Designing a 32-bit Multiplier using transistor-level Digital Gates simulated in SPICE

Pankaj Kumar Sah

Electronics and Telecommunication Dept.
ASTU Roll number -180610026032
Assam Engineering College
Guwahati, India
52punk2017@gmail.com

Mrinab Dey

Electronics and Telecommunication Dept.
ASTU Roll number -180610026026
Assam Engineering College
Guwahati, India
mrinab17@gmail.com

Souvik Das

Electronics and Telecommunication Dept.
ASTU Roll number -180610026045
Assam Engineering College
Guwahati, India
sd411524@gmail.com

Gaurab Paul

Electronics and Telecommunication Dept.
ASTU Roll number -180610026016
Assam Engineering College
Guwahati, India
gauravpaul744@gmail.com

Dip Jyoti Dutta

Electronics and Telecommunication Dept.
ASTU Roll number -180610026015
Assam Engineering College
Guwahati, India
dipjyotidutta97@gmail.com

***Abstract*—This paper is based on the design and simulation of a 32-bit multiplier using transistor-level digital Gates in SPICE. The design is structured with AND operation of one 32 bit number with the first bit of 2nd 32-bit number to get the 1st bit of the 64bit product and again doing the same with the second bit of the second 32-bit number then adding the result to get the second bit of 64-bit product and further iterating these steps for the next bits of the second 32-bit number and adding it to get the next bit of 64-bit product. After successfully achieving the output for the logic in a 16-bit multiplier, 32 bit model of the multiplier could not be tested due to the underperformance of the low-end device. This implies the logic which has been used to tackle the problem is correct.**

Keywords— Multiplier, 32 bit, CMOS, SPICE, simulation, logic gates, CppSim, Full adder.

I. Introduction

Firstly, I would to like quote that "What is a Multiplier !?", and the answer is simple as that, It follows the normal mathematical operation while we do in multiplying by taking the two numbers as usually we do.., now the technical thing about it is that, the multiplication is what the most fundamental operation in most digital signal processing(DSP) algorithms to perform functions like convolution, filtering and processing, so on... The major recent statics shows that more than 69.90% of the instructions perform addition and multiplication in most of the microprocessor and DSP algorithms .i.e., these operations consume most of the execution time after simulation on compiling. In a system comprising of a multiplier, the system performance usually determined by the performance of the multiplier as it being the slowest element of all. Henceforth, optimizing the speed of the multiplier is a major design issue.
Here due to the complexity and design of different logical circuits combine to form the implementation of the basic 32-bit Multiplier, the speed of running or execution becomes slower, and that's why, the prior multiplication process can be divided into three steps, namely, generating the partial products, reducing the partial product and the last addition to getting the final product, and also for the speed of multiplication can be improved by reduction in the generated number of partial products or thus by increasing the speed at which these partial products are accumulated.
The main proposed solution of the good multiplier is to provide a compact utilization, high speed and low power consumption unit at a certain level of implementation for the betterment of the speed and power usage.
The combinational logical components, we use for the implementation and designing of the 32-Bit Multiplier are 32-bit full adder, 32-bit AND gate, the inputs and output lines and for the display of output in CPPsim and the general purpose of multiplication of bits.
Adder->Multiplication employs addition in its operations, and their hardware is similar if not identical to additional hardware. Thus, an adder or multiple adders will be in the critical path of the design, so the performance of implementation will be often be limited by the performance of its adders. When we are looking at other attributes of a chip, such as an area or power, it is found that the hardware for addition will be a large contributor to these areas. It is therefore beneficial to choose the correct adder to implement in a design. A Ripple Carry Adder is a logical circuit using multiple full adders (FA) to add N-bit numbers. Each FA inputs a carry Cin which is the Cout of the previous adder. This kind of adder is a Ripple Carry Adder (RCA) since each carry bit "ripples" to the next full adder. The first FA can be replaced by a HA. The layout of an RCA is simple, which allows fast design time. However, the ripple carry adder is relatively slow as it has to wait for the carry bit that comes from the previous full adder.

