EXPT.-02

AIM: - Design and Simulate Half-adder and Full-adder using Verilog HDL.

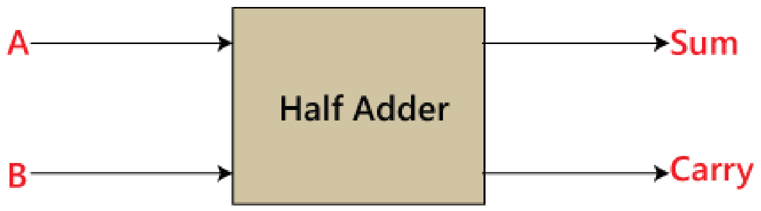
Software: - Xilinx Vivado 2019.1

Theory: -

Half Adder

The Half-Adder is a basic building block of adding two numbers as two inputs and produce out two outputs. The adder is used to perform OR operation of two single bit binary numbers. The **augent** and **addent** bits are two input states, and **'carry**' and **'sum** 'are two output states of the half adder.

Block diagram



Truth Table

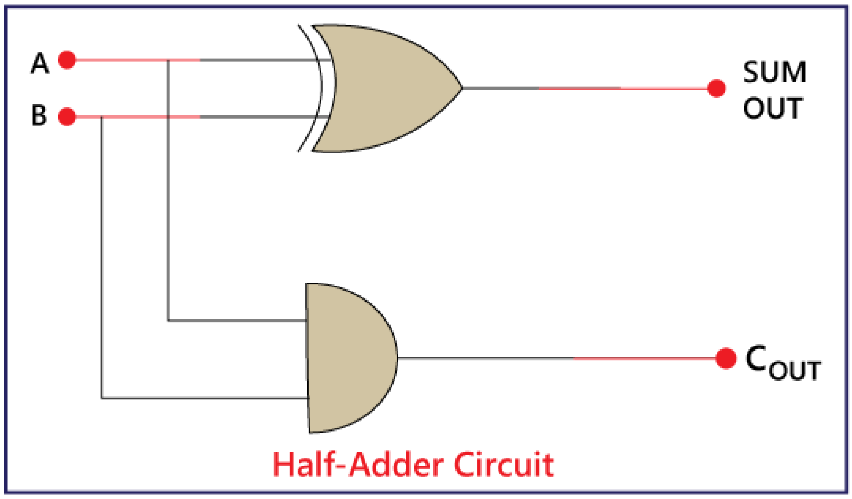
Inputs		Outputs	
A	**B**	**Sum**	**Carry**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

In the above table,

1. 'A' and' B' are the input states, and 'sum' and 'carry' are the output states.
2. The carry output is 0 in case where both the inputs are not 1.
3. The least significant bit of the sum is defined by the 'sum' bit.

The SOP form of the sum and carry are as follows:

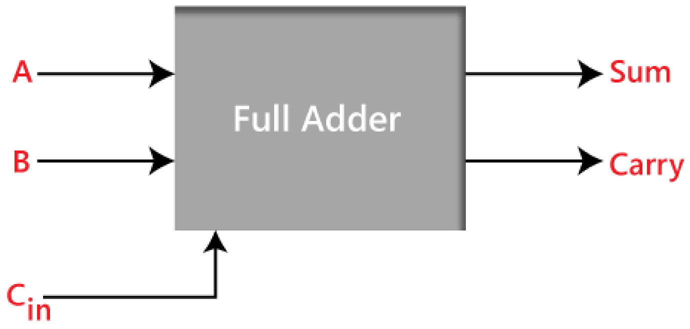
Sum=x'y+xy'
Carry = xy



Full Adder

The half adder is used to add only two numbers. To overcome this problem, the full adder was developed. The full adder is used to add three 1-bit binary numbers A, B, and carry C. The full adder has three input states and two output states i.e., sum and carry.

Block diagram



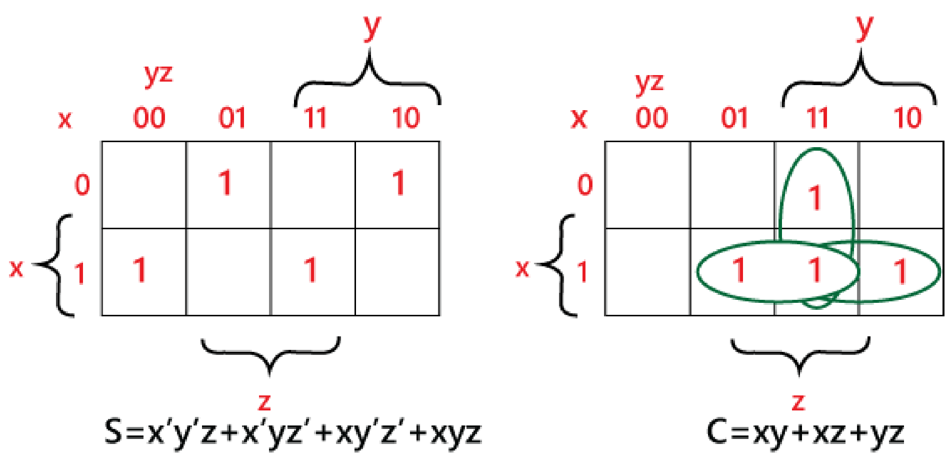
Truth Table

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
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

