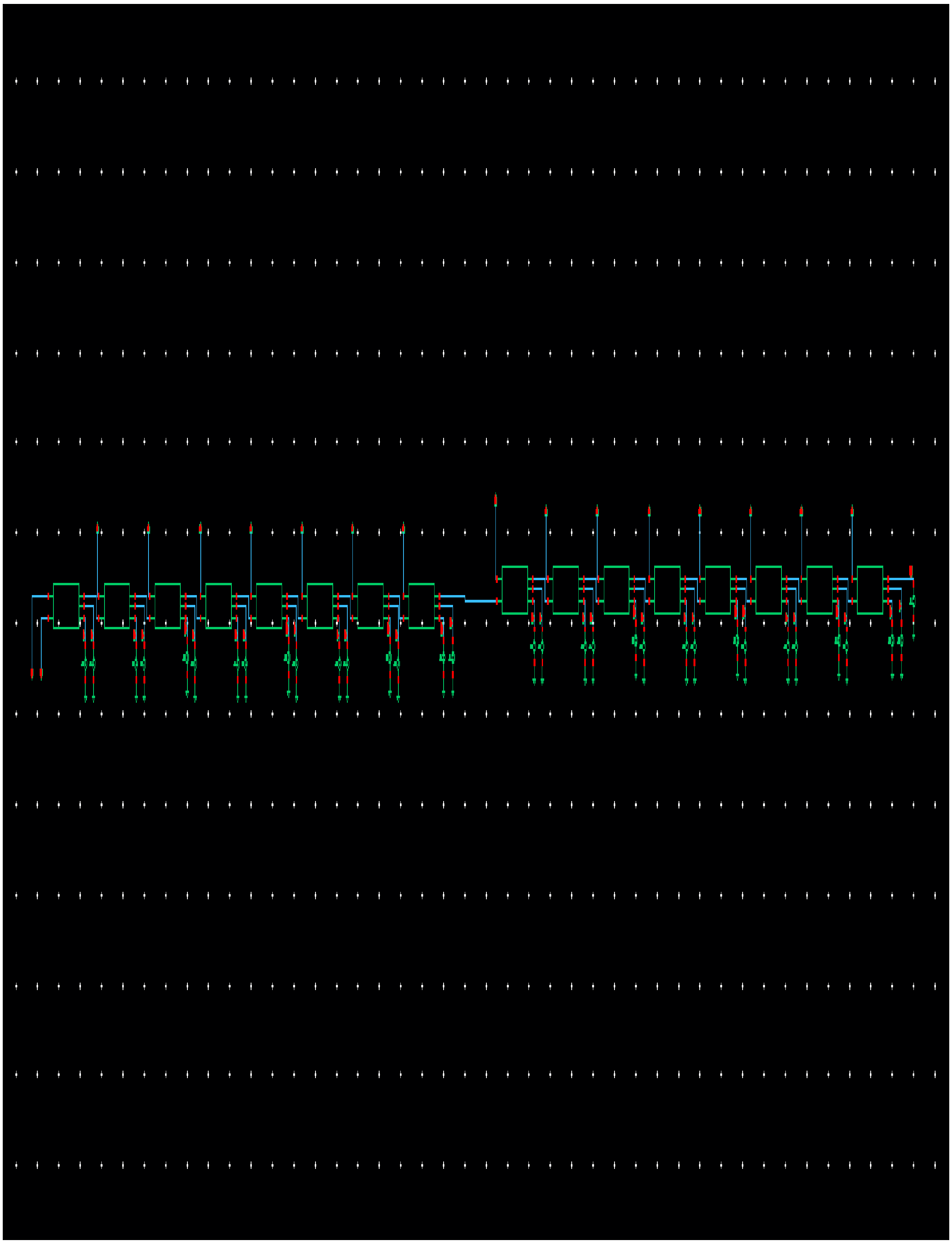


Fig.4.25 Schematic of 16-Bit Ripple Carry Adder using Static CMOS logic

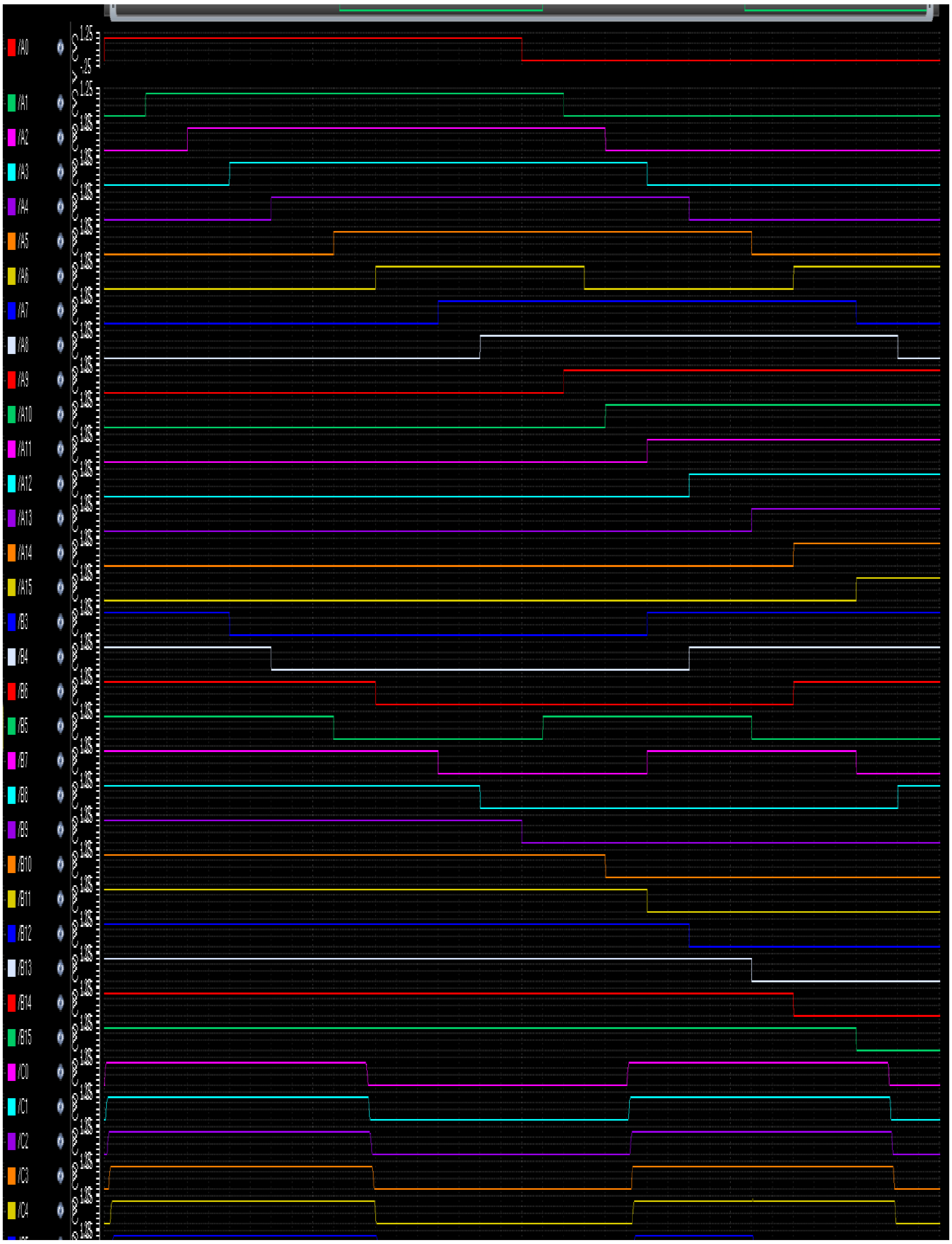


Fig.4.26 Inputs and Outputs of 16-Bit Full Adder

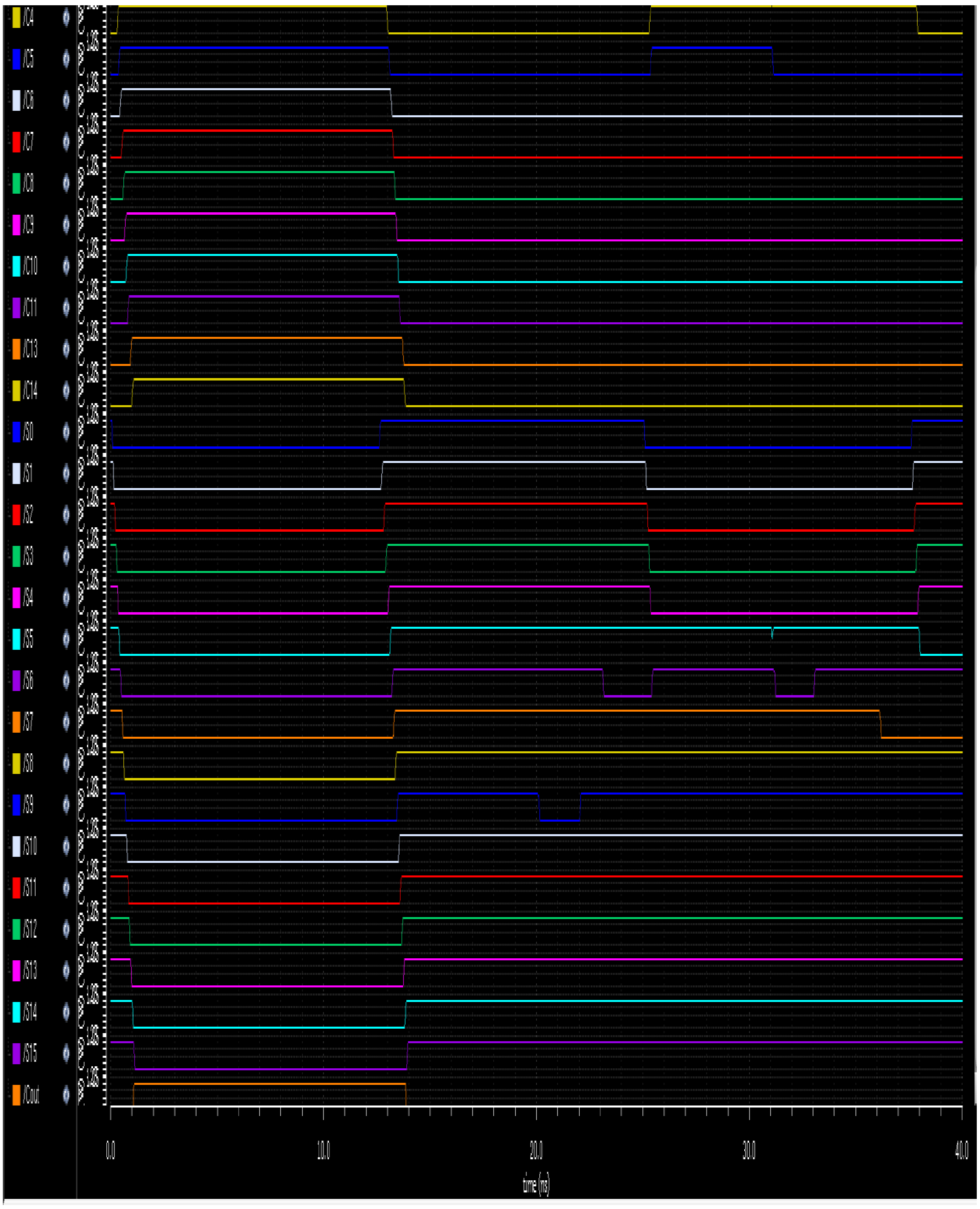


Fig.4.27 Inputs and Outputs of 16-Bit Full Adder

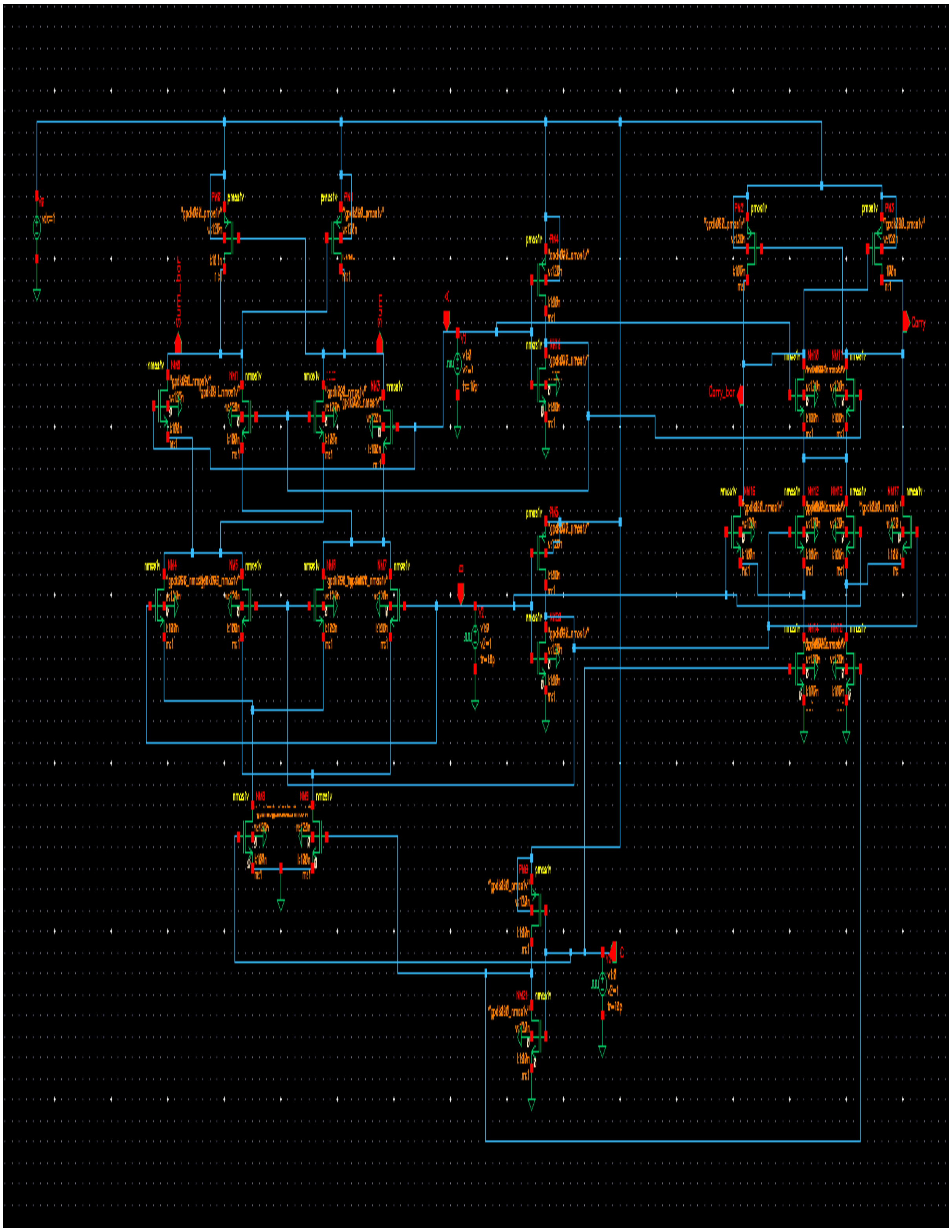


Fig.4.28 Schematic of 1-Bit full Adder using Differential Cascode Logic.