Sum (Si) = (Ai xor Bi) xorCi
Carry (Ci+1) = (Ai and Bi) or (Ci and (Ai xor Bi))

AND gate->The output state of a digital logic AND gate only returns "LOW" again when ANY of its inputs are at a logic level "0". In other words for a logic AND gate, any LOW input will give a LOW output.

II. Block Diagram Of Design: Architecture

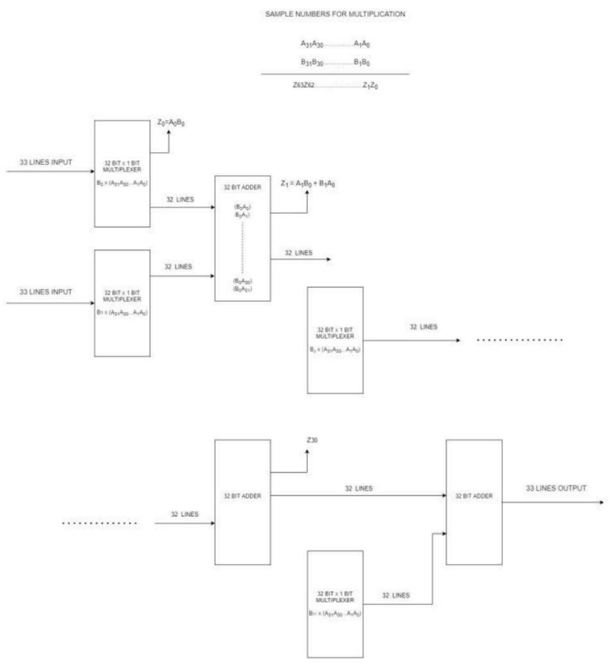
SAMPLE NUMBERS FOR MULTIPLICATION
$A_{31}A_{30}$ .... A_1A_0
$B_{31}B_{30}$ .... B_1B_0
$Z63Z62$ .... Z_1Z_0
$Z_0=A_0B_0$
33 LINES INPUT
32 BIT x 1 BIT MULTIPLEXER
$B_0 \times (A_{31}A_{30}...A_1A_0)$
32 LINES
32 BIT ADDER
(B_0A_0) $B_1A_1)$
(B_0A_{30}) (B_0A_{31})
$Z_1 = A_1B_0 + B_1A_0$
32 LINES
33 LINES INPUT
32 BIT x 1 BIT MULTIPLEXER
$B_1 \times (A_{31}A_{30}...A_1A_0)$
32 LINES
32 BIT x 1 BIT MULTIPLEXER
$B_2 \times (A_{31}A_{30}...A_1A_0)$
32 LINES
Z30
32 BIT ADDER
32 LINES
32 LINES
32 BIT ADDER
33 LINES OUTPUT
32 BIT x 1 BIT MULTIPLEXER
$B_{31} \times (A_{31}A_{30}...A_1A_0)$
32 LINES

Fig 1. Block Diagram of 32bit multiplier

The design consists of feeding 1st 32-bit number with 1st bit of 2nd 32-bit number to 32*1 AND gate then extracting the 1st bit of the 64bit output (A_0B_0) then again doing the same operation with B_1 i.e., 2nd bit of the 2nd 32-bit number and then adding the two partial products with the help of 32bit Full Adder circuit to get the 2nd bit of the 64-bit output i.e., $A_1B_1 + A_0B_1$., and then iterating these steps for 31 times to get the left part of the 64-bit number and the rest bits are extracted from the last full adder (Number 31) output.

III. Individual Component Schematics With Explanation

1. NAND Gate

The NAND Gate is a digital logic circuit. The output of the NAND Gate is high to a logic level 1 when any one of its input is low. If both of its inputs are high, the output of the NAND Gate is low to a logic level of 0. The Boolean expression of the NAND Gate is the same as that of the inverse of logical multiplication. The symbol of a NAND Gate is as follows: First, confirm that you have the correct template for your paper size. This template has been tailored for output on the A4 paper size. If you are using US letter-sized paper, please close this file and download the Microsoft Word, Letter file.