In the above table,

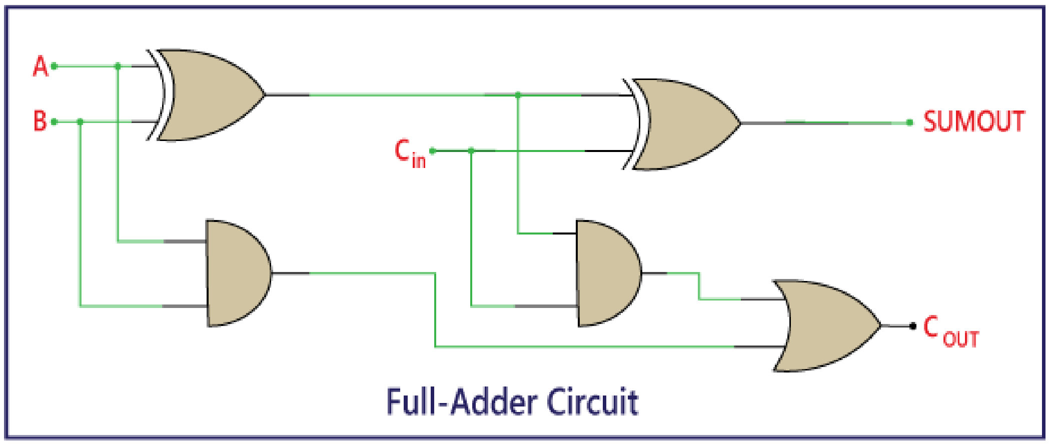
1. 'A' and' B' are the input variables. These variables represent the two significant bits which are going to be added
2. 'C_{in}' is the third input which represents the carry. From the previous lower significant position, the carry bit is fetched.
3. The 'Sum' and 'Carry' are the output variables that define the output values.
4. The eight rows under the input variable designate all possible combinations of 0 and 1 that can occur in these variables.

The SOP form can be obtained with the help of K-map as:

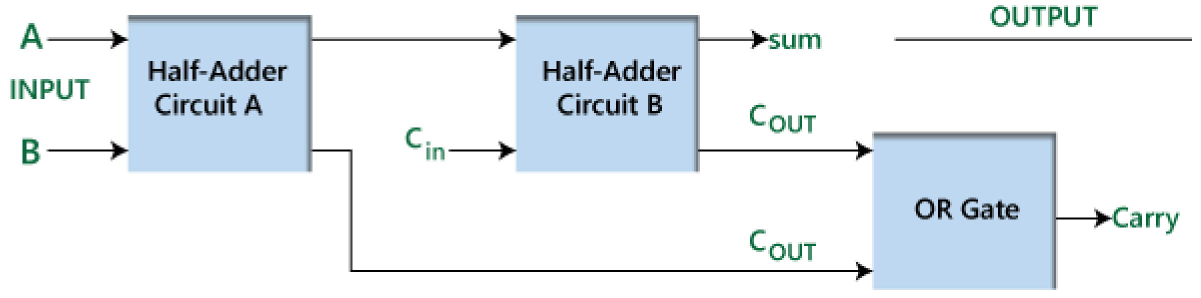


Sum = x' y' z+x' yz+xy' z'+xyz

Carry = xy+xz+yz



Construction of Half Adder Circuit:



Code: -

```
//half adder
`timescale 1ns / 1ps

module half_adder(a,b,sum,carry);
input a,b;
output sum,carry;

xor G1(sum,a,b);
and G2(carry,a,b);

endmodule
```

```
//full adder
`timescale 1ns / 1ps

module full_adder(a,b,c,sum,cout);
input a,b,c;
output sum,cout;
wire t1,t2,t3;

half_adder HA1(a,b,t1,t2);
half_adder HA2(t1,c,sum,t3);
and (cout,t2,t3);

endmodule
```