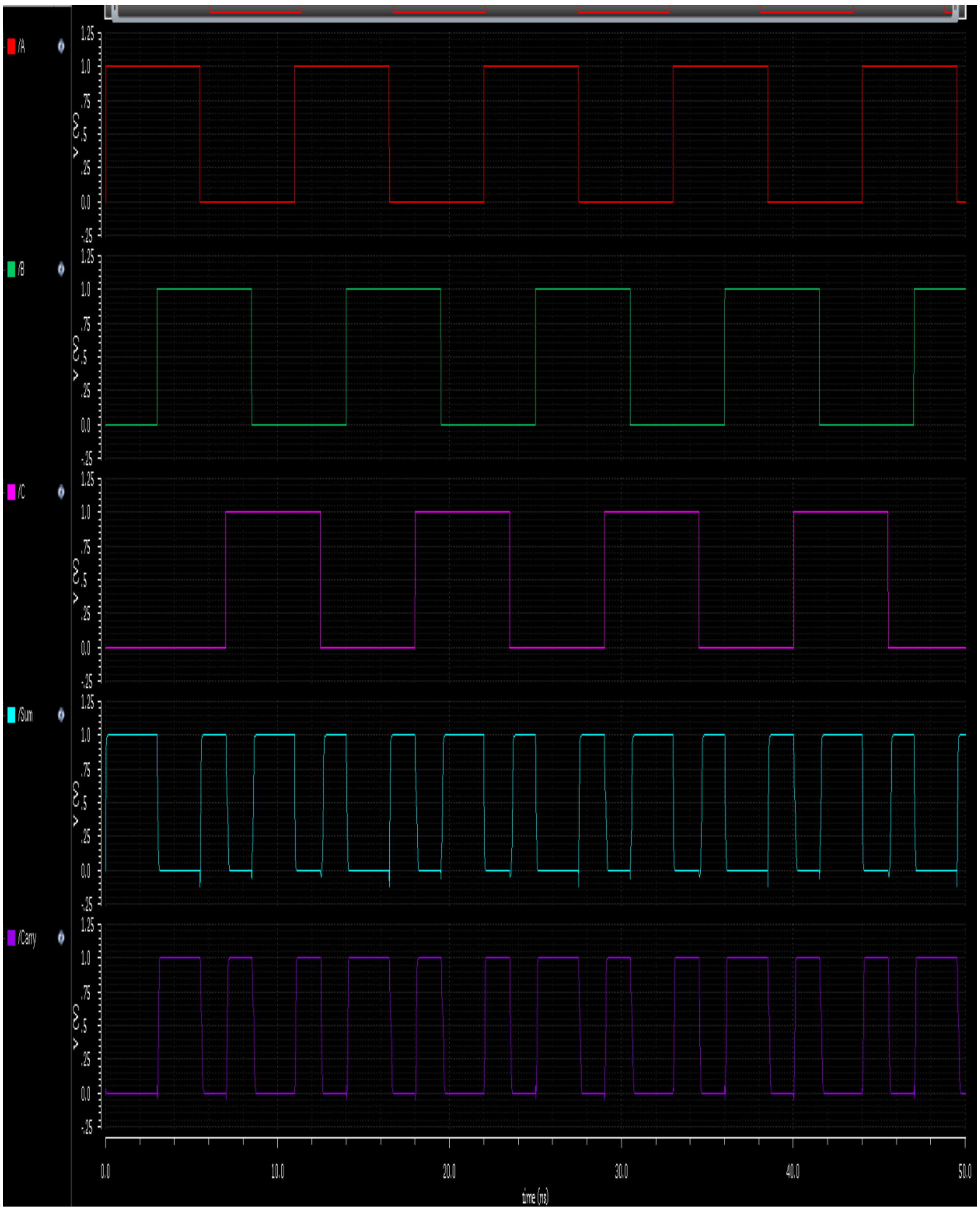


Fig.4.29 Inputs and Output of 1-Bit Full Adder using Differential Cascode logic

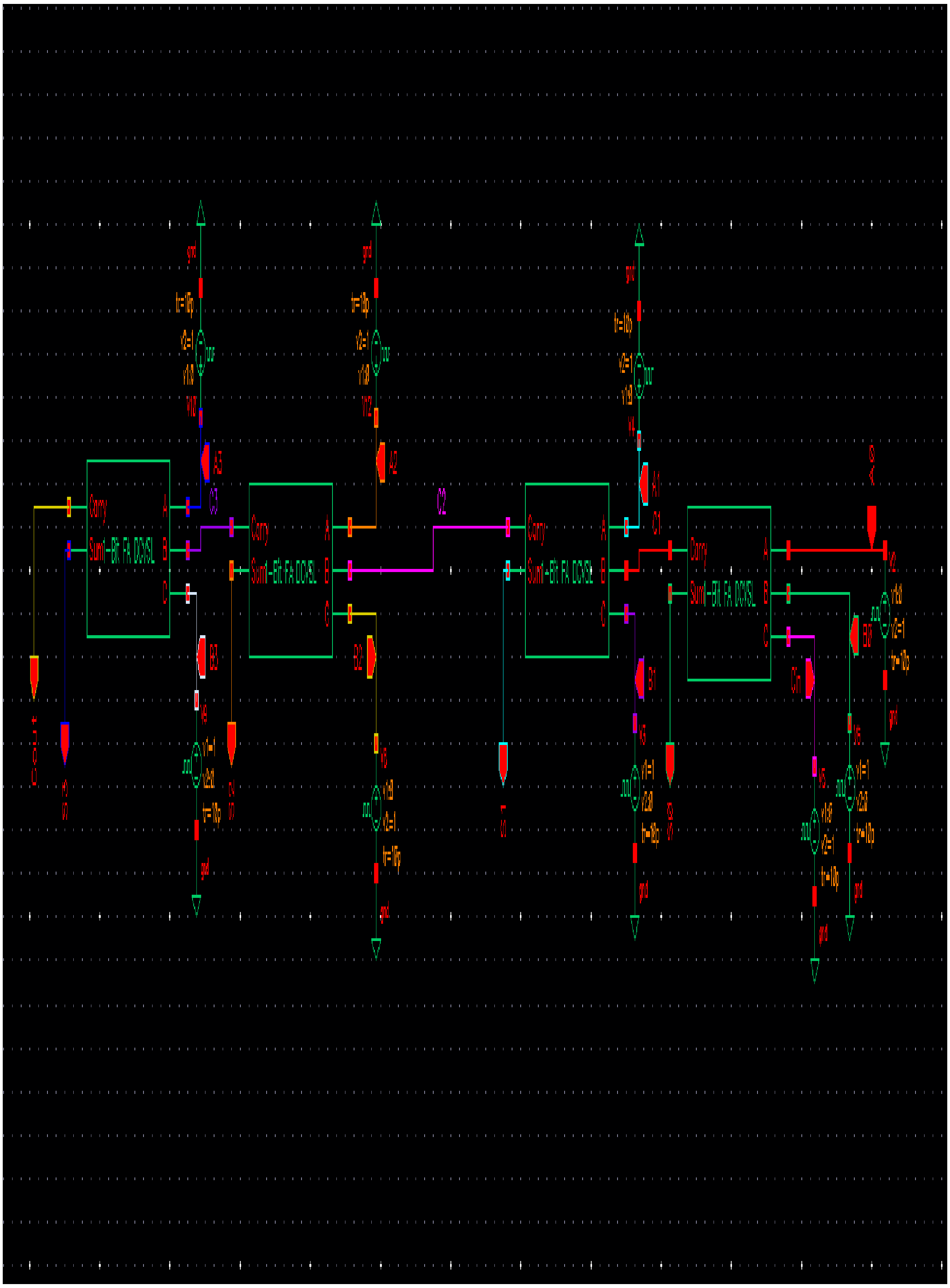


Fig.4.30 Schematic of 4-Bit full Adder using Differential Cascode Logic

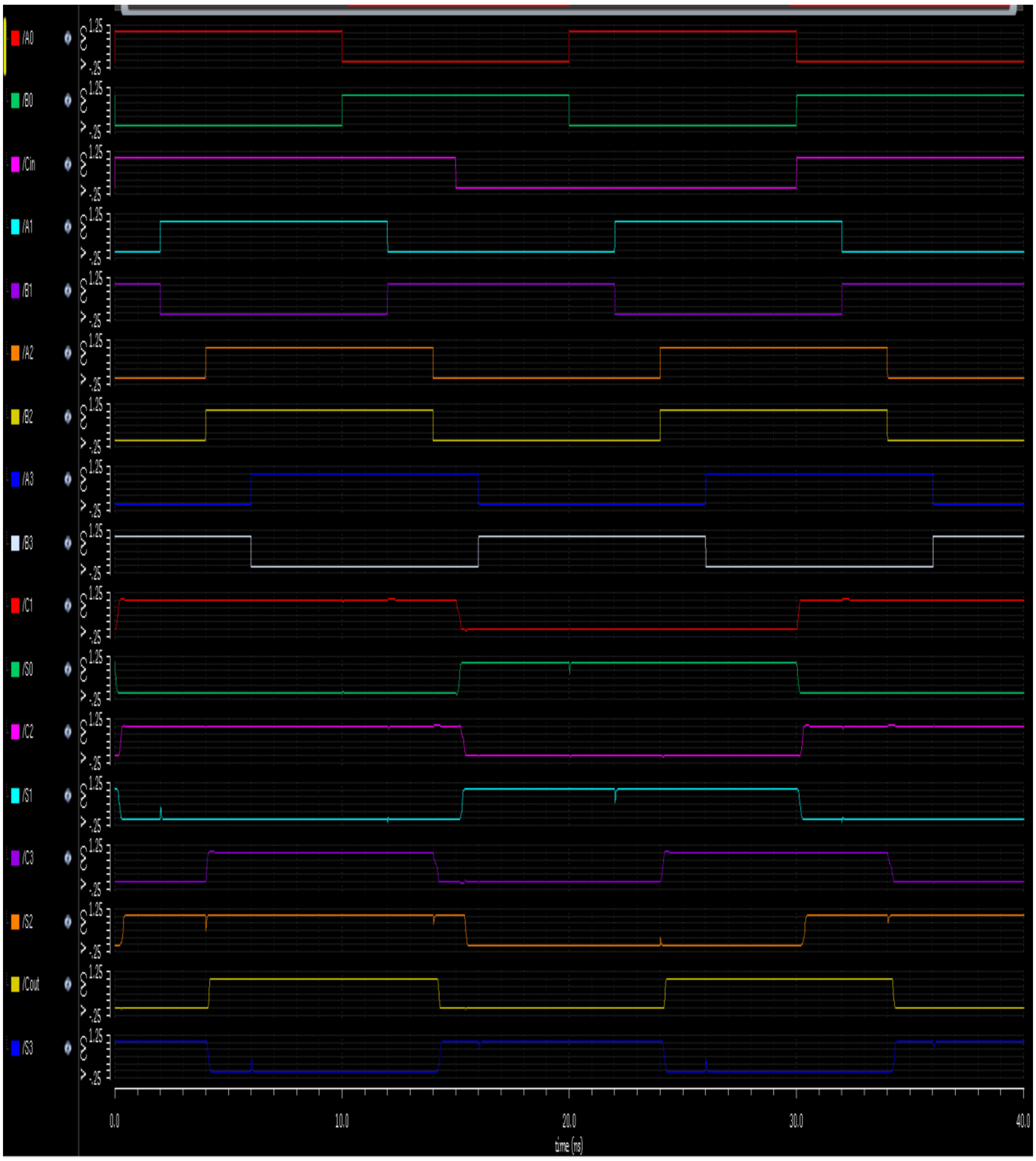


Fig.4.31 Inputs and Output of 4-Bit Full Adder using Differential Cascode logic

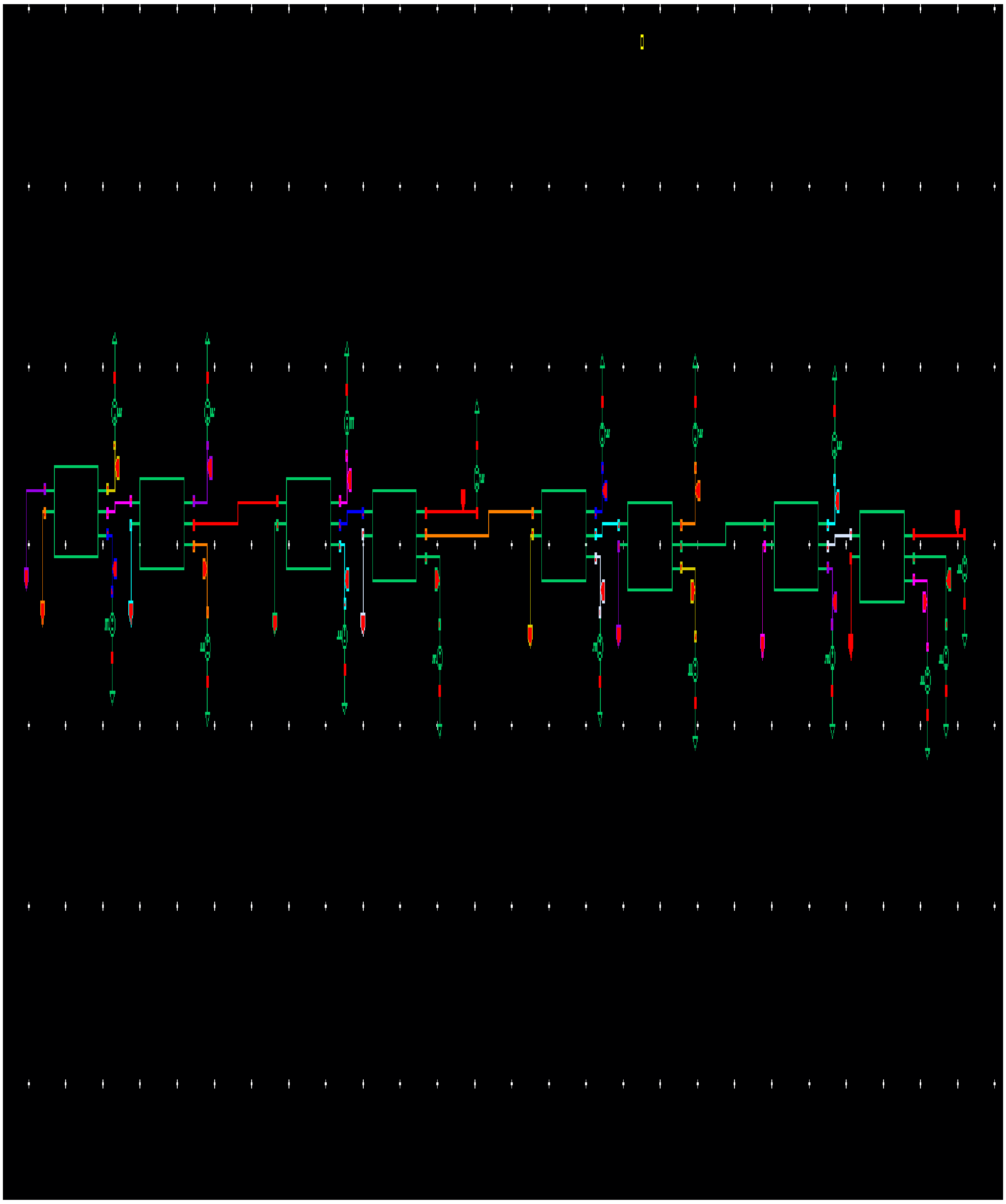


Fig.4.32 Schematic of 8-Bit full Adder using Differential Cascode Logic

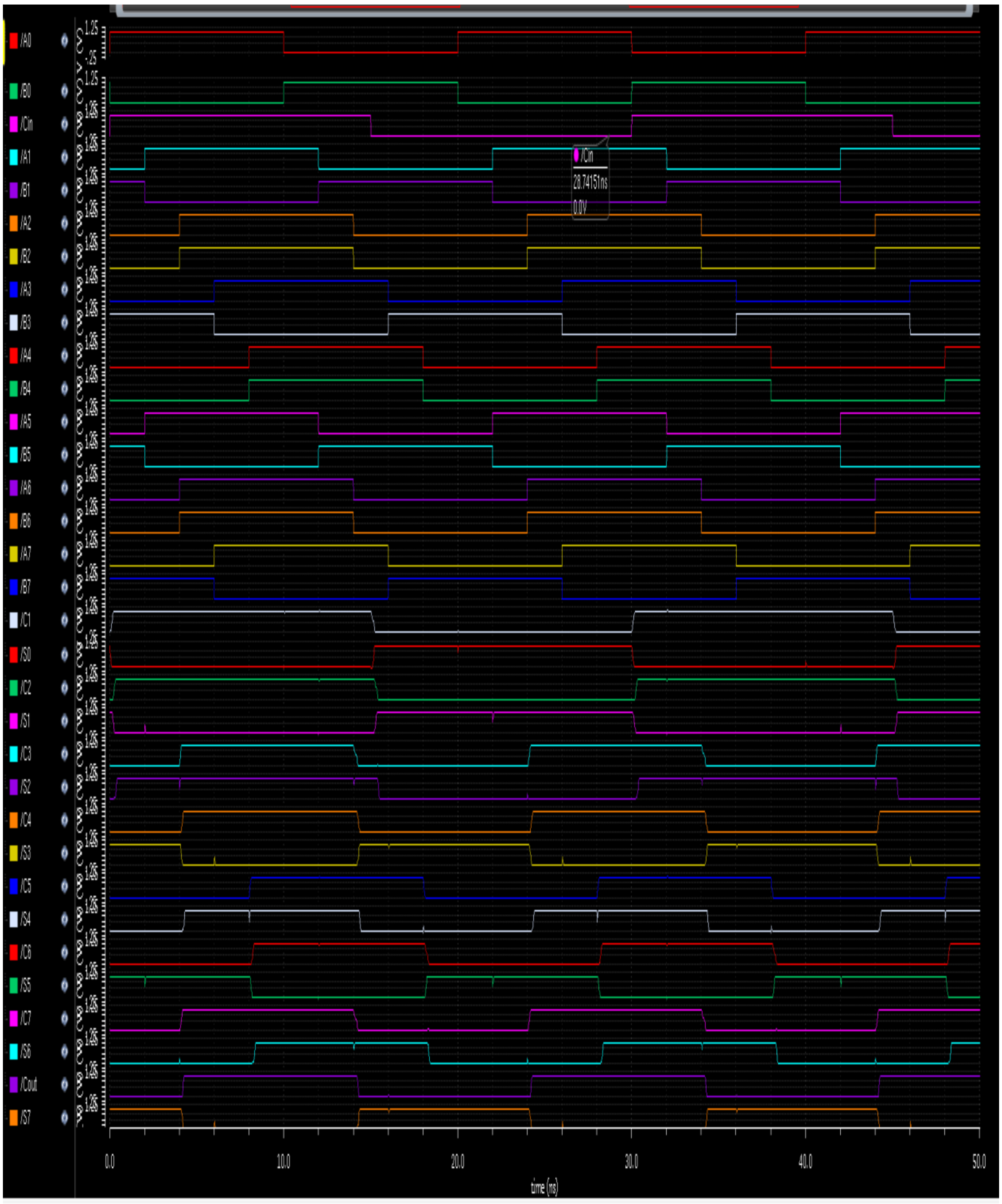


Fig.4.33 Inputs and Output of 8-Bit Full Adder using Differential Cascode logic

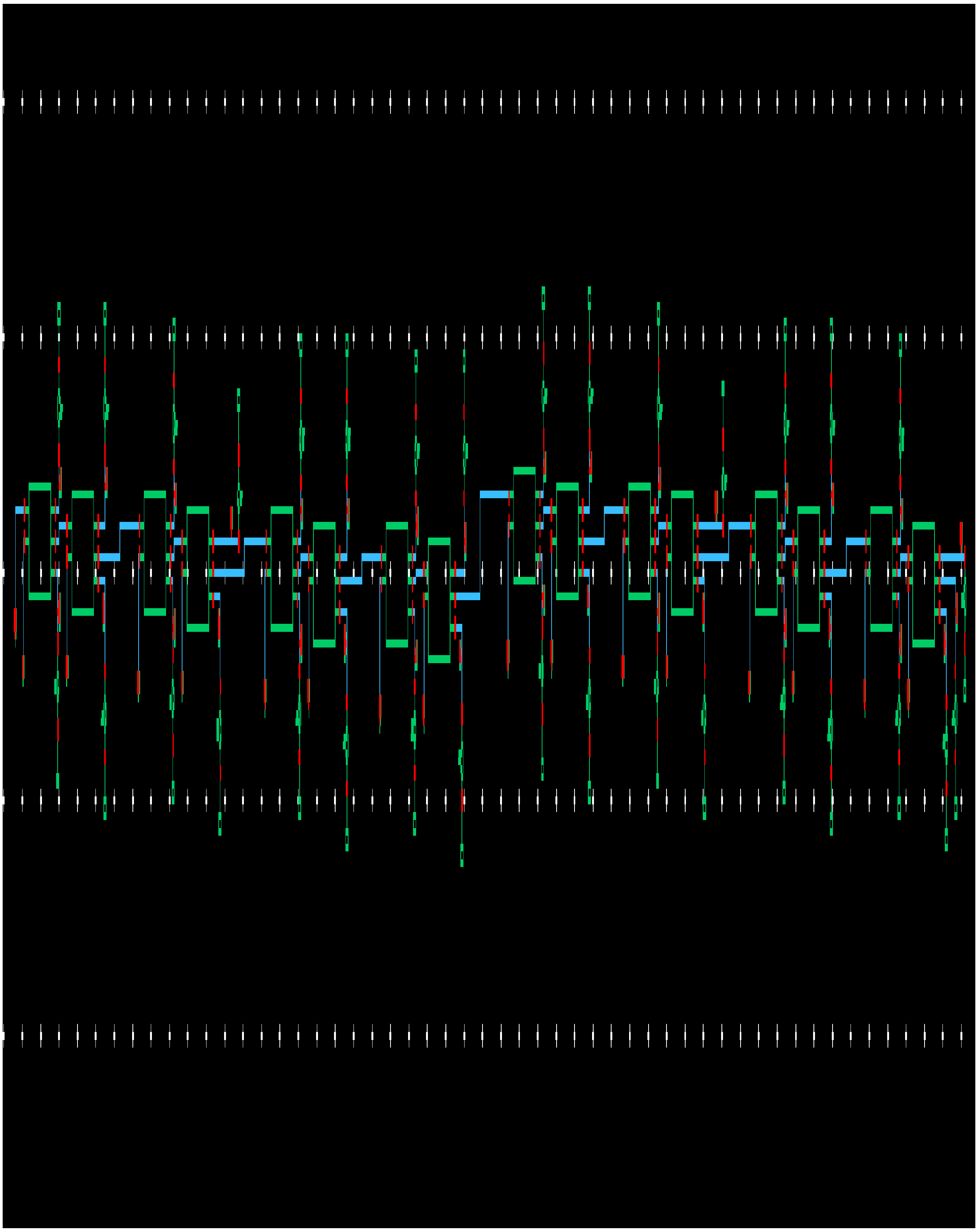