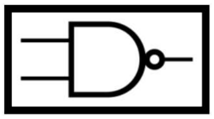

Fig 2. NAND Gate (Symbol)

Schematics of the NAND Gate:

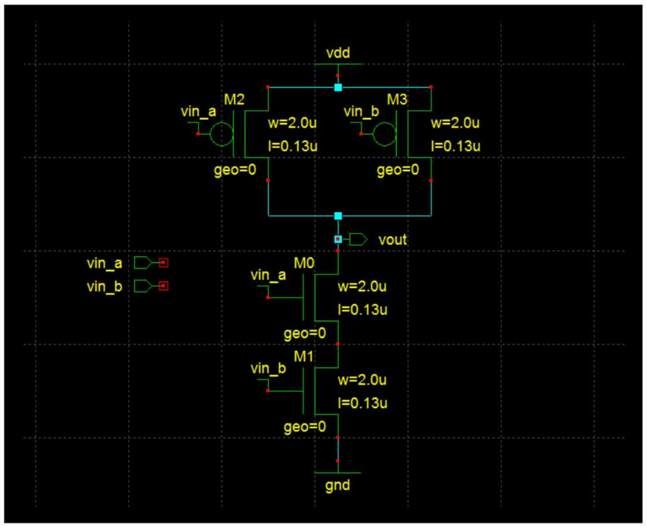

Fig 3. NAND Gate (Transistor level)

The components of the above schematic are:

- NMOS Transistor: N-type Metal Oxide Semiconductor (NMOS) Transistor is built with n-type source and drain and p-type substrate. When a high voltage is applied to the gate, NMOS will conduct. When a low voltage is applied to the gate, NMOS will not conduct.
- PMOS Transistor: P-type Metal Oxide Semiconductor (PMOS) Transistor is built with p-type source and drain and n-type substrate. When a high voltage is applied to the gate, PMOS will not conduct. When a low voltage is applied to the gate, PMOS will conduct.
- VDD: Voltage Drain Drain (VDD) is a constant positive supply voltage.
- Ground: Ground (GND) is a reference point and carries a voltage of 0V.

2. NOR Gate

The NOR Gate is a digital logic circuit. The output of the NOR Gate is high to a logic level 1 when both of its inputs are low. If any one of its input is high, the output of the NOR Gate is low to a logic level of 0. The Boolean expression of the NOR Gate is the same as that of the inverse of logical addition. The symbol of a NOR Gate is as follows:

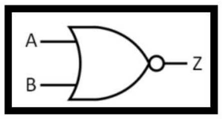

Fig 4. NOR Gate (Symbol)

Schematics of the NOR Gate:

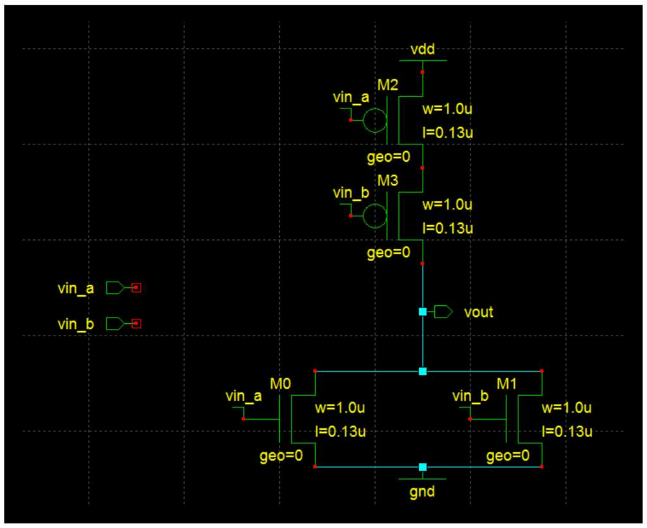

Fig 5. NOR Gate (Transistor level)

The components of the above schematic are the same as that of the NAND Gate.