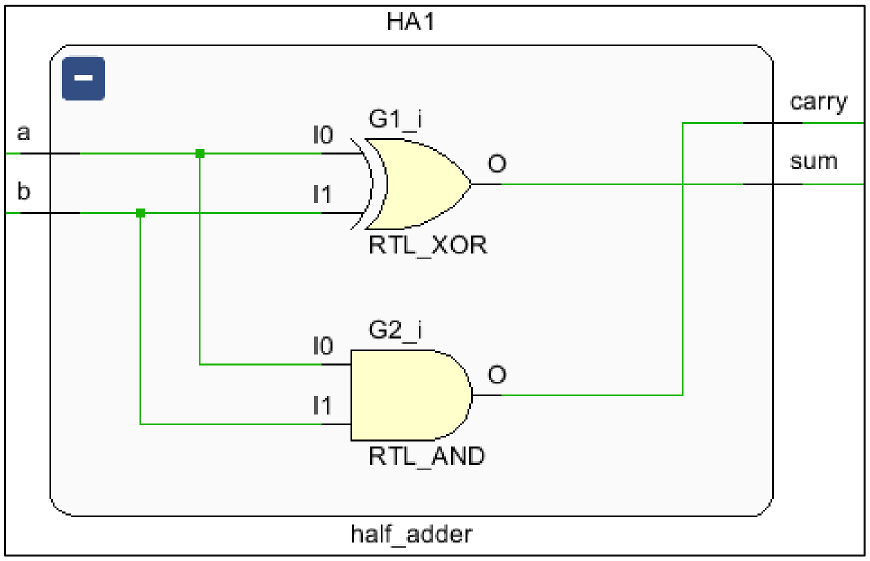
```
//test bench half adder
`timescale 1ns / 1ps

module tb();
reg a,b;
wire sum,carry;

half_adder HA(.a(a),.b(b),.sum(sum),.carry(carry));

integer i;
initial
begin
{a,b}=2'b00;
for(i=0;i<4;i=i+1)
begin
{a,b}=i;
#10;
end
end

initial
#40 $finish;

initial
$monitor($time,"a=%b,b=%b,sum=%b,carry=%b",a,b,sum,carry);

endmodule
```

```
/test bench full adder
`timescale 1ns / 1ps

module tb();
reg a,b,c;
wire sum,carry;

full_adder FA(.a(a),.b(b),.c(c),.sum(sum),.cout(carry));

integer i;
initial
begin
for(i=0;i<8;i=i+1)
begin
{a,b,c}=i;
#10;
end
end

initial
#80 $finish;

initial
$monitor($time,"a=%b,b=%b,sum=%b,carry=%b",a,b,sum,carry);

endmodule
```

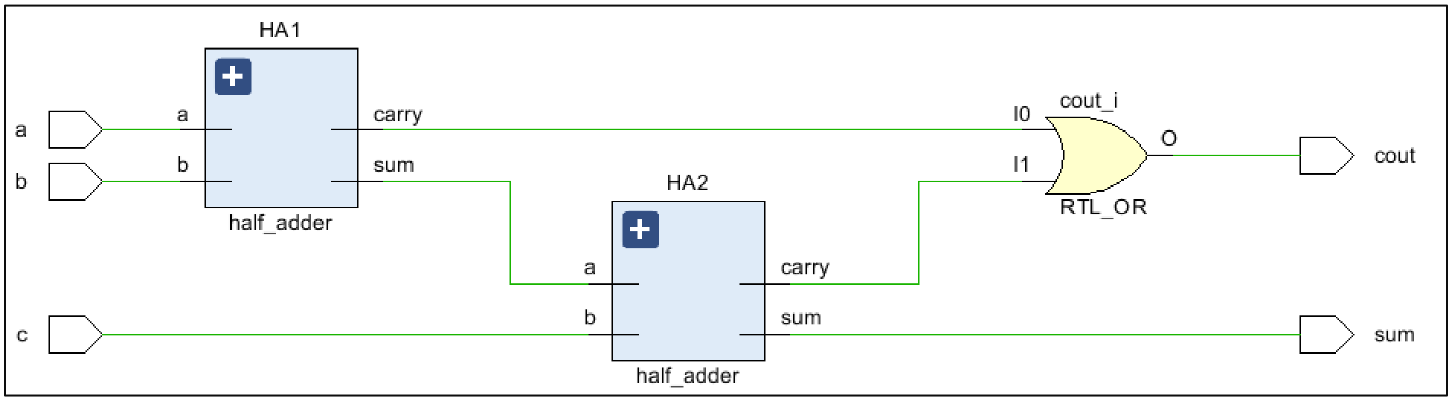
Schematics and Simulated Waveforms: -

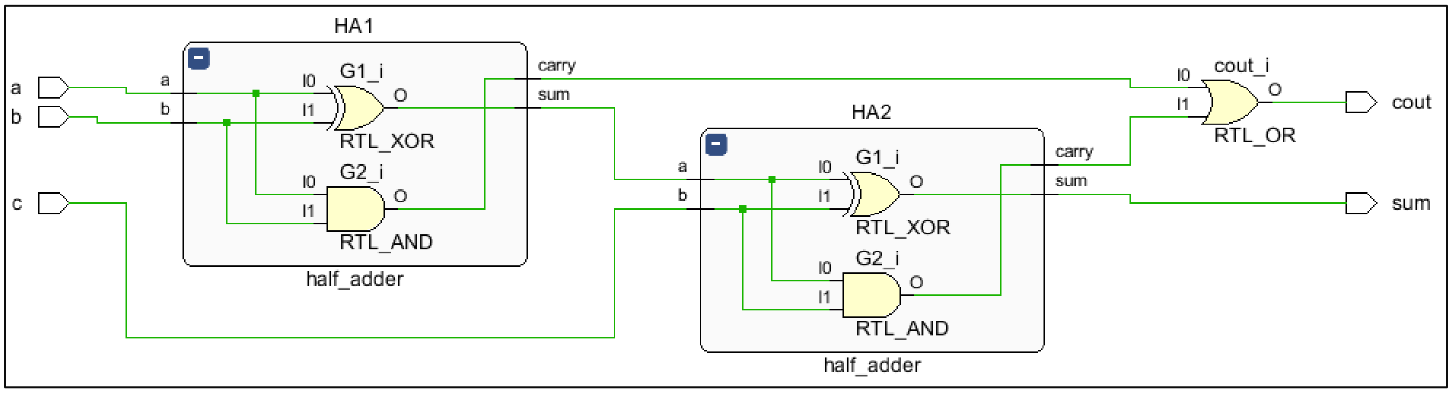
Schematic:

Half adder



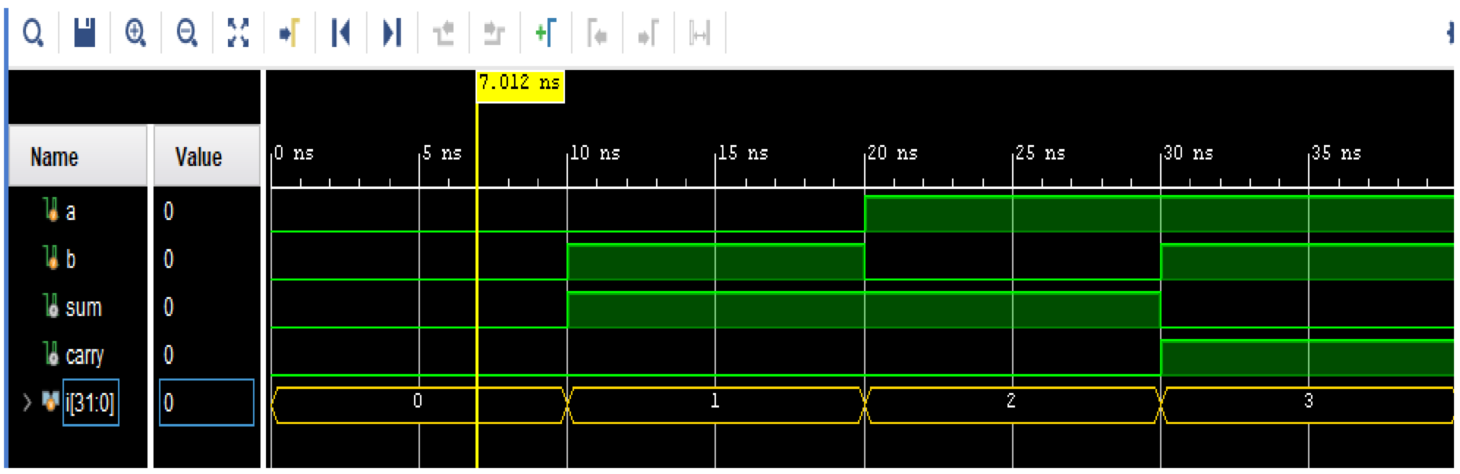
Full adder

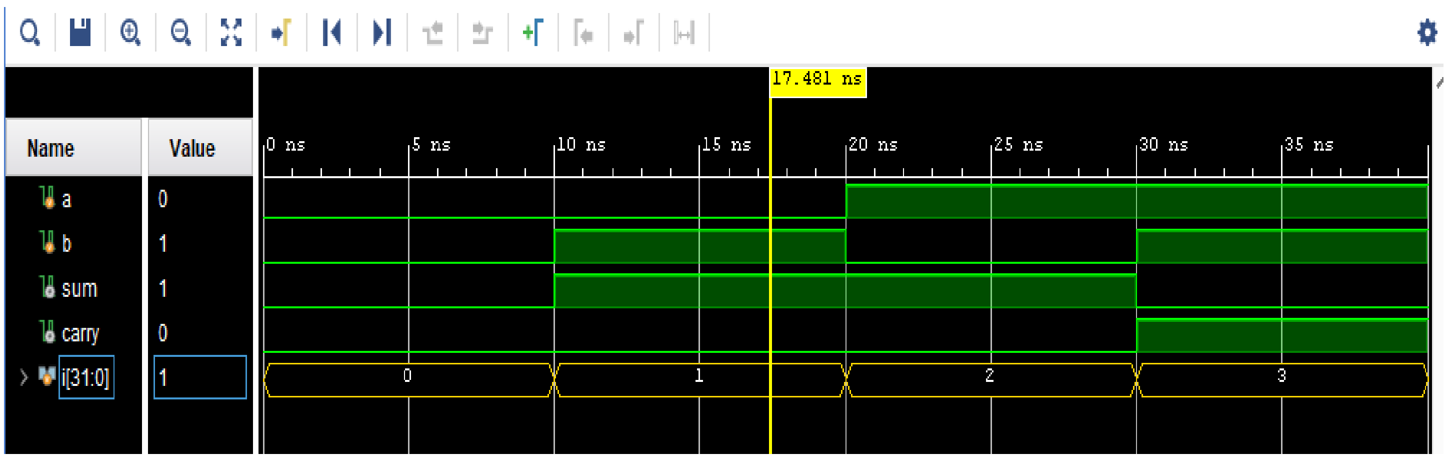