Fig.4.34 Schematic of 16-Bit full Adder using Differential Cascode Logic

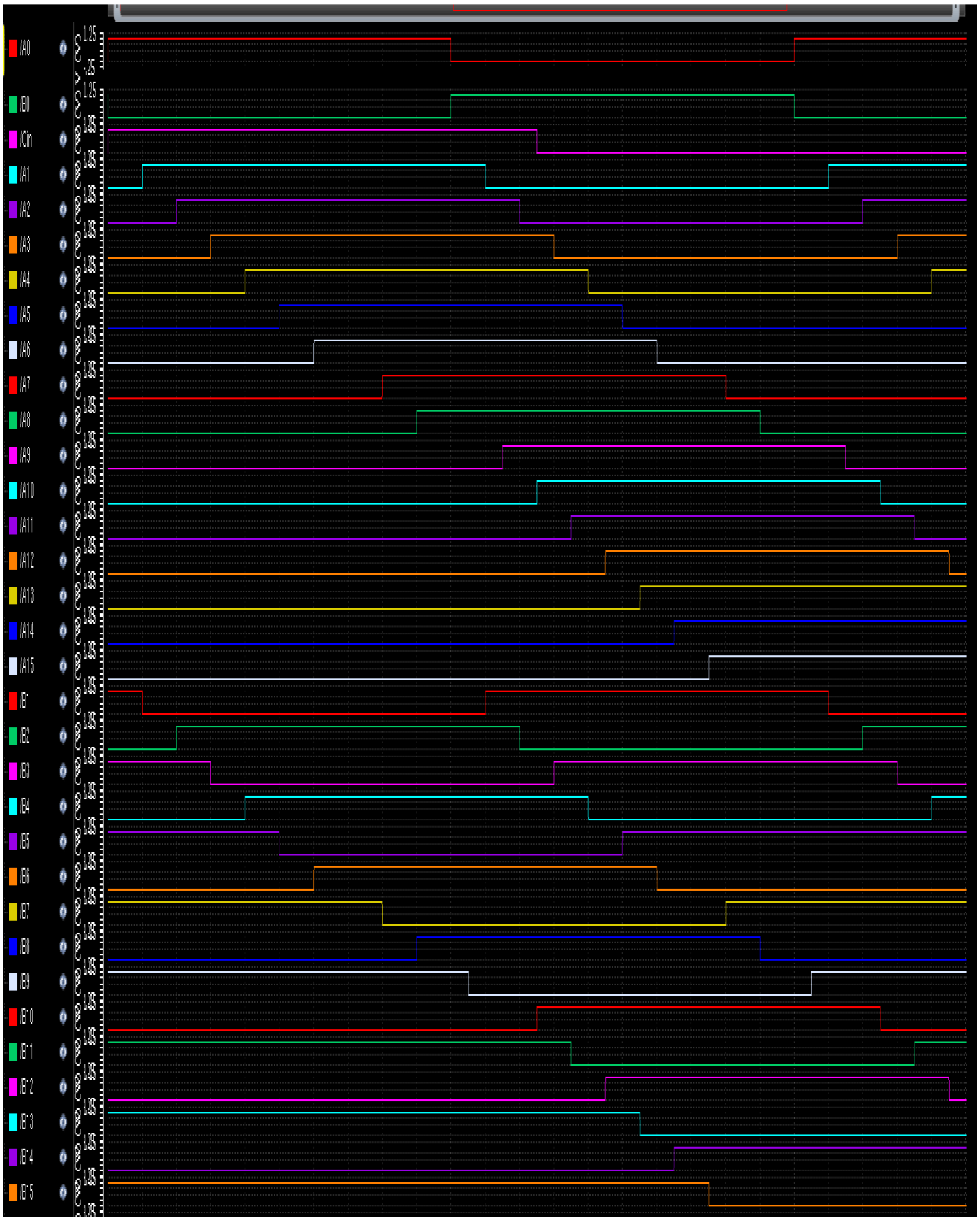


Fig.4.35 Inputs of 16-Bit Full Adder using Differential Cascode logic

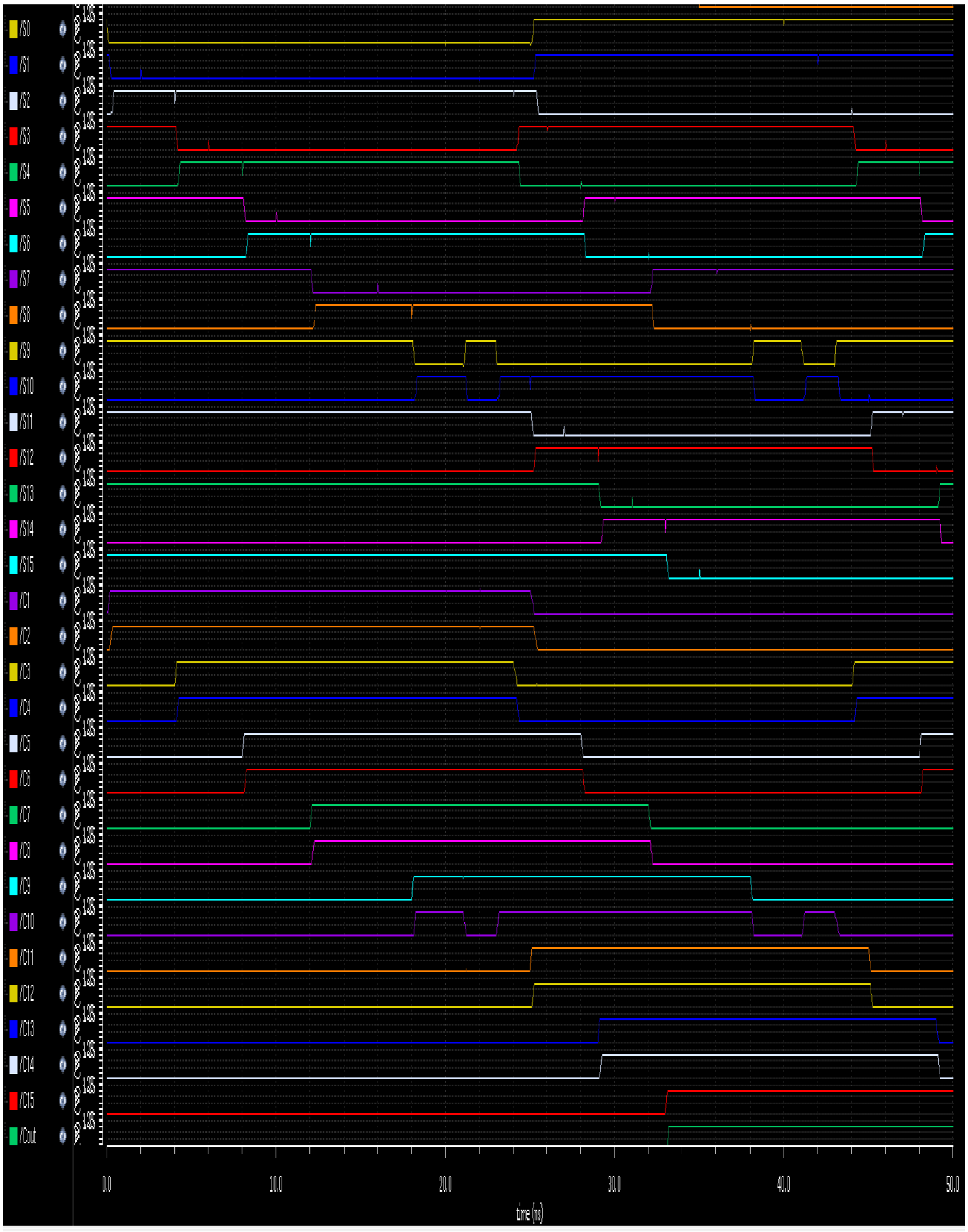


Fig.4.36 Outputs of 16-Bit Full Adder using Differential Cascode logic

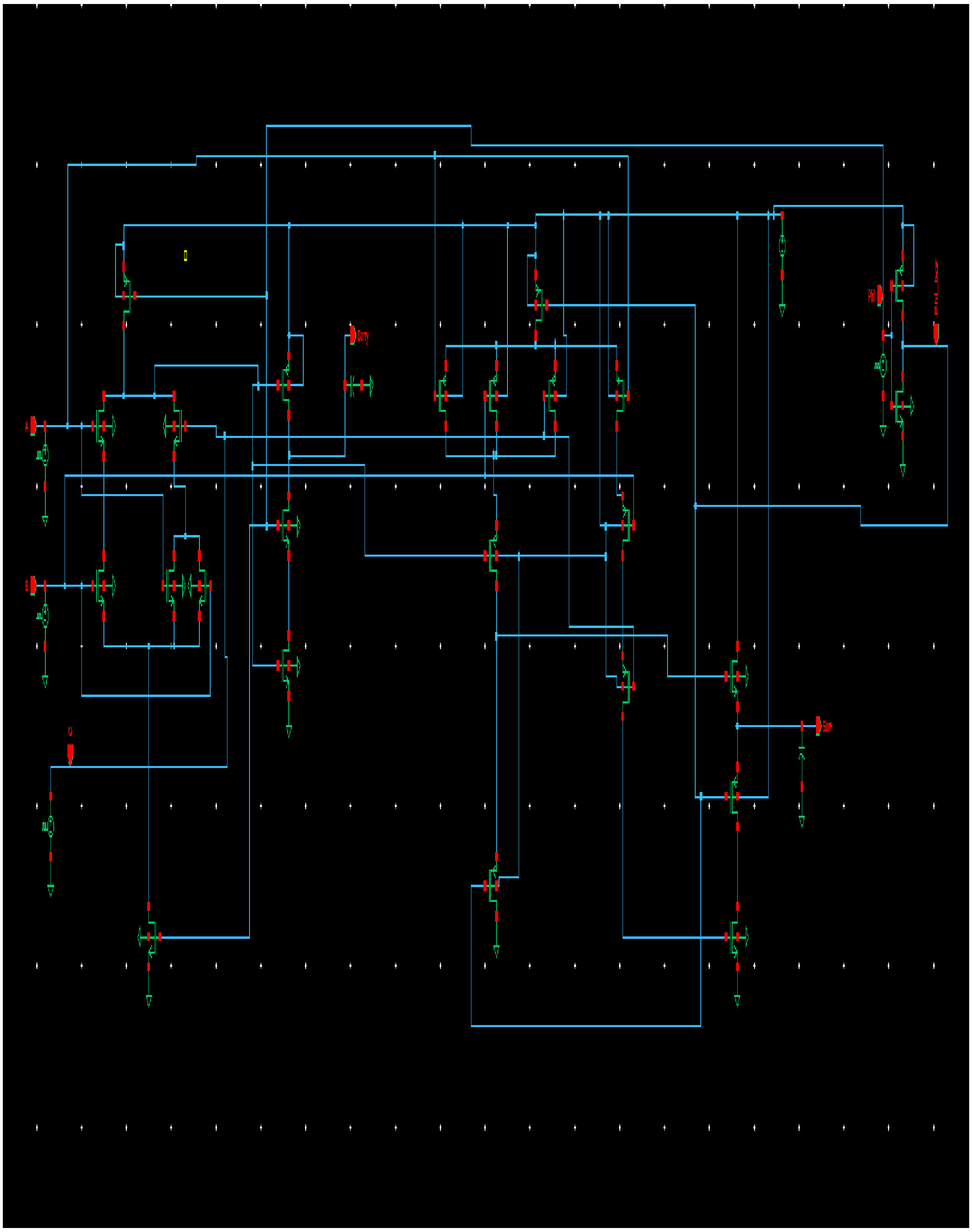


Fig.4.37 Schematic of 1-Bit full Adder using NORA (NO Race Around) Logic

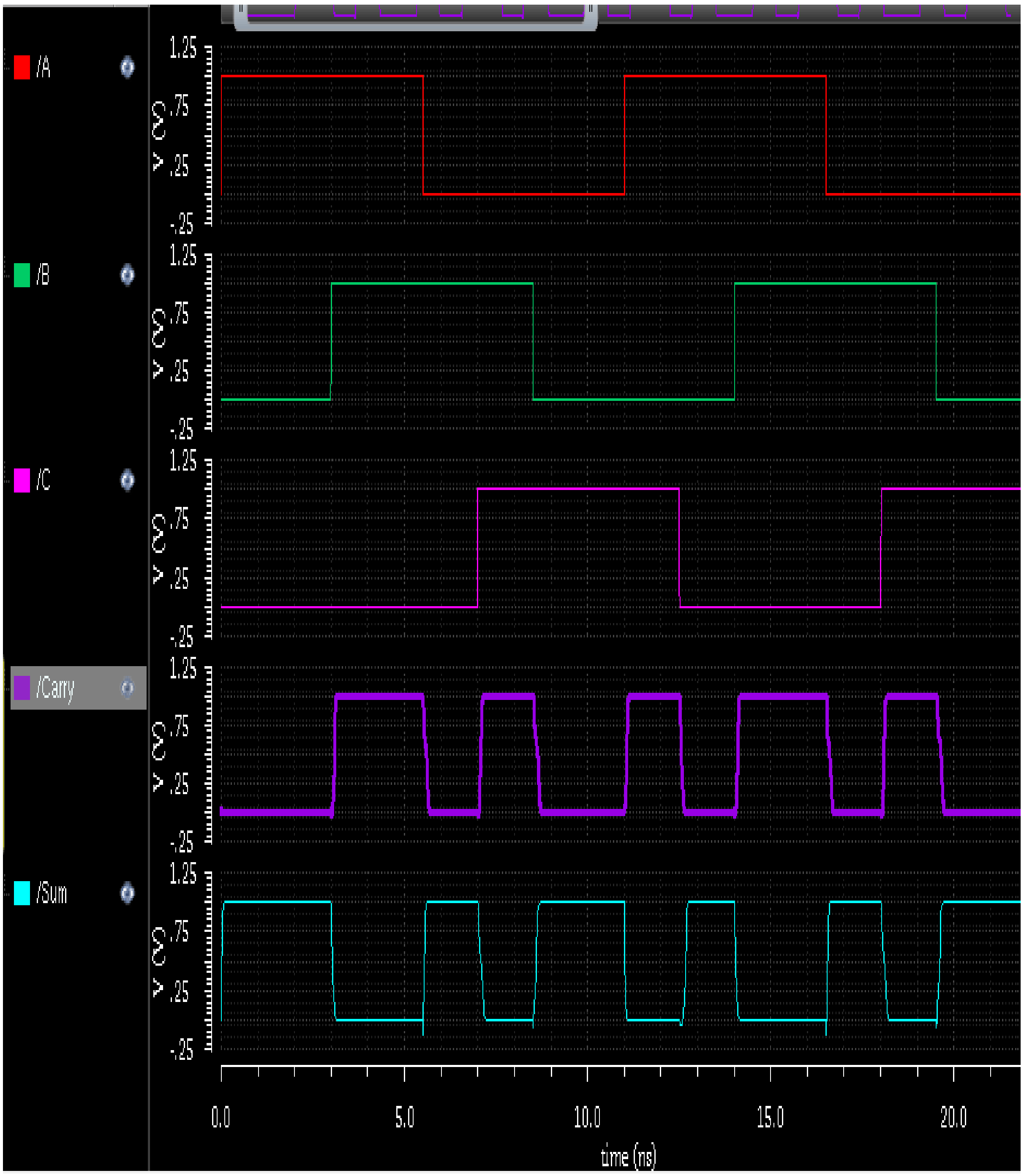


Fig.4.38 Inputs and Output of 1-Bit Full Adder using NORA logic

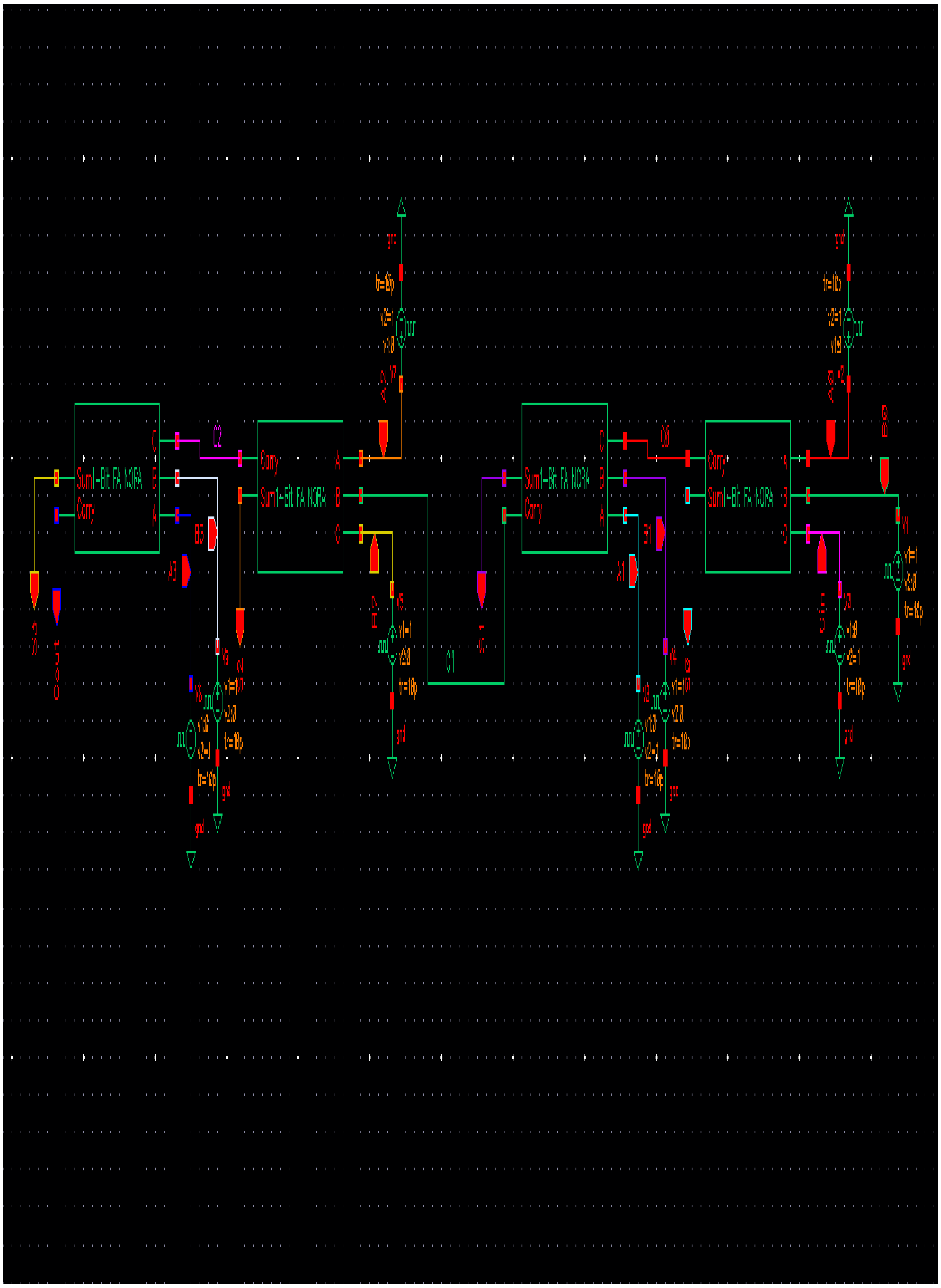


Fig.4.39 Schematic of 4-Bit full Adder using NORA (NO Race Around) Logic

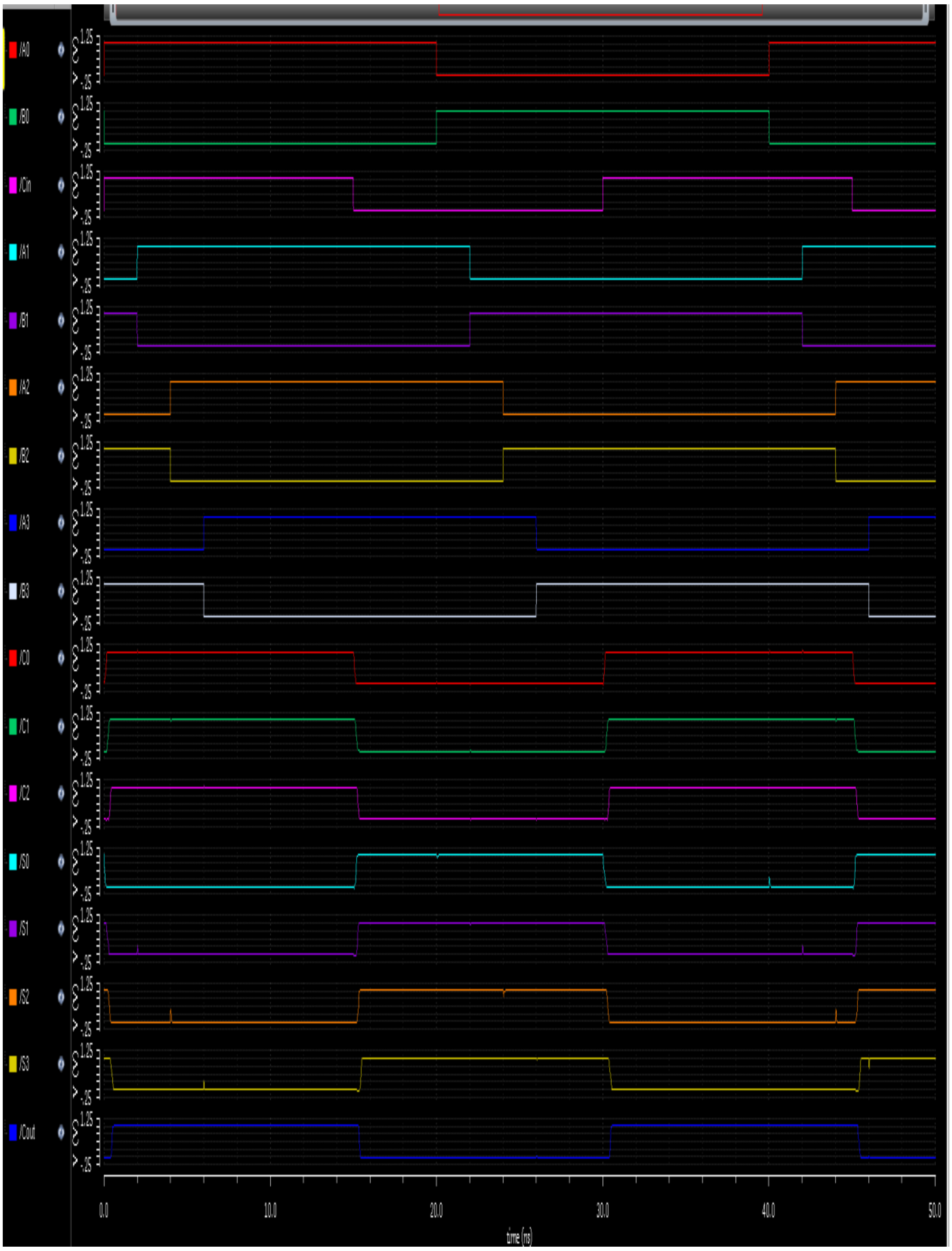