3. *NOT Gate*

The NOT Gate is a digital logic circuit. The output of the NOT Gate is high to a logic level 1 when its input is low. If its input is high, the output of the NOT Gate is low to a logic level of 0. The Boolean expression of the NOT Gate is the same as that of the inverse of the input. The symbol of a NOR Gate is as follows:

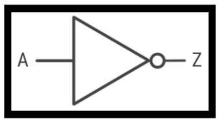

Fig 6. NOT Gate (Symbol)

Schematics of the NOT Gate:

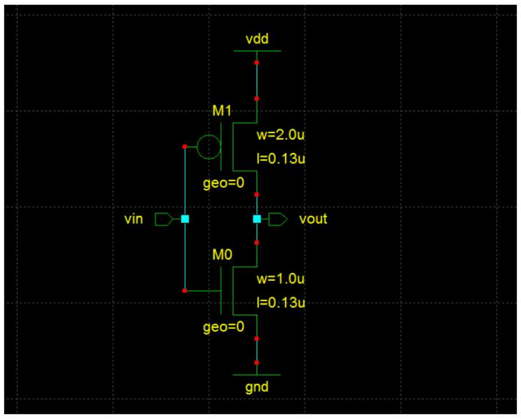

Fig 7. NOT Gate (Transistor level)

The components of the above schematic are the same as that of the NAND Gate.

4. *AND Gate*

The AND Gate is a digital logic circuit. The output of the AND Gate is high to a logic level 1 when both of its inputs are high. If any one of its input is low, the output of the AND Gate is low to a logic level of 0. The Boolean expression of the AND Gate is the same as that of logical multiplication. The symbol of an AND Gate is as follows:

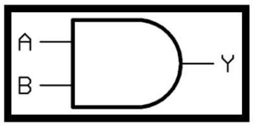

Fig 8. AND Gate (Symbol)

Schematics of the AND Gate:

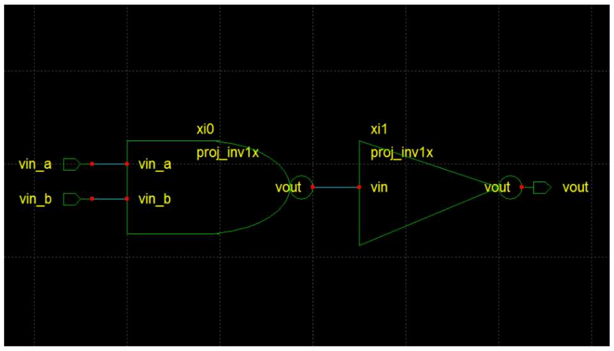

Fig 9. AND Gate (Transistor level)

The components of the above schematic are the same as that of the NAND Gate.

5. *OR Gate*

The OR Gate is a digital logic circuit. The output of the OR Gate is high to a logic level 1 when any one of its input is high. If both of its inputs are low, the output of the OR Gate is low to a logic level of 0. The Boolean expression of the OR Gate is the same as that of logical addition. The symbol of an OR Gate is as follows:

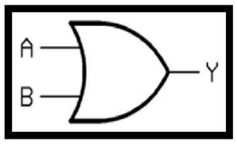

Fig 10. OR Gate (Symbol)

Schematics of the OR Gate:

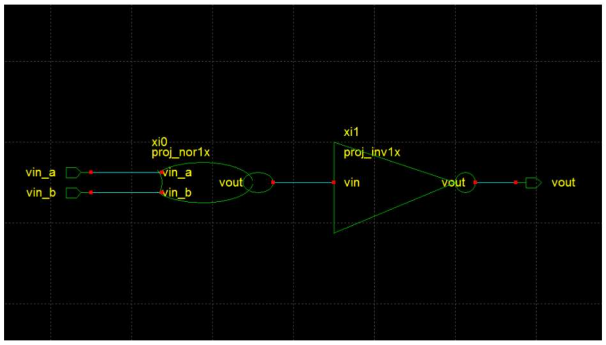

Fig 11. OR Gate (Transistor level)

The Components of the OR gate is the same as that of the NAND gate.

6. *XOR Gate*

The XOR Gate was made using four NAND Gates only. XOR gate is a digital logic circuit that gives high output only when the number of high inputs is odd. The XOR Gate implements an exclusive or which means a true result is obtained when only one of the inputs is true.