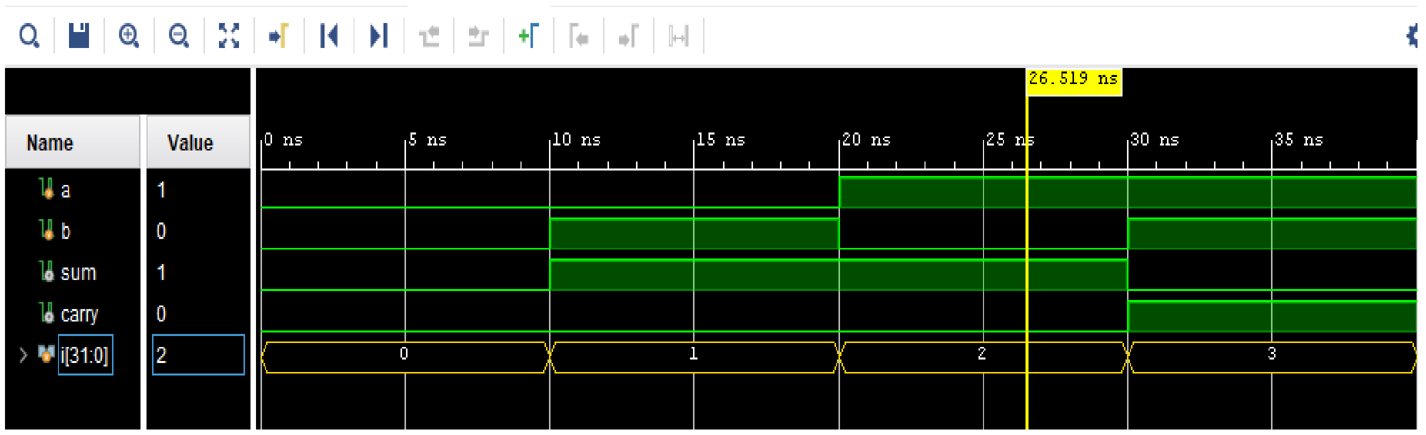


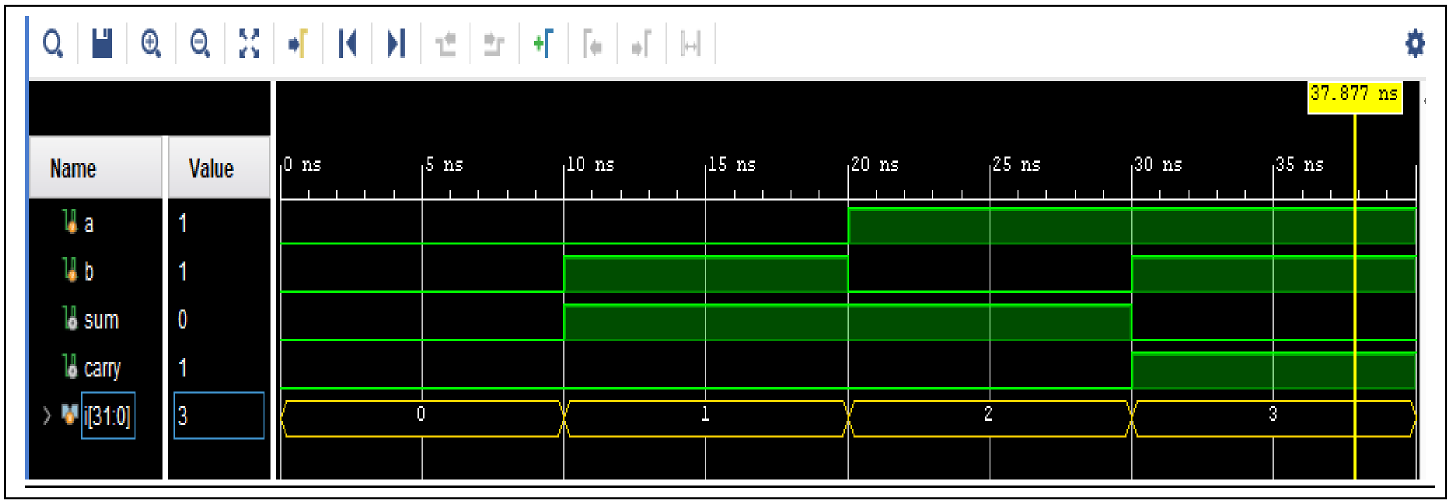
Waveform:

Half adder

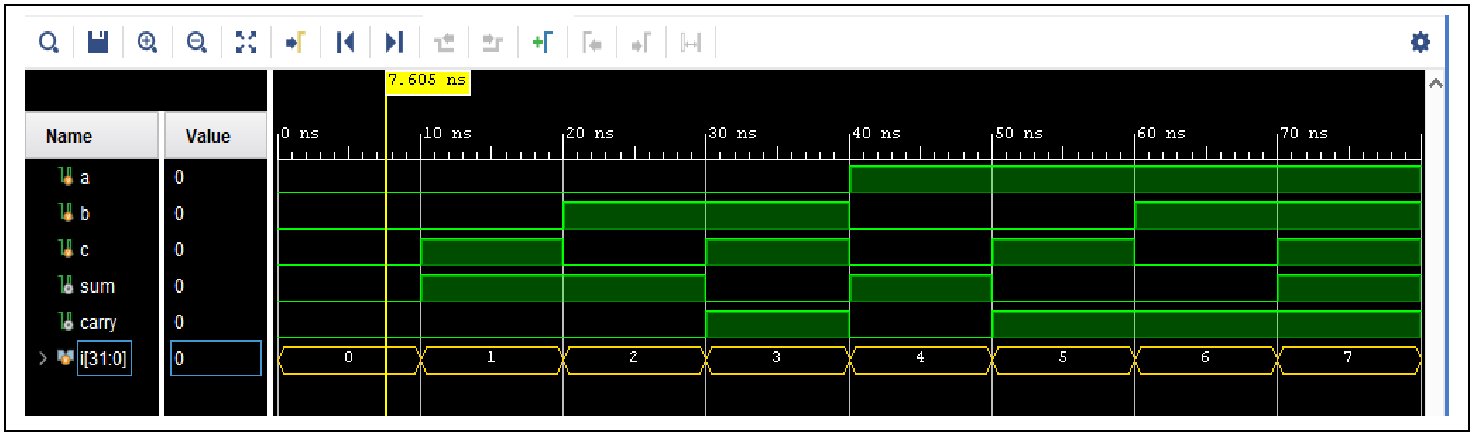


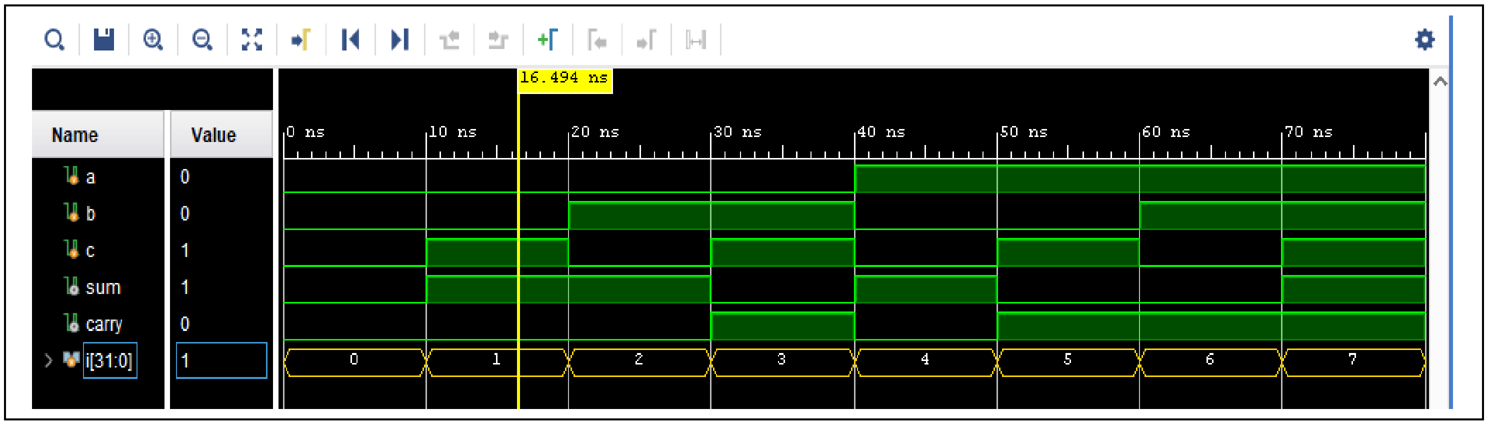


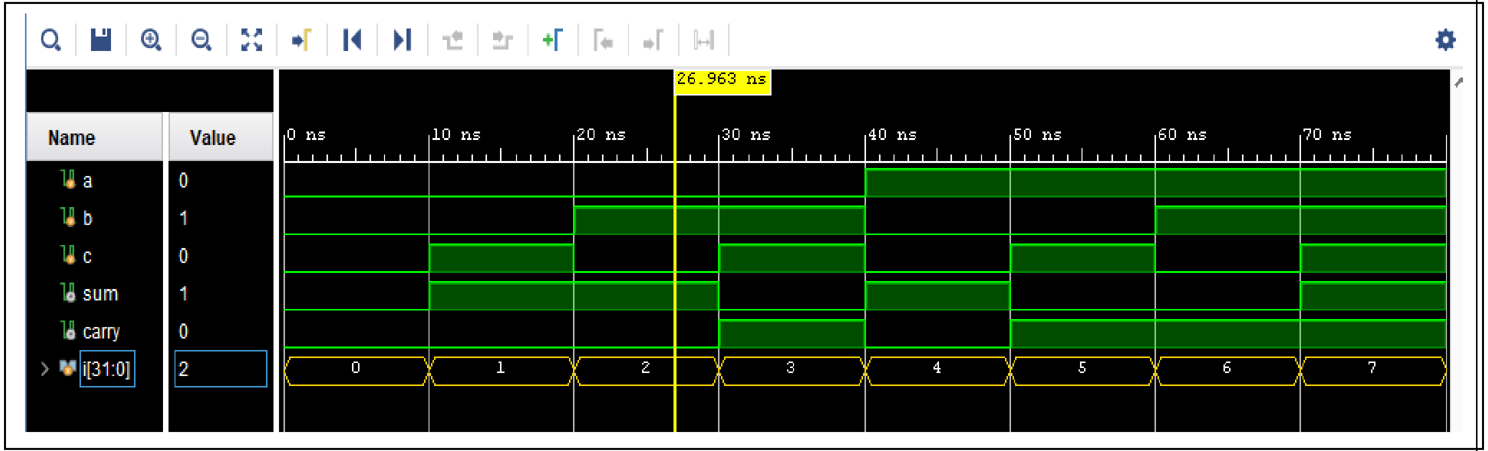



Name
Value
a 1
b 1
sum 0
carry 1
i[31:0] 3
37.877 ns
0 ns
5 ns
10 ns
15 ns
20 ns
25 ns
30 ns
35 ns
0
1
2
3

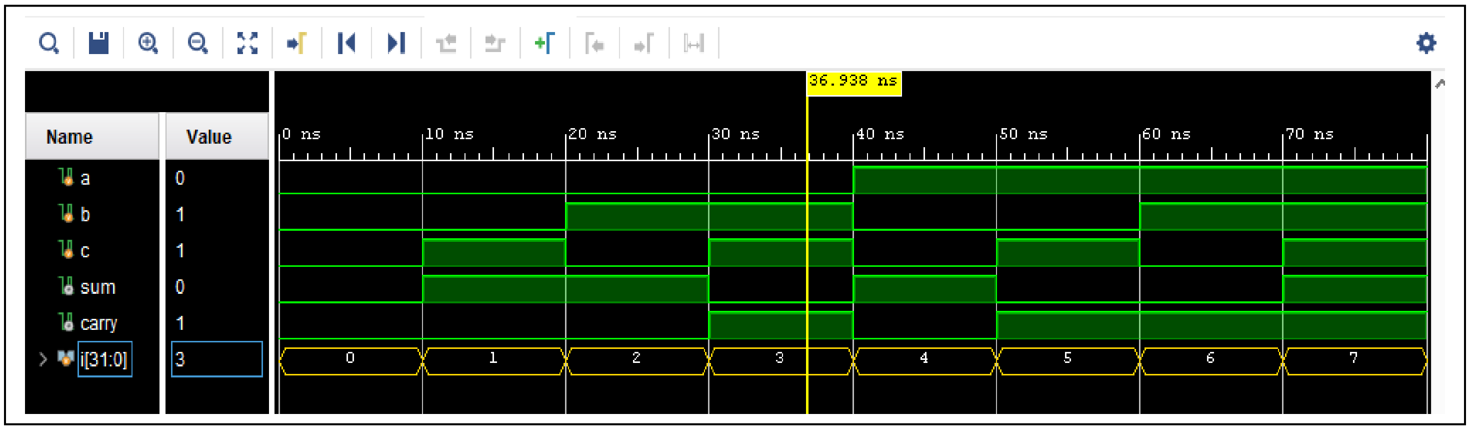
Full adder