Fig.4.40 Inputs and Output of 4-Bit Full Adder using NORA logic

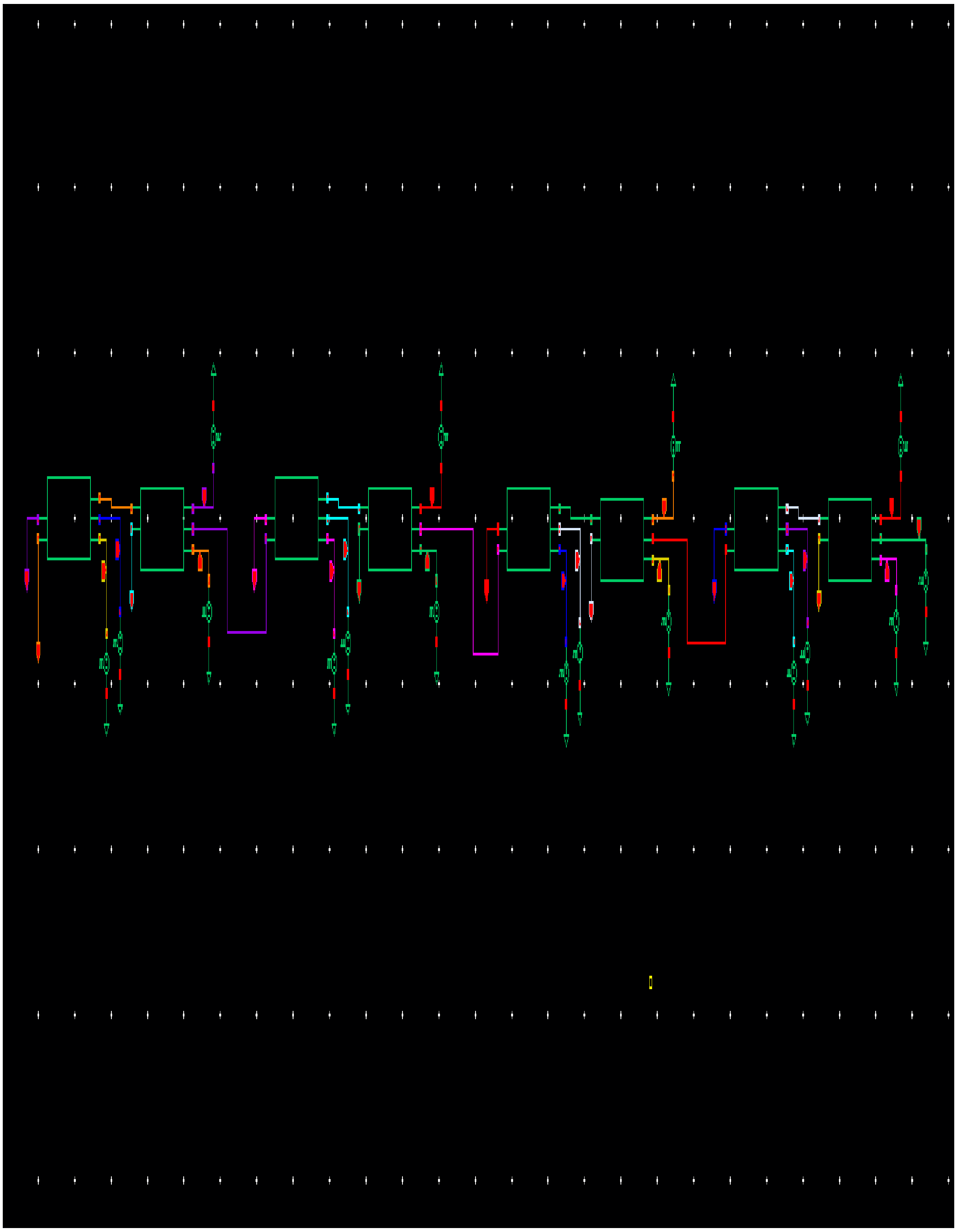


Fig.4.41 Schematic of 8-Bit full Adder using NORA (NO Race Around) Logic

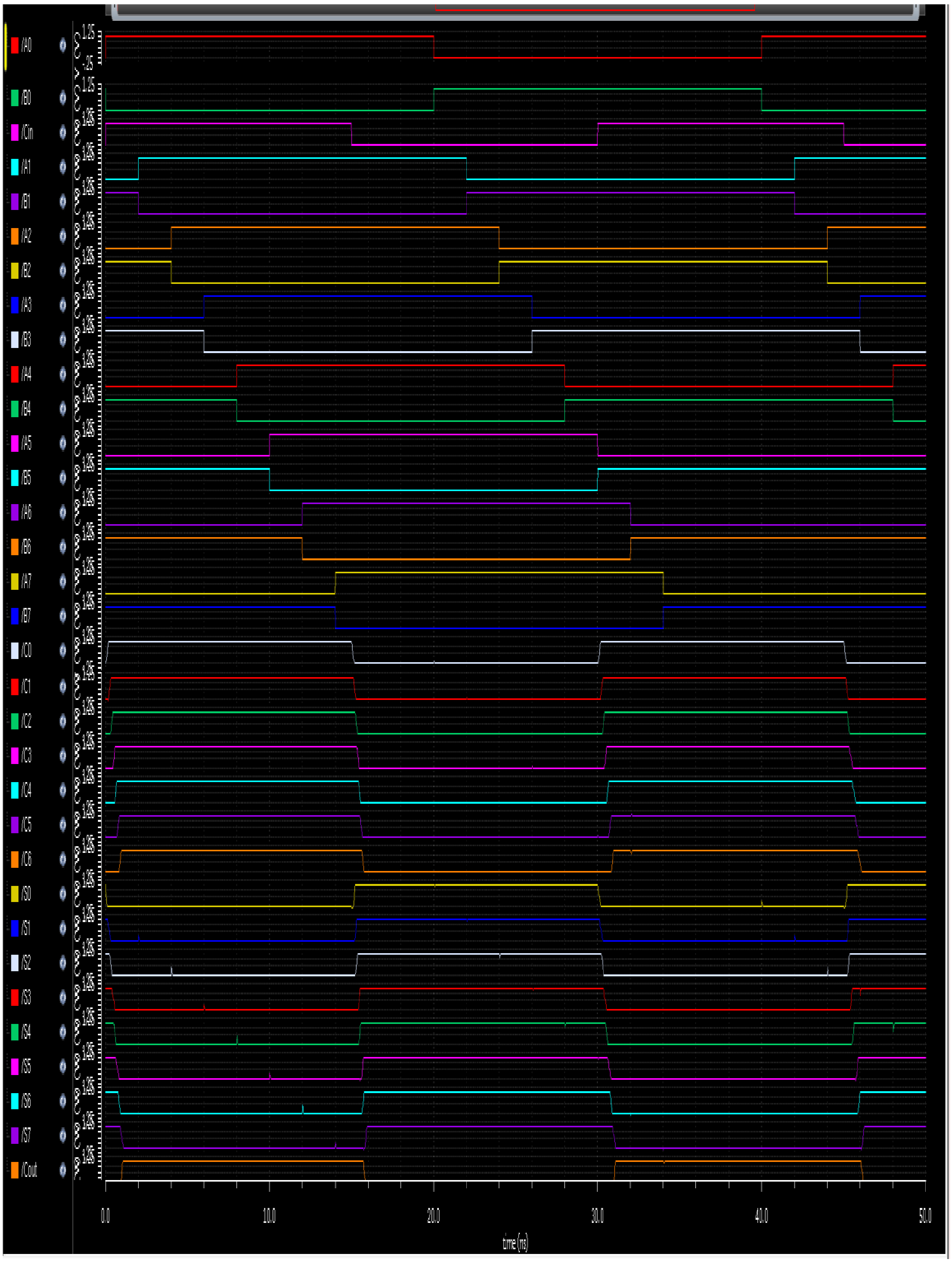


Fig.4.42 Inputs and Output of 8-Bit Full Adder using NORA logic

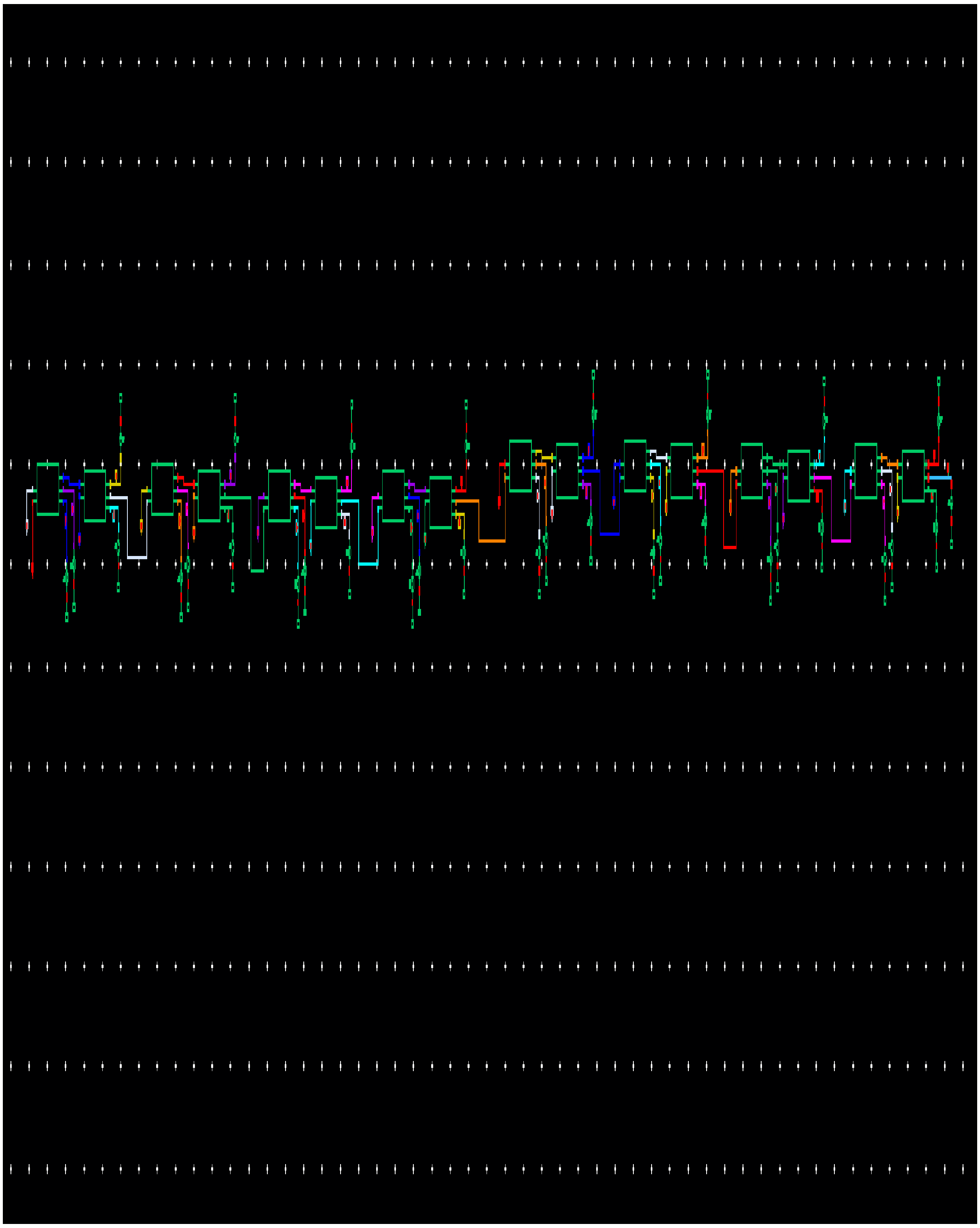


Fig.4.43 Schematic of 16-Bit full Adder using NORA (NO Race Around) Logic

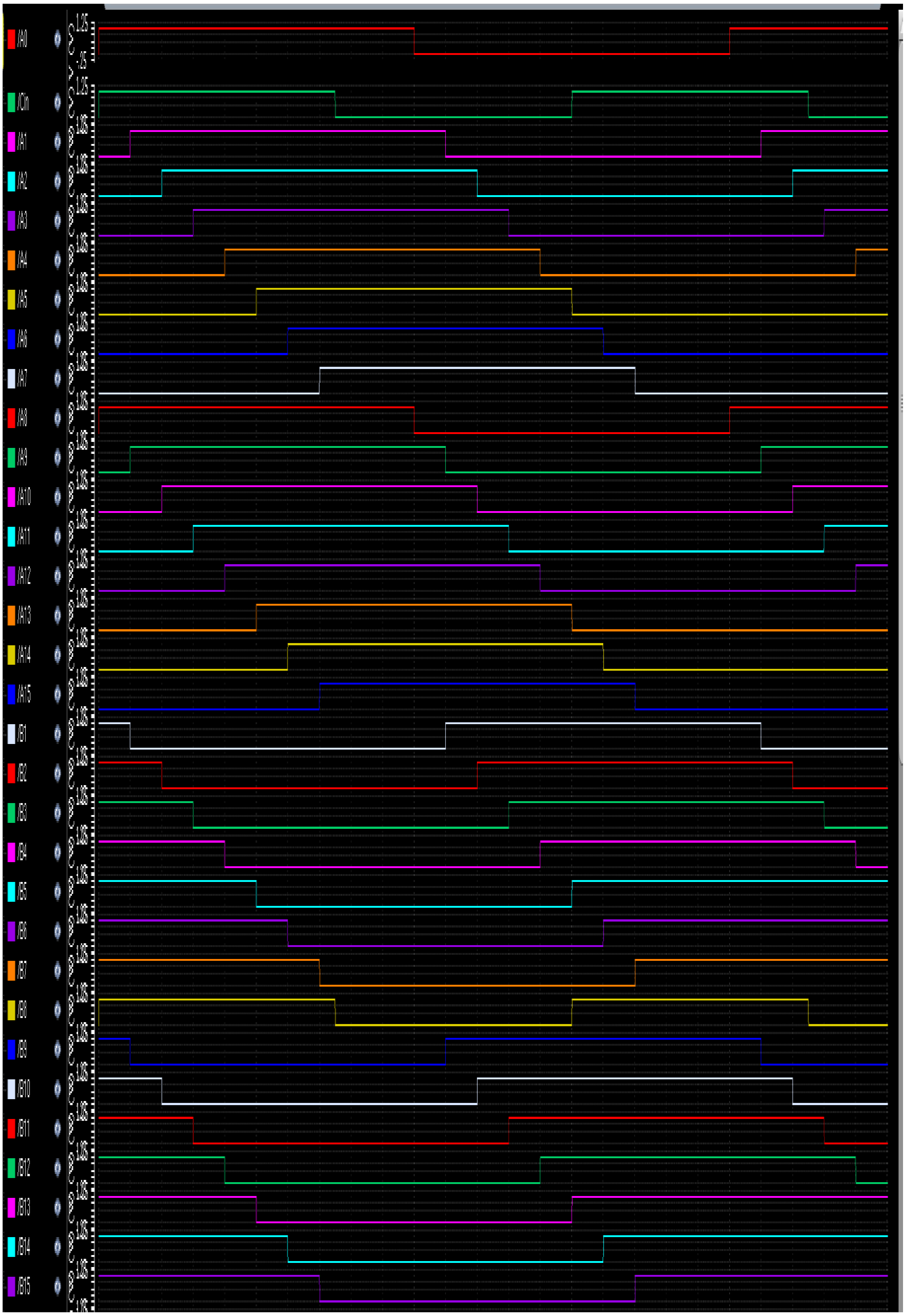


Fig.4.44 Inputs and Output of 16-Bit Full Adder using NORA logic

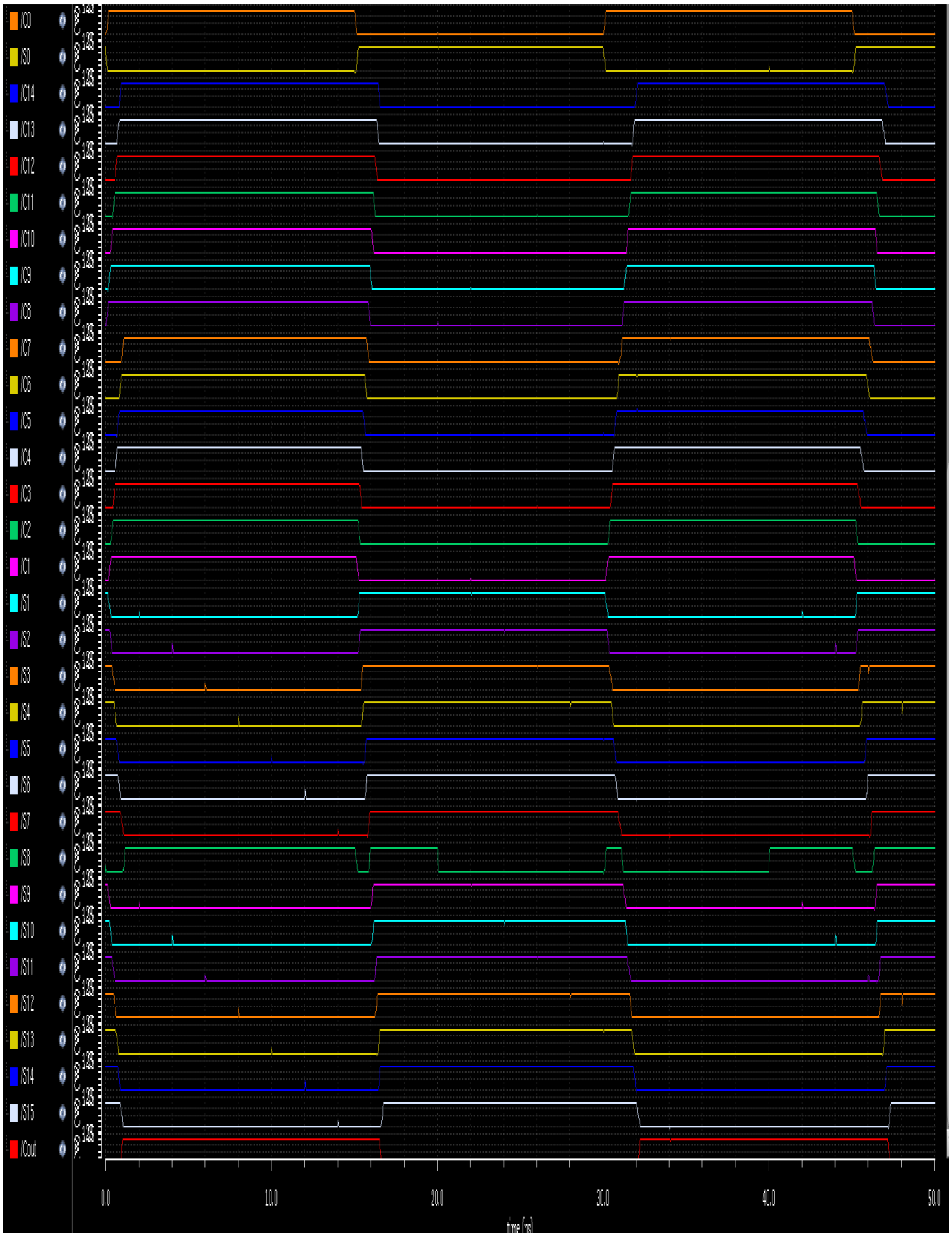
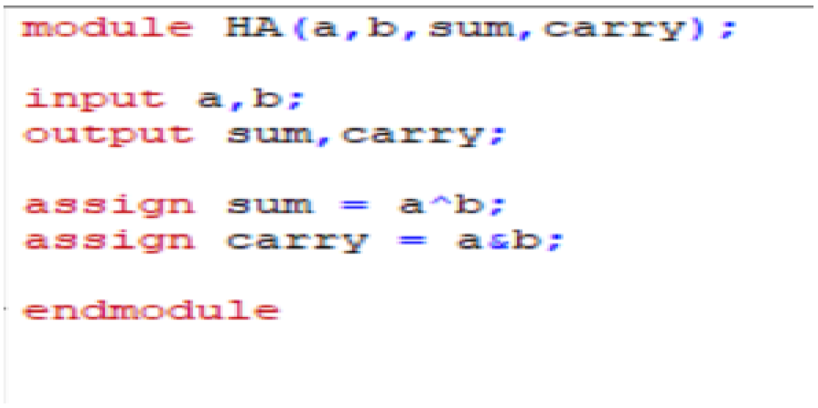


Fig.4.45 Inputs and Output of 16-Bit Full Adder using NORA logic