Schematics of the XOR Gate:

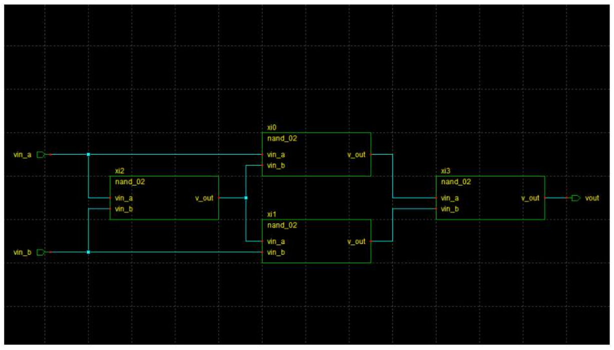

Fig 12. XOR Gate (Transistor level)

7. *Half Adder*

The Half Adder was made using one XOR Gate, one NAND Gate and one NOT Gate. The half adder is a digital logic circuit that performs the addition of two binary digits and produces the corresponding sum and carry value which is also binary digits.

Schematics of the Half Adder:

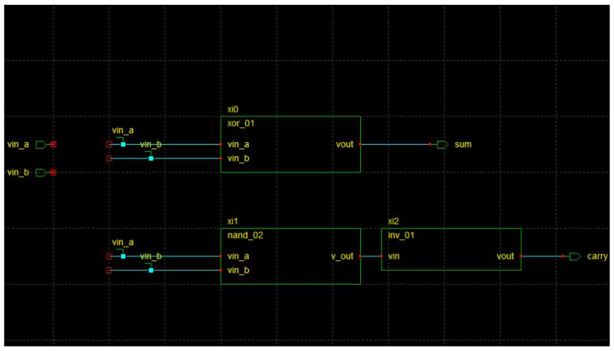

Fig 13. Half Adder (Transistor level)

8. *Full Adder*

The Full Adder was made using two Half Adders and one OR Gate. The full adder is a digital logic circuit that performs the addition of two binary digits, plus a carry-in digit and produces the corresponding sum and carry value which is also binary digits.

Schematics of the Full Adder:

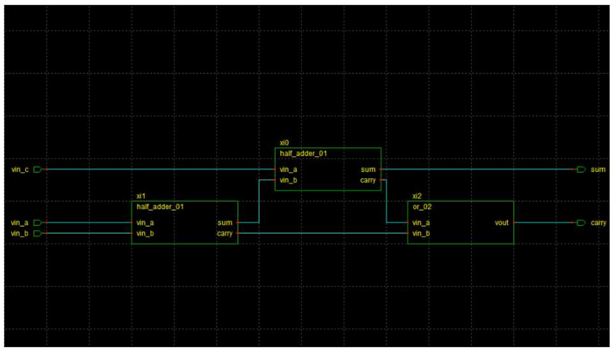

Fig 14. Full Adder (Transistor level)

9. *32-bit Full Adder*

Two of the above full adders were used to make a 2-bit Full Adder. Two 2-bit Full Adders were used to make a 4-bit full adder. Two 4-bit Full Adders were made into an 8-bit Full Adder and so on. Finally, two 16-bit Full Adders were used to make a 32-bit Full Adder. The 32-bit full adder is used to add a 32-bit binary number with another 32-bit binary number and produces the corresponding output which can be up to 64-bits long.

Schematics of the 32-bit Full Adder:

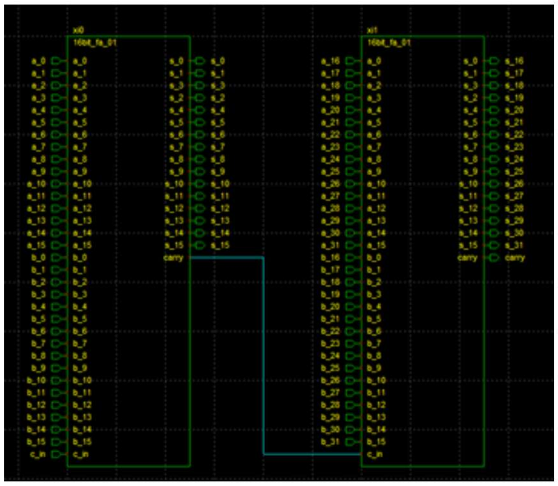

Fig 15. 32-bit Full Adder (Transistor level)

10. *32 bit×1 bit multiplier*

Thirty-two AND Gates were used to make the 32 bit×1 bit multiplier. The 32 bit×1 bit multiplier is used to multiply a 32-bit number with a 1-bit number. The corresponding output is 32 bits long.