Name
Value
a 0
b 0
c 0
sum 0
carry 0
i[31:0] 0
7.605 ns
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
0
1
2
3
4
5
6
7

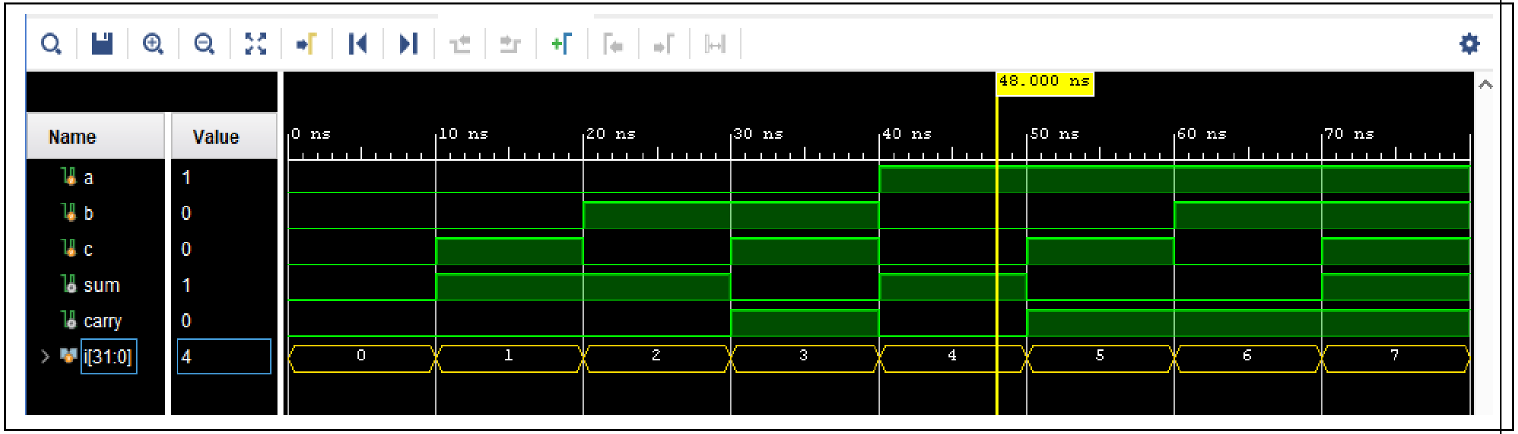

Name
Value
a 0
b 0
c 1
sum 1
carry 0
i[31:0] 1
16.494 ns
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
0
1
2
3
4
5
6
7