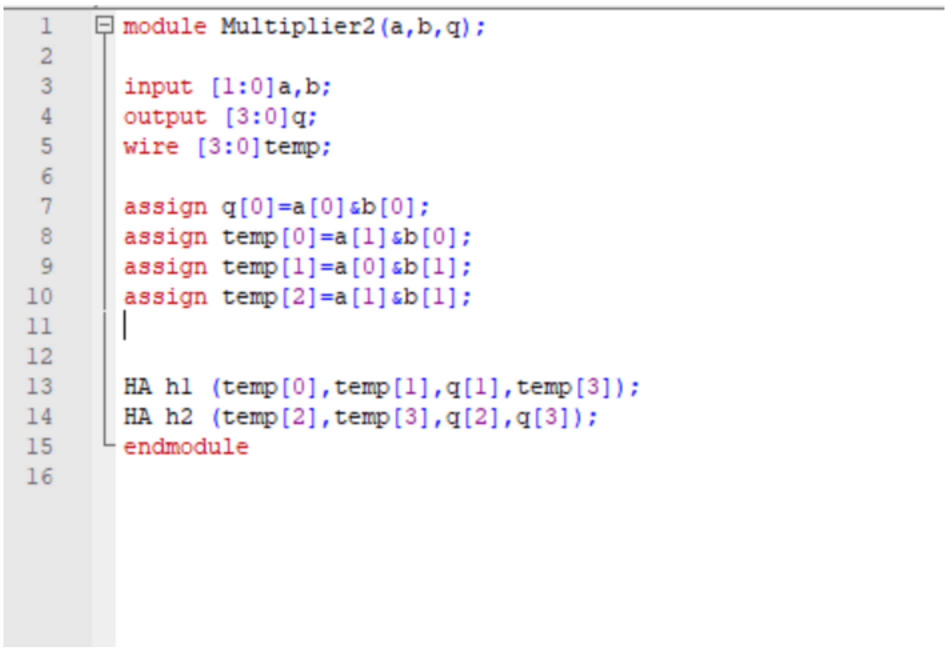
Table 1. Delays of 16bit- Ripple Carry Adder using various Logic Styles

S.No	Logic Style	Delay
1.	Static CMOS	78.69ps
2.	DCVSL	87.4ps
3.	NORA	53.68ps



```
module  HA(a,b,sum,carry);

input  a,b;
output  sum,carry;

assign  sum  =  a^b;
assign  carry  =  a&b;

endmodule
```

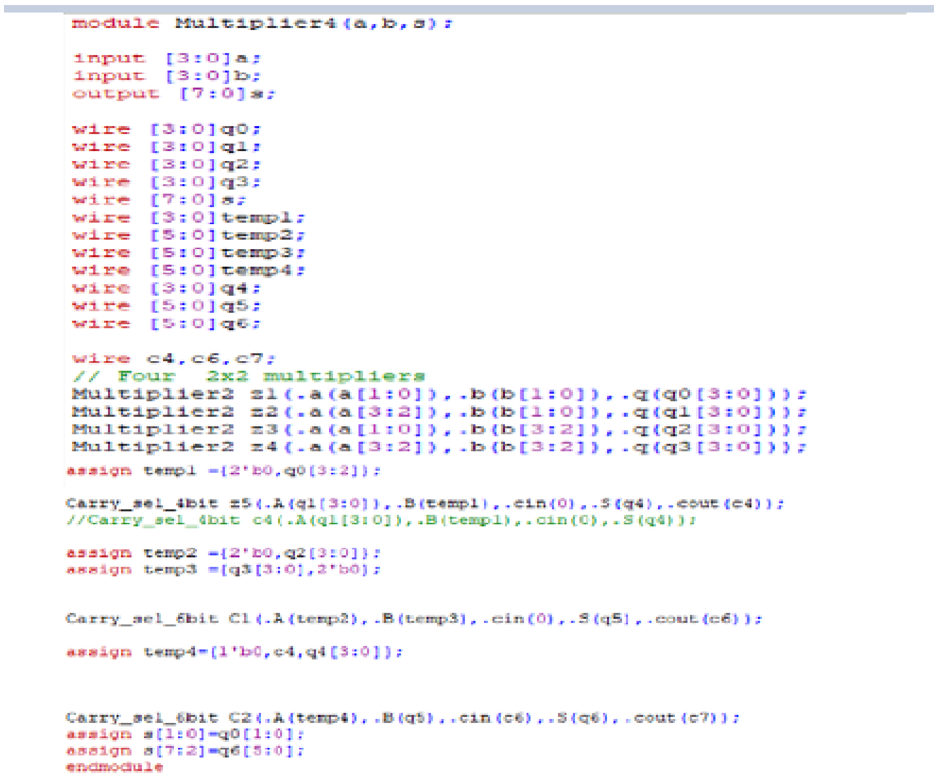
Fig.4.46 Half Adder



```
module Multiplier2(a,b,q);

input  [1:0]a,b;
output  [3:0]q;
wire  [3:0]temp;

assign q[0]=a[0]&b[0];
assign temp[0]=a[1]&b[0];
assign temp[1]=a[0]&b[1];
assign temp[2]=a[1]&b[1];


HA h1  (temp[0],temp[1],q[1],temp[3]);
HA h2  (temp[2],temp[3],q[2],q[3]);
endmodule