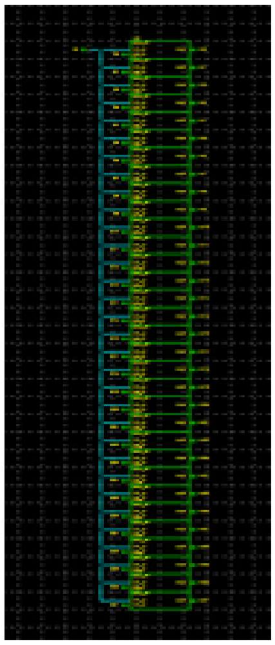

Fig 16. 32*1 bit AND Gate (Transistor level)

11. 32-bit Multiplier

Two 32 bit numbers are provided at the input and a 64-bit number is extracted as the output (product). The product is achieved by doing AND operation using 1st 32-bit number with 1st bit of 2nd 32-bit number and then adding them to get the 2nd bit of the product because 1st bit of product is extracted at the 1st output terminal of 32 *1 AND gate.

Schematic of the 32-bit Multiplier:

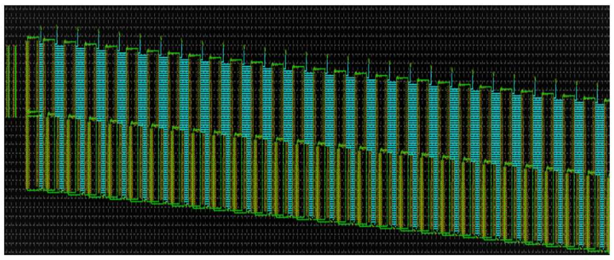

Fig 17. 32-bit Multiplier Schematic (Transistor level)

IV. Detailed Test Bench Description

We were unable to simulate the 32-bit design of our multiplier due to our low-end systems, so we decided to simulate a lesser number of bits i.e., 16-bit multiplier with the same logic as we used for a 32-bit multiplier, so if this simulation processed without any problem then it implies that our 32-bit design is also correct.

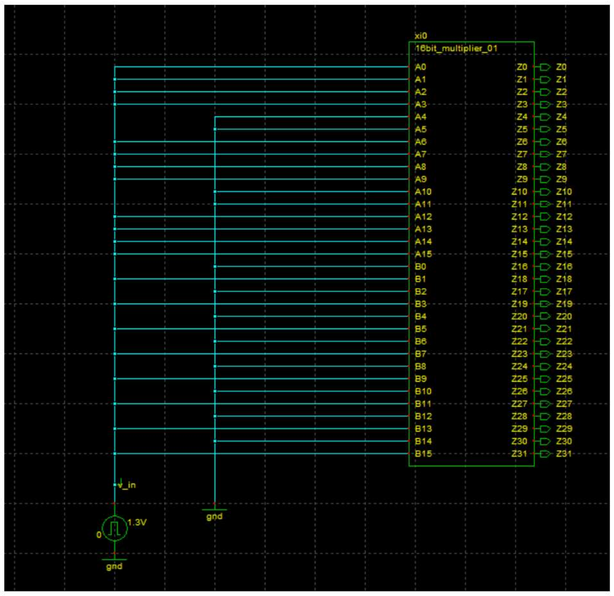

Fig 18. Test Bench of 32-bit Multiplier Schematic (Transistor level)

The above schematic is the test bench for our 16bit multiplier, it accepts 2 16-bit numbers as the input and gives one 32-bit output as the result.
For testing the 16bit multiplier we used pulse from the Spice Module in the CppSim Sue2 application to give the input 1 (1.3V) and ground (global pin from Devices module renamed as GND,) to use the input 0.