```

Fig.4.47 2-Bit Multiplier

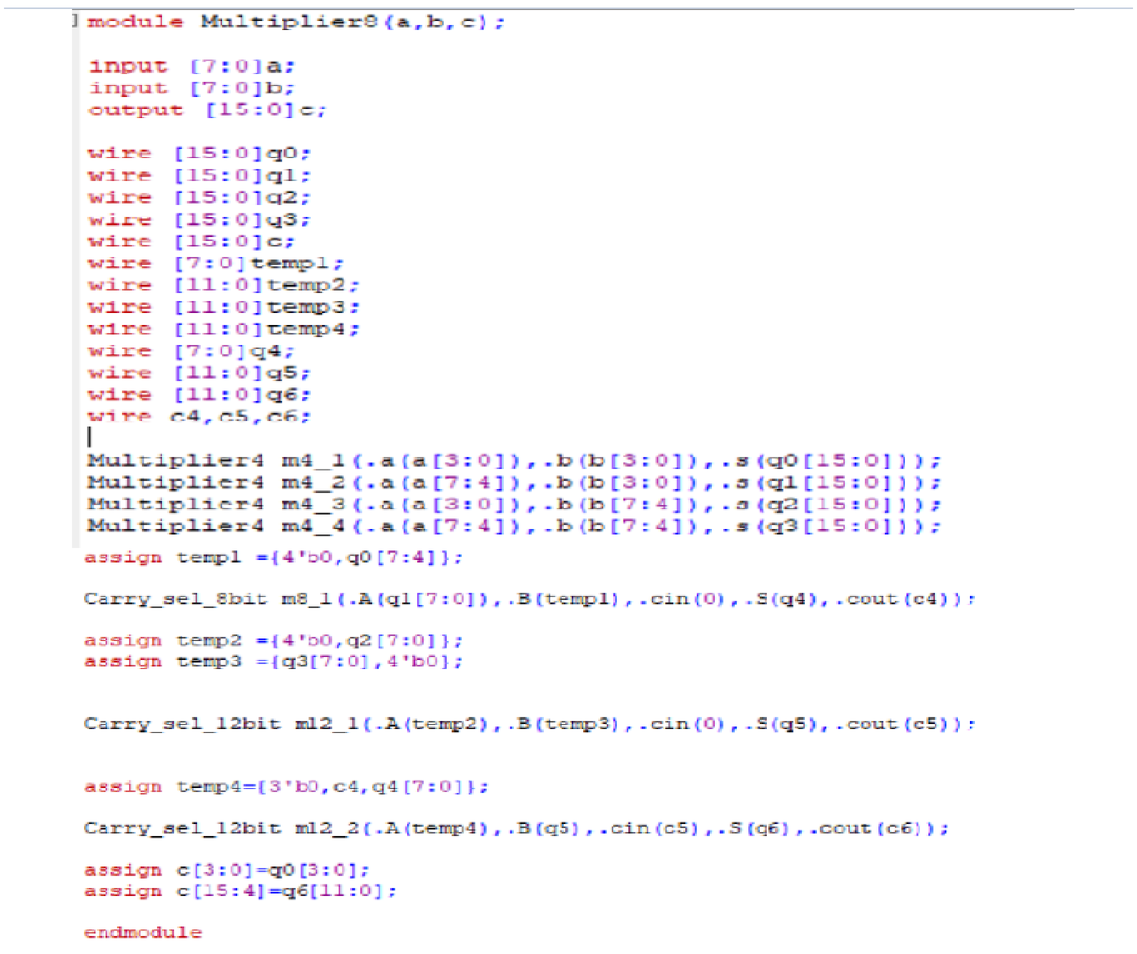
In the above code (Figure 4.47), the design function is to implement the 2*2 Bit Vedic Multiplier which have the inputs are A, B and Output as Q. To implement the 2-Bit Vedic Multiplier we require two half Adders as well which are directly initialized in this module and are have the code in the another module. In Multiplier when the inputs are of N-Bit and outputs should be of 2N capacity i.e why we declare output should have the capacity of four while the max capacity of input here is two. Here in the (Figure 4.1), we can see the simulation along with some test cases and in (Figure 4.2) we can see the Netlist generated for above 2-Bit Multiplier.



```
module Multiplier4(a,b,s);

input  [3:0]a;
input  [3:0]b;
output  [7:0]s;

wire  [3:0]q0;
wire  [3:0]q1;
wire  [3:0]q2;
wire  [3:0]q3;
wire  [7:0]s;
wire  [3:0]temp1;
wire  [5:0]temp2;
wire  [5:0]temp3;
wire  [5:0]temp4;
wire  [3:0]q4;
wire  [5:0]q5;
wire  [5:0]q6;

wire c4,c6,c7;
// Four  2x2 multipliers
Multiplier2 z1(.a(a[1:0]),.b(b[1:0]),.q(q0[3:0]));
Multiplier2 z2(.a(a[3:2]),.b(b[1:0]),.q(q1[3:0]));
Multiplier2 z3(.a(a[1:0]),.b(b[3:2]),.q(q2[3:0]));
Multiplier2 z4(.a(a[3:2]),.b(b[3:2]),.q(q3[3:0]));
assign temp1 ={2'b0,q0[3:2]};

Carry_sel_4bit z5(.A(q1[3:0]),.B(temp1),.cin(0),.S(q4),.cout(c4));
//Carry_sel_4bit c4(.A(q1[3:0]),.B(temp1),.cin(0),.S(q4));

assign temp2 ={2'b0,q2[3:0]};
assign temp3 ={q3[3:0],2'b0};


Carry_sel_6bit C1(.A(temp2),.B(temp3),.cin(0),.S(q5),.cout(c6));

assign temp4={1'b0,c4,q4[3:0]};


Carry_sel_6bit C2(.A(temp4),.B(q5),.cin(c6),.S(q6),.cout(c7));
assign s[1:0]=q0[1:0];
assign s[7:2]=q6[5:0];
endmodule
```

Fig.4.48 4-Bit Multiplier

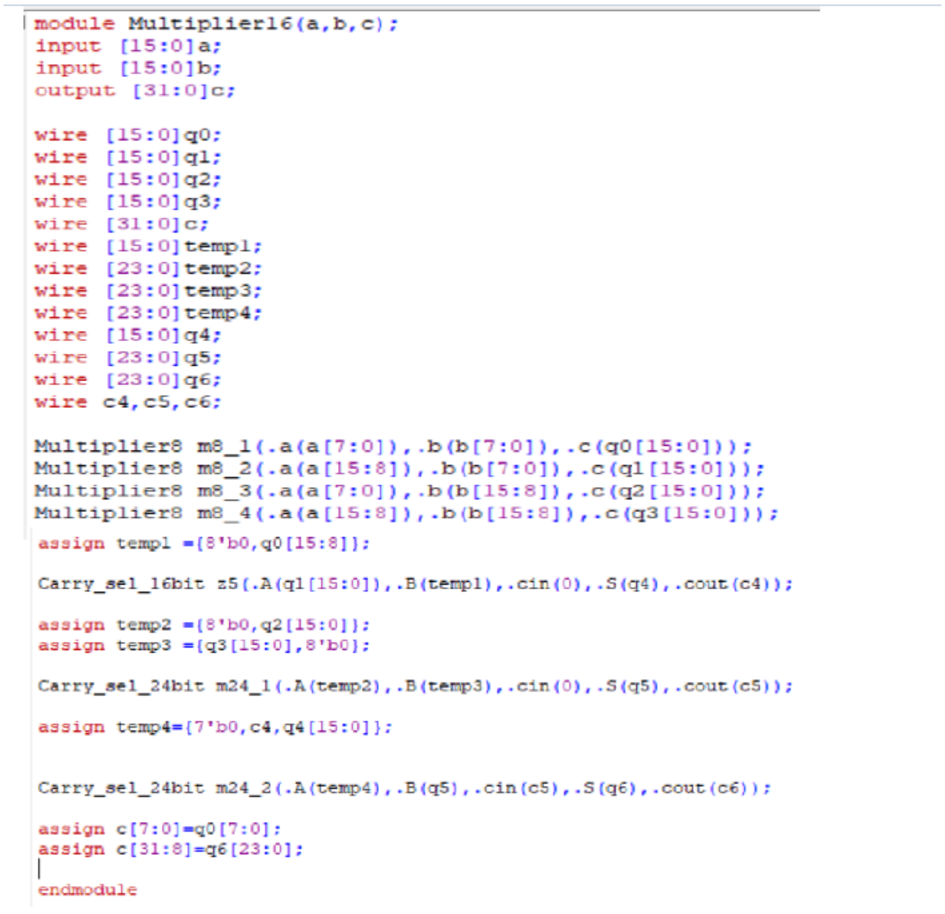
In the above code (Figure 4.48), the design function is to implement the 4*4 Bit Vedic Multiplier which have the inputs are A, B and Output as S. Along with inputs and outputs we require some internal wire which drives the signals we have taken the help of wire. To implement 4-Bit Vedic Multiplier we initiated four 2-Bit Vedic Multiplier which shown in the above fig and along with as every Multipliers we require adders here we used Carry Select Adder. As internally shifting of bits is required in Multipliers we used 4-Bit and 6-Bit Carry Select Adders. Here in the (Figure 4.3), we can see the simulation along with some test cases and in (Figure 4.4) we can see the Netlist generated for above 4-Bit Multiplier.



```
module Multiplier8(a,b,c);

input [7:0]a;
input [7:0]b;
output [15:0]c;

wire [15:0]q0;
wire [15:0]q1;
wire [15:0]q2;
wire [15:0]q3;
wire [15:0]c;
wire [7:0]temp1;
wire [11:0]temp2;
wire [11:0]temp3;
wire [11:0]temp4;
wire [7:0]q4;
wire [11:0]q5;
wire [11:0]q6;
wire c4,c5,c6;

Multiplier4 m4_1(.a(a[3:0]),.b(b[3:0]),.s(q0[15:0]));
Multiplier4 m4_2(.a(a[7:4]),.b(b[3:0]),.s(q1[15:0]));
Multiplier4 m4_3(.a(a[3:0]),.b(b[7:4]),.s(q2[15:0]));
Multiplier4 m4_4(.a(a[7:4]),.b(b[7:4]),.s(q3[15:0]));
assign temp1 ={4'b0,q0[7:4]};

Carry_sel_8bit m8_1(.A(q1[7:0]),.B(temp1),.cin(0),.S(q4),.cout(c4));

assign temp2 ={4'b0,q2[7:0]};
assign temp3 ={q3[7:0],4'b0};


Carry_sel_12bit m12_1(.A(temp2),.B(temp3),.cin(0),.S(q5),.cout(c5));


assign temp4={3'b0,c4,q4[7:0]};

Carry_sel_12bit m12_2(.A(temp4),.B(q5),.cin(c5),.S(q6),.cout(c6));

assign c[3:0]=q0[3:0];
assign c[15:4]=q6[11:0];

endmodule
```

Fig.4.49 8-Bit Multiplier

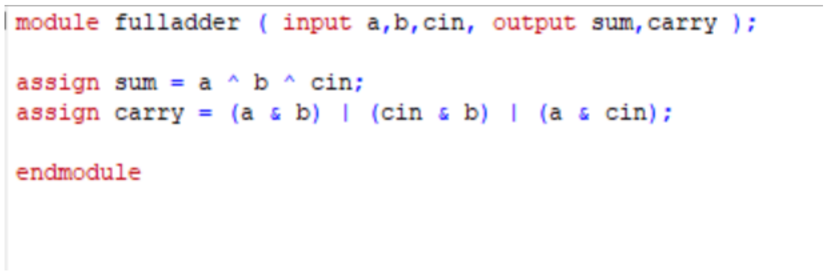
In the above code (Figure 4.49), the design function is to implement the 8*8 Bit Vedic Multiplier which have the inputs are A, B and Output as C. Along with inputs and outputs we require some internal wire which drives the signals we have taken the help of wire. Here as input is of 4-bit and in multipliers the output capacity would be double of maximum of input capacity i.e why we declared output as of 8-Bit. To implement 8-Bit Vedic Multiplier we initiated four 4-Bit Vedic Multiplier which shown in the above fig and along with as every Multipliers we require adders here we used Carry Select Adder. As internally shifting of bits is required in Multipliers we used 8-Bit and 12-Bit Carry Select Adders. Here in the (Figure 4.5), we can see the simulation along with some test cases and in (Figure 4.6) we can see the Netlist generated for above 8-Bit Multiplier.



```
module Multiplier16(a,b,c);
input  [15:0]a;
input  [15:0]b;
output  [31:0]c;

wire  [15:0]q0;
wire  [15:0]q1;
wire  [15:0]q2;
wire  [15:0]q3;
wire  [31:0]c;
wire  [15:0]temp1;
wire  [23:0]temp2;
wire  [23:0]temp3;
wire  [23:0]temp4;
wire  [15:0]q4;
wire  [23:0]q5;
wire  [23:0]q6;
wire  c4,c5,c6;

Multiplier8 m8_1(.a(a[7:0]),.b(b[7:0]),.c(q0[15:0]));
Multiplier8 m8_2(.a(a[15:8]),.b(b[7:0]),.c(q1[15:0]));
Multiplier8 m8_3(.a(a[7:0]),.b(b[15:8]),.c(q2[15:0]));
Multiplier8 m8_4(.a(a[15:8]),.b(b[15:8]),.c(q3[15:0]));
assign temp1 ={8'b0,q0[15:8]};

Carry_sel_16bit z5(.A(q1[15:0]),.B(temp1),.cin(0),.S(q4),.cout(c4));

assign temp2 ={8'b0,q2[15:0]};
assign temp3 ={q3[15:0],8'b0};

Carry_sel_24bit m24_1(.A(temp2),.B(temp3),.cin(0),.S(q5),.cout(c5));

assign temp4={7'b0,c4,q4[15:0]};


Carry_sel_24bit m24_2(.A(temp4),.B(q5),.cin(c5),.S(q6),.cout(c6));

assign c[7:0]=q0[7:0];
assign c[31:8]=q6[23:0];

endmodule
```

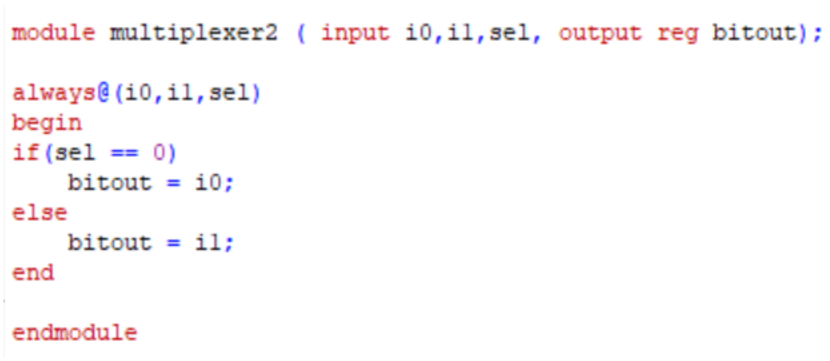
Fig.4.50 16-Bit Multiplier

In the above code (Figure 4.50), the design function is to implement the 16*16 Bit Vedic Multiplier which have the inputs are A, B and Output as C. Along with inputs and outputs we require some internal wire which drives the signals we have taken the help of wire. Here as input is of 16-bit and in multipliers the output capacity would be double of maximum of input capacity i.e why we declared output as of 32-Bit. To implement 8-Bit Vedic Multiplier we initiated four 8-Bit Vedic Multiplier which shown in the above fig and along with as every Multipliers we require adders here we used Carry Select Adder. As internally shifting of bits is required in Multipliers we used 16-Bit and 24-Bit Carry Select Adders. Here in the (Figure 4.7), we can see the simulation along with some test cases and in (Figure 4.8) we can see the Netlist generated for above 16-Bit Multiplier.



```
module fulladder ( input a,b,cin, output sum,carry );

assign sum = a ^ b ^ cin;
assign carry = (a & b) | (cin & b) | (a & cin);

endmodule
```

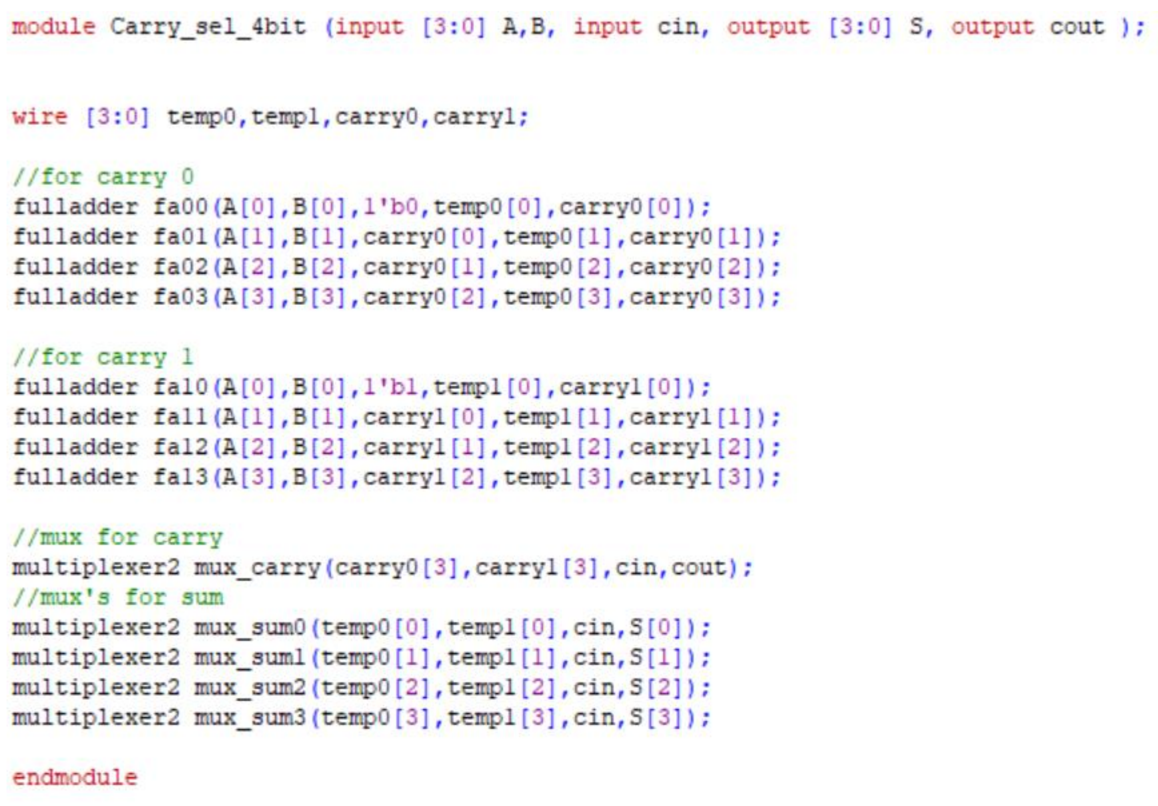
Fig.4.51 Full Adder



```
module multiplexer2 ( input i0,i1,sel, output reg bitout);

always@(i0,i1,sel)
begin
if(sel == 0)
    bitout = i0;
else
    bitout = i1;
end

endmodule
```

Fig.4.52 2*1 Multiplexer

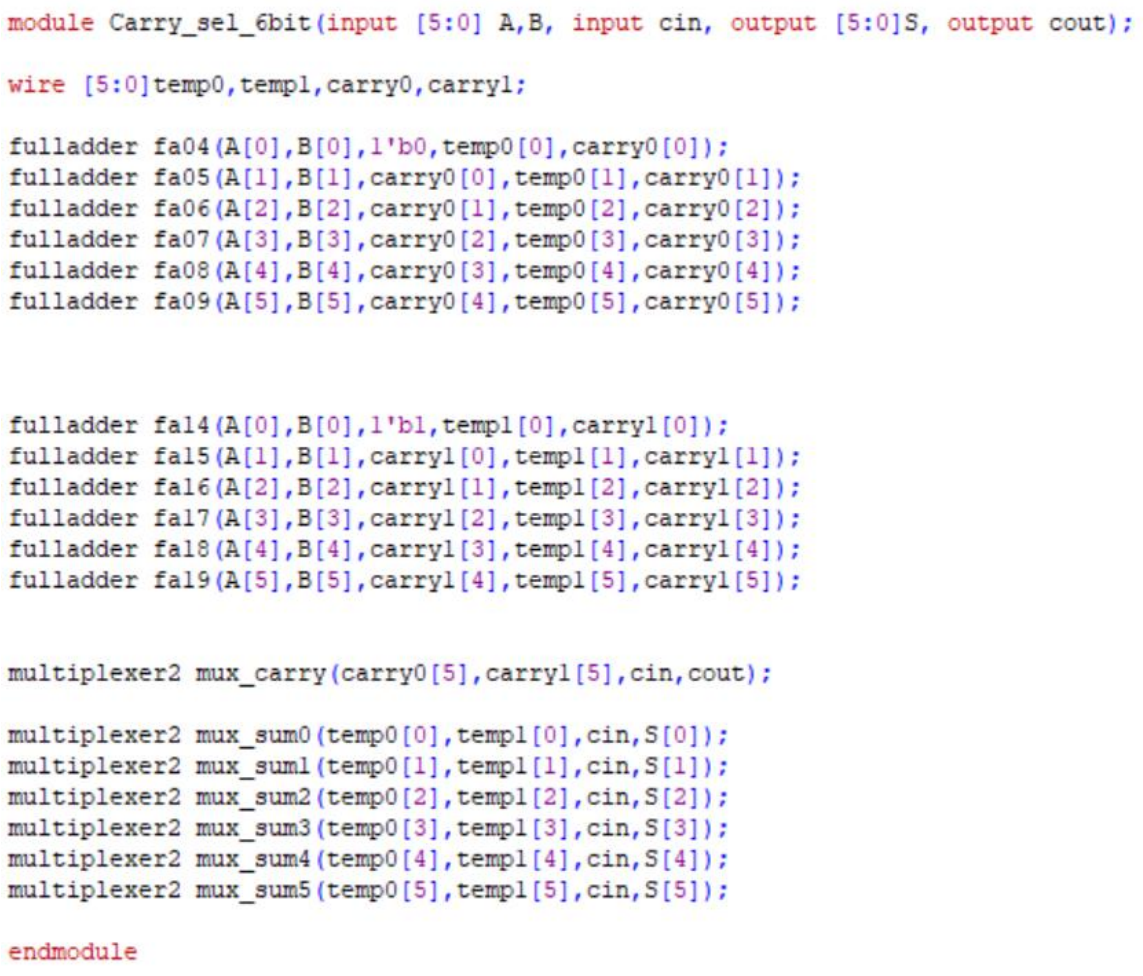
The above two modules which are shown in the Figure(4.51) Full Adder and Figure (4.52) 2*1 Multiplexer are used inside the modules of Carry select Adder.



```
module Carry_sel_4bit (input [3:0] A,B, input cin, output [3:0] S, output cout );


wire [3:0] temp0,temp1,carry0,carry1;

//for carry 0
fulladder fa00(A[0],B[0],1'b0,temp0[0],carry0[0]);
fulladder fa01(A[1],B[1],carry0[0],temp0[1],carry0[1]);
fulladder fa02(A[2],B[2],carry0[1],temp0[2],carry0[2]);
fulladder fa03(A[3],B[3],carry0[2],temp0[3],carry0[3]);

//for carry 1
fulladder fa10(A[0],B[0],1'b1,temp1[0],carry1[0]);
fulladder fa11(A[1],B[1],carry1[0],temp1[1],carry1[1]);
fulladder fa12(A[2],B[2],carry1[1],temp1[2],carry1[2]);
fulladder fa13(A[3],B[3],carry1[2],temp1[3],carry1[3]);

//mux for carry
multiplexer2 mux_carry(carry0[3],carry1[3],cin,cout);
//mux's for sum
multiplexer2 mux_sum0(temp0[0],temp1[0],cin,S[0]);
multiplexer2 mux_sum1(temp0[1],temp1[1],cin,S[1]);
multiplexer2 mux_sum2(temp0[2],temp1[2],cin,S[2]);
multiplexer2 mux_sum3(temp0[3],temp1[3],cin,S[3]);

endmodule
```

Fig.4.53 4-Bit Carry Select Adder

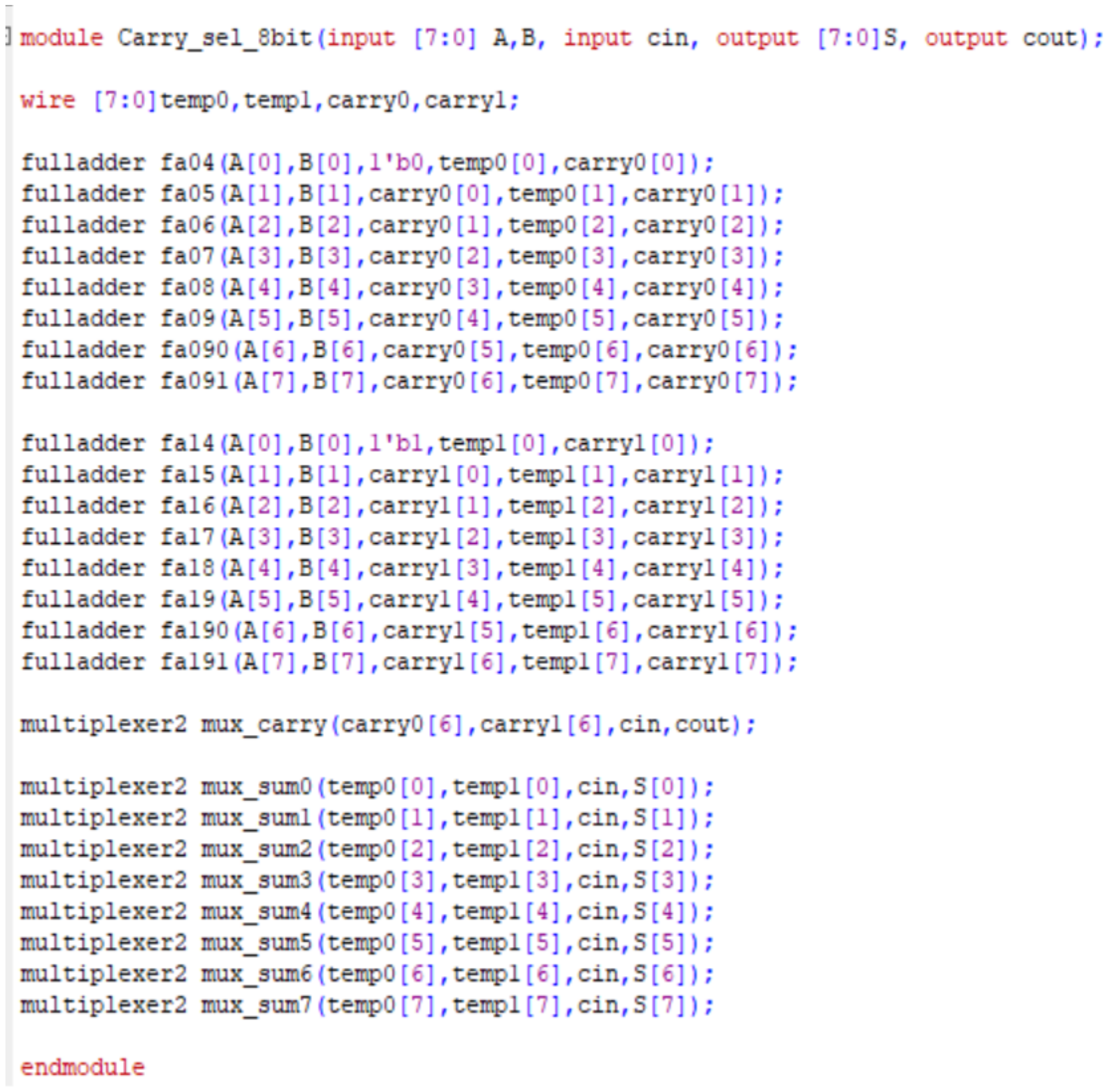
In the above code (Figure 4.53), the design function is to implement the 4-Bit Carry Select Adder which have the inputs are A, B and Cin and Outputs are Sum is denoted by S and Carryout as cout. Along with inputs and outputs we require some internal wire which drives the signals we have taken the help of wire. As in this method we use total of Eight Full Adders in which Four Adders are used when cin=0 and other four Adders when cin=1, and calculations are done parallelly and finally we need multilplexers to choose final Sum by choosing appropriate select lines. Here total five Multiplexers are used in which four are being involved in sum and the remaining one mux is used to choose the carryout.



```
module Carry_sel_6bit(input [5:0] A,B, input cin, output [5:0]S, output cout);

wire [5:0]temp0,temp1,carry0,carry1;

fulladder fa04(A[0],B[0],1'b0,temp0[0],carry0[0]);
fulladder fa05(A[1],B[1],carry0[0],temp0[1],carry0[1]);
fulladder fa06(A[2],B[2],carry0[1],temp0[2],carry0[2]);
fulladder fa07(A[3],B[3],carry0[2],temp0[3],carry0[3]);
fulladder fa08(A[4],B[4],carry0[3],temp0[4],carry0[4]);
fulladder fa09(A[5],B[5],carry0[4],temp0[5],carry0[5]);



fulladder fa14(A[0],B[0],1'b1,temp1[0],carry1[0]);
fulladder fa15(A[1],B[1],carry1[0],temp1[1],carry1[1]);
fulladder fa16(A[2],B[2],carry1[1],temp1[2],carry1[2]);
fulladder fa17(A[3],B[3],carry1[2],temp1[3],carry1[3]);
fulladder fa18(A[4],B[4],carry1[3],temp1[4],carry1[4]);
fulladder fa19(A[5],B[5],carry1[4],temp1[5],carry1[5]);


multiplexer2 mux_carry(carry0[5],carry1[5],cin,cout);

multiplexer2 mux_sum0(temp0[0],temp1[0],cin,S[0]);
multiplexer2 mux_sum1(temp0[1],temp1[1],cin,S[1]);
multiplexer2 mux_sum2(temp0[2],temp1[2],cin,S[2]);
multiplexer2 mux_sum3(temp0[3],temp1[3],cin,S[3]);
multiplexer2 mux_sum4(temp0[4],temp1[4],cin,S[4]);
multiplexer2 mux_sum5(temp0[5],temp1[5],cin,S[5]);

endmodule
```

Fig 4.54 6-Bit Carry Select Adder

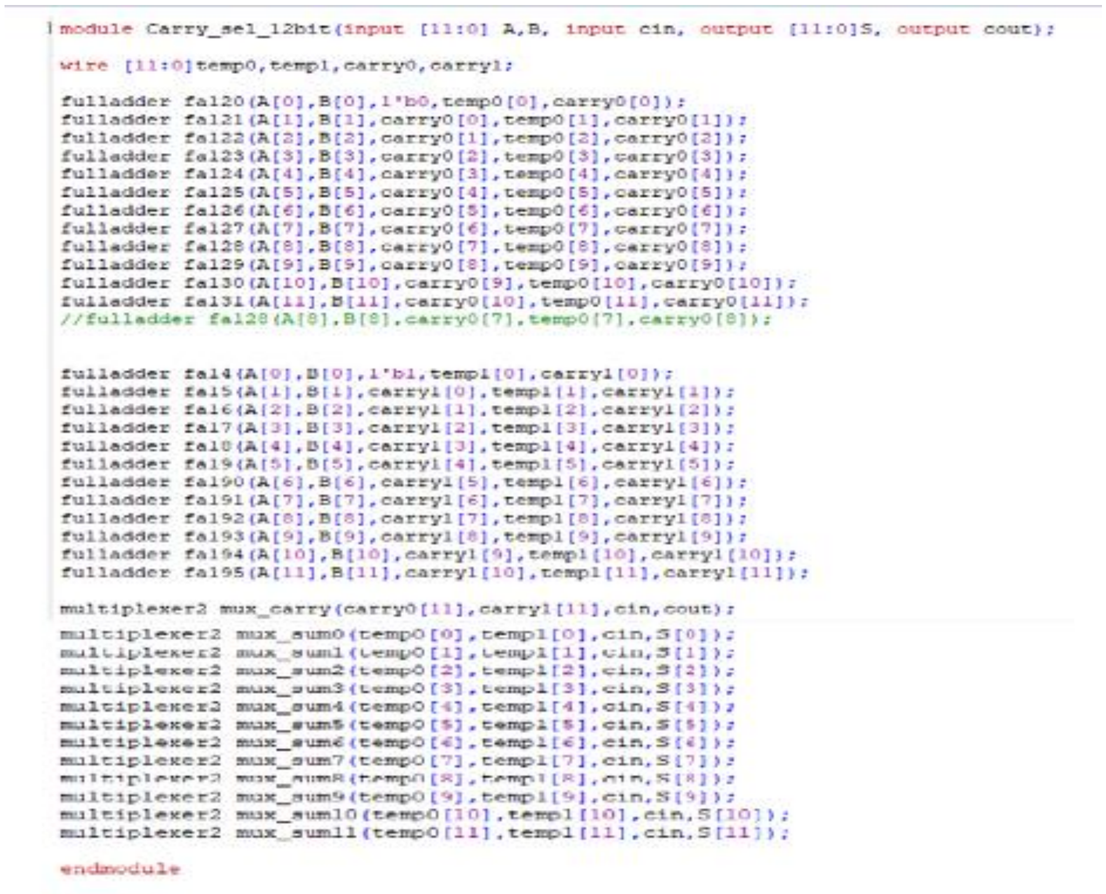
In the above code (Figure 4.54), the design function is to implement the 6-Bit Carry Select Adder which have the inputs are A, B and Cin and Outputs are Sum is denoted by S and Carryout as cout. Along with inputs and outputs we require some internal wire which drives the signals we have taken the help of wire. As in this method we use total of twelve Full Adders in which Six Adders are used when cin=0 and other Six Adders when cin=1, and calculations are done parallelly and finally we need multilplexers to choose final Sum by choosing appropriate select lines. Here total Seven Multiplexers are used in which Six are being involved in sum and the remaining one mux is used to choose the carryout.



```
module Carry_sel_8bit(input [7:0] A,B, input cin, output [7:0]S, output cout);

wire [7:0]temp0,templ,carry0,carryl;

fulladder fa04(A[0],B[0],1'b0,temp0[0],carry0[0]);
fulladder fa05(A[1],B[1],carry0[0],temp0[1],carry0[1]);
fulladder fa06(A[2],B[2],carry0[1],temp0[2],carry0[2]);
fulladder fa07(A[3],B[3],carry0[2],temp0[3],carry0[3]);
fulladder fa08(A[4],B[4],carry0[3],temp0[4],carry0[4]);
fulladder fa09(A[5],B[5],carry0[4],temp0[5],carry0[5]);
fulladder fa090(A[6],B[6],carry0[5],temp0[6],carry0[6]);
fulladder fa091(A[7],B[7],carry0[6],temp0[7],carry0[7]);

fulladder fa14(A[0],B[0],1'bl,templ[0],carryl[0]);
fulladder fa15(A[1],B[1],carryl[0],templ[1],carryl[1]);
fulladder fa16(A[2],B[2],carryl[1],templ[2],carryl[2]);
fulladder fa17(A[3],B[3],carryl[2],templ[3],carryl[3]);
fulladder fa18(A[4],B[4],carryl[3],templ[4],carryl[4]);
fulladder fa19(A[5],B[5],carryl[4],templ[5],carryl[5]);
fulladder fa190(A[6],B[6],carryl[5],templ[6],carryl[6]);
fulladder fa191(A[7],B[7],carryl[6],templ[7],carryl[7]);

multiplexer2 mux_carry(carry0[6],carryl[6],cin,cout);

multiplexer2 mux_sum0(temp0[0],templ[0],cin,S[0]);
multiplexer2 mux_suml(temp0[1],templ[1],cin,S[1]);
multiplexer2 mux_sum2(temp0[2],templ[2],cin,S[2]);
multiplexer2 mux_sum3(temp0[3],templ[3],cin,S[3]);
multiplexer2 mux_sum4(temp0[4],templ[4],cin,S[4]);
multiplexer2 mux_sum5(temp0[5],templ[5],cin,S[5]);
multiplexer2 mux_sum6(temp0[6],templ[6],cin,S[6]);
multiplexer2 mux_sum7(temp0[7],templ[7],cin,S[7]);

endmodule
```