We used the two numbers for testing:
Input 1: 1111001111001111 ---- 62415 (decimal form)
Input 2: 1010101010101010 ---- 43690 (decimal form)
Output: 10100010100010010101110101110110
(Theoretical)
---- 2726911350 (decimal form)

For supplying the1st 32-input
we connected the input pins A0 A1 A2 A3 A6 A7 A8 A9 A12 A13 A14 A15 to the pulse signal (1)
and A4 A5 A10 A11 to the ground (0)
For 2nd 32-bit input
we connected the input pins A1 A3 A5 A7 A9 A11 A13 A15 to the pulse signal (1)
and A0 A2 A4 A6 A8 A10 A12 A14 to the ground (0)

And we got the correct practical output corresponding to the input:

Sl no.	Z (Product Variable)	
	Theoretical	*Practical (Simulated)*
0	0	0
1	1	1
2	1	1
3	0	0
4	1	1
5	1	1
6	1	1
7	0	0
8	1	1
9	0	0
10	1	1
11	1	1
12	1	1
13	0	0
14	1	1
15	0	0
16	1	1
17	0	0
18	0	0
19	1	1
20	0	0
21	0	0
22	0	0
23	1	1
24	0	0
25	1	1
26	0	0
27	0	0
28	0	0
29	1	1
30	0	0
31	1	1

V. Simulations Plots Of inputs, Outputs And Important Intermediate Signals

For simulation we used the CppSim software, Sue2 application for designing the circuit and the test bench and CppSimView application for viewing the results in Graphs.

We used the pulse signal names as v_in i.e., if v_in is equal to 1 then we will get the desired input as described in the test bench description section.

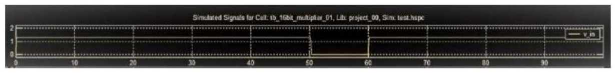

Fig 19. v_in (input signal)

From the image:
v_in -> 1 for 50 ns and 0 for 10 ns

Output plots:

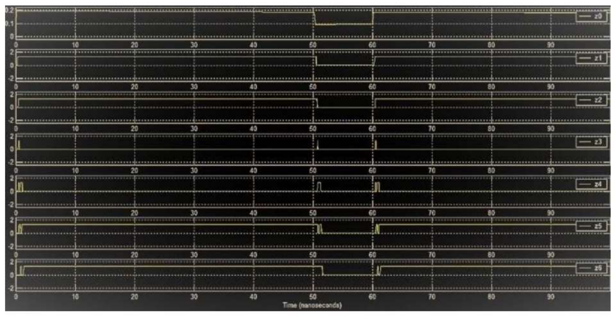

Fig 20. z0 to z6

From the image
Z0 -> 0, Z1 -> 1, Z2 -> 1, Z3 -> 0, Z4 -> 1, Z5 -> 1, Z6 -> 1

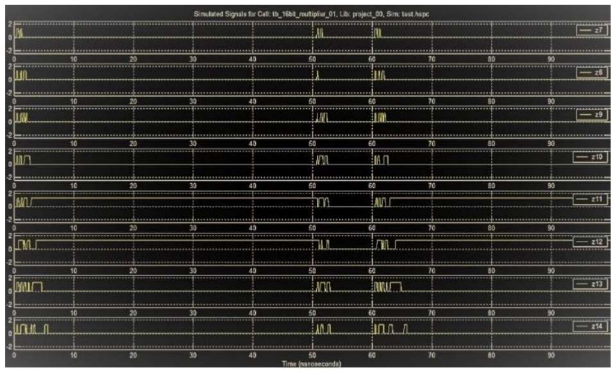

Fig 21. z7 to z14

From the image
Z7 -> 0, Z8 -> 1, Z9 -> 0, Z10 -> 1, Z11 -> 1, Z12 -> 1, Z13 -> 0, Z14 -> 1