Fig 4.55 8-Bit Carry Select Adder

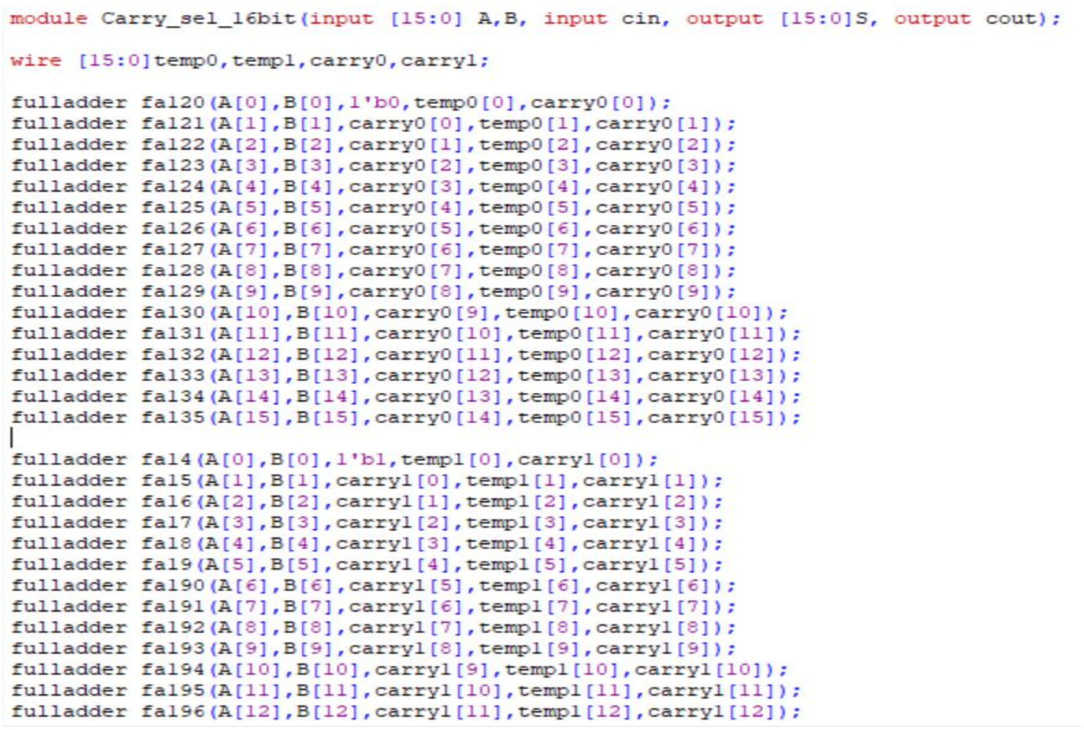
In the above code (Figure 4.55), the design function is to implement the 8-Bit Carry Select Adder which have the inputs are A, B and Cin and Outputs are Sum is denoted by S and Carryout as cout. Along with inputs and outputs we require some internal wire which drives the signals we have taken the help of wire. As in this method we use total of Sixteen Full Adders in which eight Adders are used when cin=0 and other eight Adders when cin=1, and calculations are done parallelly and finally we need multilplexers to choose final Sum by choosing appropriate select lines. Here total Nine Multiplexers are used in which Eight are being involved in sum and the remaining one mux is used to choose the carryout.



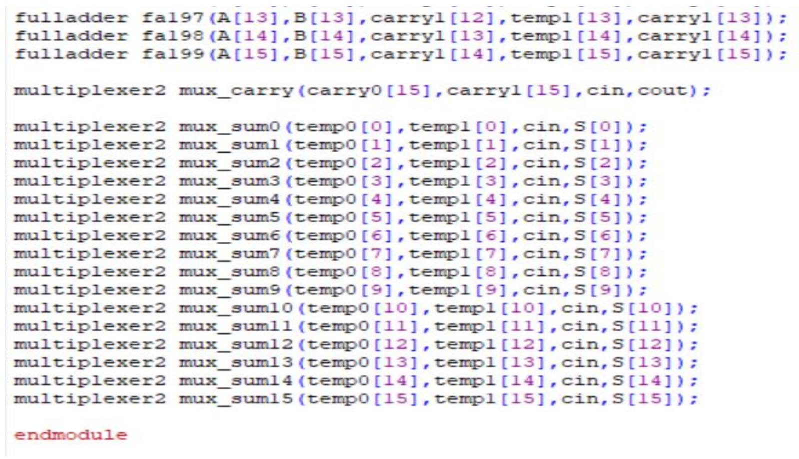
```
module Carry_sel_12bit(input [11:0] A,B, input cin, output [11:0]S, output cout);

wire [11:0]temp0,temp1,carry0,carry1;

fulladder fa120(A[0],B[0],1'b0,temp0[0],carry0[0]);
fulladder fa121(A[1],B[1],carry0[0],temp0[1],carry0[1]);
fulladder fa122(A[2],B[2],carry0[1],temp0[2],carry0[2]);
fulladder fa123(A[3],B[3],carry0[2],temp0[3],carry0[3]);
fulladder fa124(A[4],B[4],carry0[3],temp0[4],carry0[4]);
fulladder fa125(A[5],B[5],carry0[4],temp0[5],carry0[5]);
fulladder fa126(A[6],B[6],carry0[5],temp0[6],carry0[6]);
fulladder fa127(A[7],B[7],carry0[6],temp0[7],carry0[7]);
fulladder fa128(A[8],B[8],carry0[7],temp0[8],carry0[8]);
fulladder fa129(A[9],B[9],carry0[8],temp0[9],carry0[9]);
fulladder fa130(A[10],B[10],carry0[9],temp0[10],carry0[10]);
fulladder fa131(A[11],B[11],carry0[10],temp0[11],carry0[11]);
//fulladder fa128(A[8],B[8],carry0[7],temp0[7],carry0[8]);


fulladder fa14(A[0],B[0],1'b1,temp1[0],carry1[0]);
fulladder fa15(A[1],B[1],carry1[0],temp1[1],carry1[1]);
fulladder fa16(A[2],B[2],carry1[1],temp1[2],carry1[2]);
fulladder fa17(A[3],B[3],carry1[2],temp1[3],carry1[3]);
fulladder fa18(A[4],B[4],carry1[3],temp1[4],carry1[4]);
fulladder fa19(A[5],B[5],carry1[4],temp1[5],carry1[5]);
fulladder fa190(A[6],B[6],carry1[5],temp1[6],carry1[6]);
fulladder fa191(A[7],B[7],carry1[6],temp1[7],carry1[7]);
fulladder fa192(A[8],B[8],carry1[7],temp1[8],carry1[8]);
fulladder fa193(A[9],B[9],carry1[8],temp1[9],carry1[9]);
fulladder fa194(A[10],B[10],carry1[9],temp1[10],carry1[10]);
fulladder fa195(A[11],B[11],carry1[10],temp1[11],carry1[11]);

multiplexer2 mux_carry(carry0[11],carry1[11],cin,cout);
multiplexer2 mux_sum0(temp0[0],temp1[0],cin,S[0]);
multiplexer2 mux_sum1(temp0[1],temp1[1],cin,S[1]);
multiplexer2 mux_sum2(temp0[2],temp1[2],cin,S[2]);
multiplexer2 mux_sum3(temp0[3],temp1[3],cin,S[3]);
multiplexer2 mux_sum4(temp0[4],temp1[4],cin,S[4]);
multiplexer2 mux_sum5(temp0[5],temp1[5],cin,S[5]);
multiplexer2 mux_sum6(temp0[6],temp1[6],cin,S[6]);
multiplexer2 mux_sum7(temp0[7],temp1[7],cin,S[7]);
multiplexer2 mux_sum8(temp0[8],temp1[8],cin,S[8]);
multiplexer2 mux_sum9(temp0[9],temp1[9],cin,S[9]);
multiplexer2 mux_sum10(temp0[10],temp1[10],cin,S[10]);
multiplexer2 mux_sum11(temp0[11],temp1[11],cin,S[11]);

endmodule
```

Fig 4.56 12-Bit Carry Select Adder

In the above code (Figure 4.56), the design function is to implement the 12-Bit Carry Select Adder which have the inputs are A, B and Cin and Outputs are Sum is denoted by S and Carryout as cout. Along with inputs and outputs we require some internal wire which drives the signals we have taken the help of wire. As in this method we use total of Twenty Four Full Adders in which twelve Adders are used when cin=0 and other twelve Adders when cin=1, and calculations are done parallelly and finally we need multilplexers to choose final Sum by choosing appropriate select lines. Here total Thirteen Multiplexers are used in which Twelve are being involved in sum and the remaining one mux is used to choose the carryout.



```
module Carry_sel_16bit(input [15:0] A,B, input cin, output [15:0]S, output cout);

wire [15:0]temp0,temp1,carry0,carry1;

fulladder fa120(A[0],B[0],1'b0,temp0[0],carry0[0]);
fulladder fa121(A[1],B[1],carry0[0],temp0[1],carry0[1]);
fulladder fa122(A[2],B[2],carry0[1],temp0[2],carry0[2]);
fulladder fa123(A[3],B[3],carry0[2],temp0[3],carry0[3]);
fulladder fa124(A[4],B[4],carry0[3],temp0[4],carry0[4]);
fulladder fa125(A[5],B[5],carry0[4],temp0[5],carry0[5]);
fulladder fa126(A[6],B[6],carry0[5],temp0[6],carry0[6]);
fulladder fa127(A[7],B[7],carry0[6],temp0[7],carry0[7]);
fulladder fa128(A[8],B[8],carry0[7],temp0[8],carry0[8]);
fulladder fa129(A[9],B[9],carry0[8],temp0[9],carry0[9]);
fulladder fa130(A[10],B[10],carry0[9],temp0[10],carry0[10]);
fulladder fa131(A[11],B[11],carry0[10],temp0[11],carry0[11]);
fulladder fa132(A[12],B[12],carry0[11],temp0[12],carry0[12]);
fulladder fa133(A[13],B[13],carry0[12],temp0[13],carry0[13]);
fulladder fa134(A[14],B[14],carry0[13],temp0[14],carry0[14]);
fulladder fa135(A[15],B[15],carry0[14],temp0[15],carry0[15]);

fulladder fa14(A[0],B[0],1'b1,temp1[0],carry1[0]);
fulladder fa15(A[1],B[1],carry1[0],temp1[1],carry1[1]);
fulladder fa16(A[2],B[2],carry1[1],temp1[2],carry1[2]);
fulladder fa17(A[3],B[3],carry1[2],temp1[3],carry1[3]);
fulladder fa18(A[4],B[4],carry1[3],temp1[4],carry1[4]);
fulladder fa19(A[5],B[5],carry1[4],temp1[5],carry1[5]);
fulladder fa190(A[6],B[6],carry1[5],temp1[6],carry1[6]);
fulladder fa191(A[7],B[7],carry1[6],temp1[7],carry1[7]);
fulladder fa192(A[8],B[8],carry1[7],temp1[8],carry1[8]);
fulladder fa193(A[9],B[9],carry1[8],temp1[9],carry1[9]);
fulladder fa194(A[10],B[10],carry1[9],temp1[10],carry1[10]);
fulladder fa195(A[11],B[11],carry1[10],temp1[11],carry1[11]);
fulladder fa196(A[12],B[12],carry1[11],temp1[12],carry1[12]);
```



```
fulladder fa197(A[13],B[13],carry1[12],temp1[13],carry1[13]);
fulladder fa198(A[14],B[14],carry1[13],temp1[14],carry1[14]);
fulladder fa199(A[15],B[15],carry1[14],temp1[15],carry1[15]);

multiplexer2 mux_carry(carry0[15],carry1[15],cin,cout);

multiplexer2 mux_sum0(temp0[0],temp1[0],cin,S[0]);
multiplexer2 mux_sum1(temp0[1],temp1[1],cin,S[1]);
multiplexer2 mux_sum2(temp0[2],temp1[2],cin,S[2]);
multiplexer2 mux_sum3(temp0[3],temp1[3],cin,S[3]);
multiplexer2 mux_sum4(temp0[4],temp1[4],cin,S[4]);
multiplexer2 mux_sum5(temp0[5],temp1[5],cin,S[5]);
multiplexer2 mux_sum6(temp0[6],temp1[6],cin,S[6]);
multiplexer2 mux_sum7(temp0[7],temp1[7],cin,S[7]);
multiplexer2 mux_sum8(temp0[8],temp1[8],cin,S[8]);
multiplexer2 mux_sum9(temp0[9],temp1[9],cin,S[9]);
multiplexer2 mux_sum10(temp0[10],temp1[10],cin,S[10]);
multiplexer2 mux_sum11(temp0[11],temp1[11],cin,S[11]);
multiplexer2 mux_sum12(temp0[12],temp1[12],cin,S[12]);
multiplexer2 mux_sum13(temp0[13],temp1[13],cin,S[13]);
multiplexer2 mux_sum14(temp0[14],temp1[14],cin,S[14]);
multiplexer2 mux_sum15(temp0[15],temp1[15],cin,S[15]);

endmodule
```

Fig 4.57 16-Bit Carry Select Adder

In the above code (Figure 4.57), the design function is to implement the 16-Bit Carry Select Adder which have the inputs are A, B and Cin and Outputs are Sum is denoted by S and Carryout as cout. Along with inputs and outputs we require some internal wire which drives the signals we have taken the help of wire. As in this method we use total of Thirty two Full Adders in which Sixteen Adders are used when cin=0 and other Sixteen Adders when cin=1, and calculations are done parallelly and finally we need multilplexers to choose final Sum by choosing appropriate select lines. Here total Seventeen Multiplexers are used in which Sixteen are being involved in sum and the remaining one mux is used to choose the carryout.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

With the tremendous growth in VLSI, the products are manufacturing in great extent. This leads to the increase in demand of electronic products like Memory components, Processors, and other functional blocks. There is a need to proper flow from the algorithm level specification till transistor level. The first part of the work deals with Multipliers.

As there are various types of Multipliers and here in the thesis Vedic Multiplier is covered using the technique of Urdhva Triyagbhyam method. After understanding the topics which have been covered in this thesis, one can easily get the idea about how the UT method is good enough to get multiplication providing two levels of Adders at all stage of time i.e 2,4,8-Bit Vedic Multiplier. There is a possibility by varying the levels of Adders from two to one which increase the size of adders.

The Second half deals with the Adders as an efficient Multiplier is only possible with proper Adder selection. Here in the thesis we can see the Ripple Carry Adder, Carry Select Adder and Carry Look Ahead adder. And also at transistor level implementation of Ripple carry Adder upto 16-bit and each 16-bit Adders are implemented using various circuit design styles like Static CMOS, DCVSL, NORA based Adders and by knowing the delays and number of transistor we can choose appropriately choose the Adder depending on budget as there is a lot of possibilities coming from Number of levels of Adders, type of Adder, Ciruit design style of adder and size of Adders at a circuit level a minimum of 21 transistors to 28 transistors of a 1-bit Adder Full Adder is required.

These Vedic Multipliers can be efficiently used in Fast Fourier Transform and in Digital Signal Processing Applications as well as including the processors where basic Arithmetic operations are used.

REFERENCES

[1] Sameer Palitkar, “Verilog HDL A guide to Digital Design and Synthesis” SunSoft Press, 1996.

[2] Ankita Jain, “Design , Implementation & Comparission of Vedic Multipliers with Conventional Multipliers”, Springer.

[3] Pushpalata Verme, “ Design of 4*4 bit Vedic Multiplier using EDA Tool ”, International Journal of Computer Applications (0975 888)Volume 48 No. 20 June 2012.

[4] Anirudha Kanhe, Shishir Kumar Dasand Ankit Kumar Singh, “ Design and implementation of low Power Multiplier using Vedic Multiplication Technique ”, International Journal of Computer Applications, Vol. 3, No.1, June 2012,pp. 131-132.

[5] Kavita, Umesh Goyal, “Performance Analysis of Various Vedic Techniques For Multiplication” International Journal of Engineering Trends and Technology- Volume4 Issue3-2013.

[6] Sung-Mo Kang, Yusuf Leblebici. (199).Mc Graw Hill Higher Education : CMOS Digital Integrated Circuits

[7] Rabaey, Pedram. (1996). Kluwer Academic Publisher: Low Power Design Methodoloy.