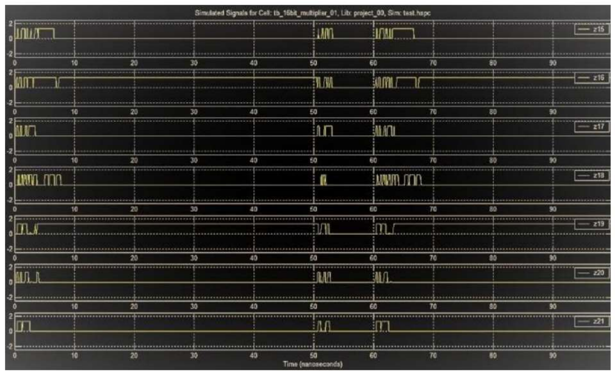

Fig 22. z15 to z21

From the image
Z15 -> 0, Z16 -> 1, Z17 -> 0, Z18 -> 0, Z19 -> 1, Z20 -> 0, Z21 -> 0

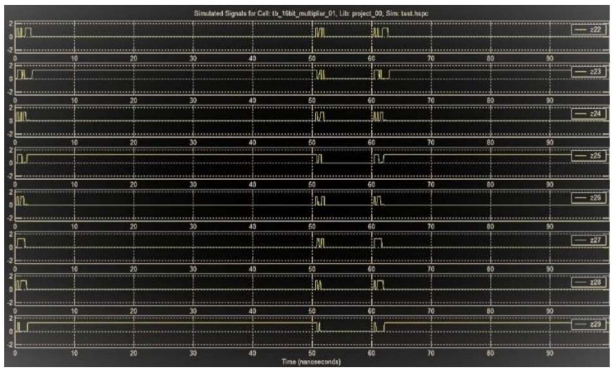

Fig 23. z22 to z29

From the image
Z22 -> 0, Z23 -> 1, Z24 -> 0, Z25 -> 1, Z26 -> 0, Z27 -> 0, Z28 -> 0, Z29 -> 1

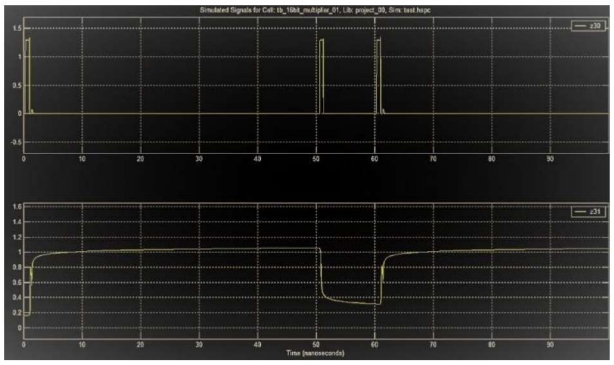

Fig 24. z30 and z31

From the image
Z30 -> 0, Z31 -> 1

Thus, from the given plots it is clear that the logic behind the solution was correct, only due to the unavailability of high-end devices we were unable to simulate the 32-bit version of the multiplier.

VI. Conclusion And Scope Of Development

The project on " 32-Bit Multiplier by transistor-level gates " through the VLSI Experts', has helped us to gain the insights and heights of the complete process in this very project of RISC-V using the spice.
During this project, we have identified that on the implementation of the design of 32 – bit Multiplier, that we have made using the Sue2, is that, we were unable to make it towards our desire output (32-bit Multiplier output); but, on behalf of that, we had to make another schematic of the shortly represented of 32-bit Multiplier i.e., the 16-bit Multiplier and which helps us to get the output on CPPsim.
From this Summer Internship on such wonderful platform of "VLSI Experts'", I would also like to conclude that, this project helps us to get on to the industry level of perspective and experiences on how to manage the task of doing the project on a team, very gradually and sequentially.
Hence, we did the work on the design and implementation of the 32-Bit Multiplier by using those main featured components as such like 32-Bit AND gates, Full Adders and all, etc.
On and all from this project, that is the 32-Bit Multiplier; we have acquired a lot as well as plenty amount of good knowledge and wisdom on using the Sue2 and CPPsim simulations and schematics, that is by far, we have made on the process of doing the project of completion.

For Scope of development:

We can use hexadecimal representation in input and output Instead of binary representation as it becomes difficult to give 32bit input and to check the 64-bit product.
The process needs to be modified so that the multiplier becomes more efficient and fast to give the result.
The 32-bit multiplier is to used in the processor which we are planning to design in future as our next project.