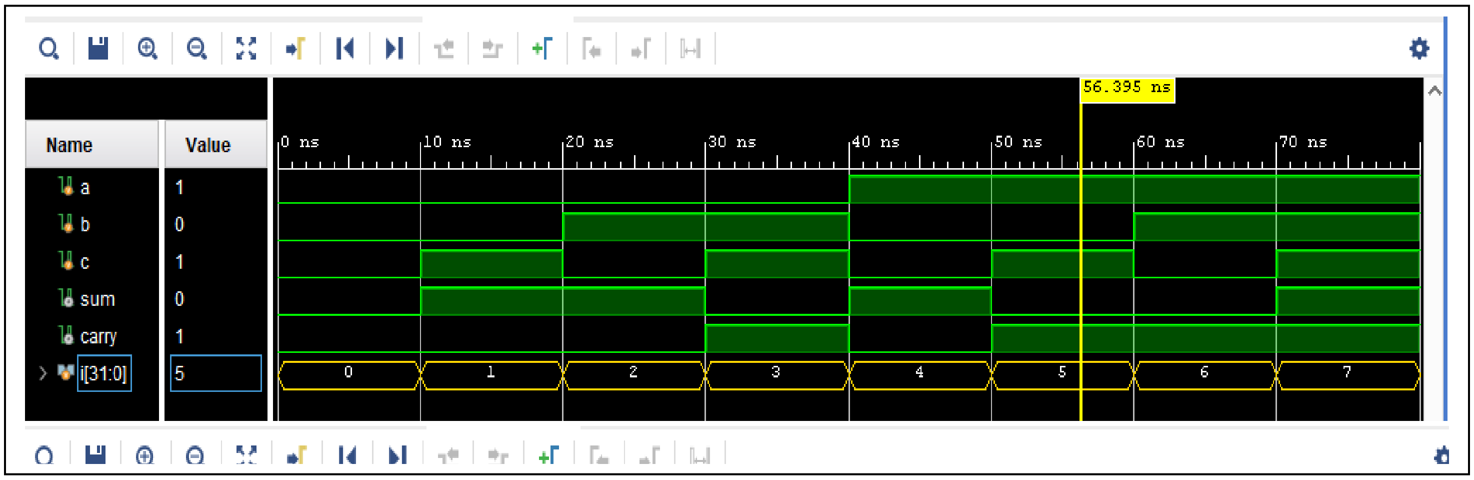
Name	Value
a	0
b	1
c	0
sum	1
carry	0
i[31:0]	2



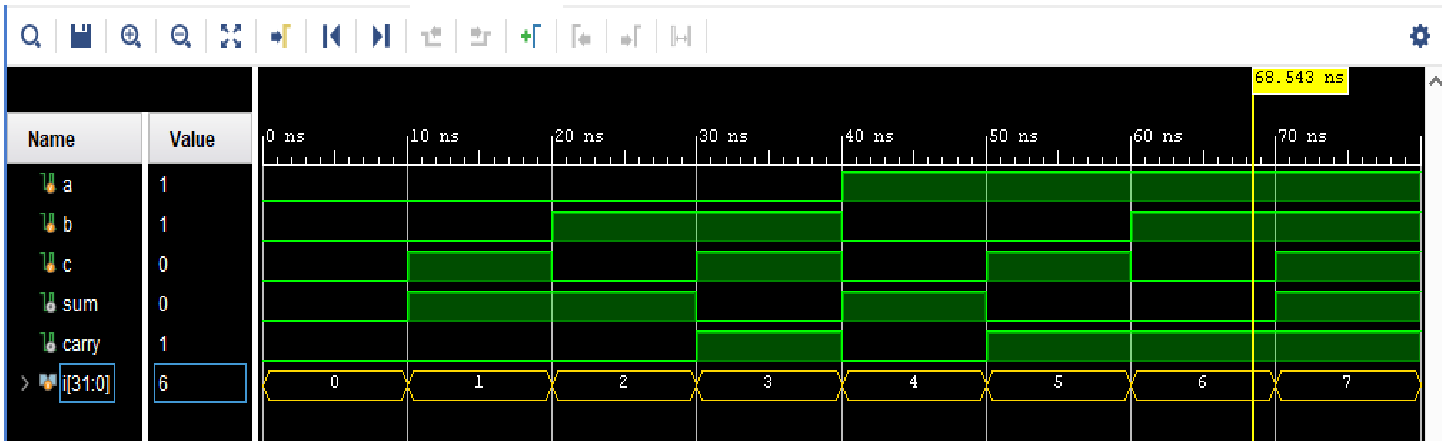
Name	Value
a	0
b	1
c	1
sum	0
carry	1
i[31:0]	3

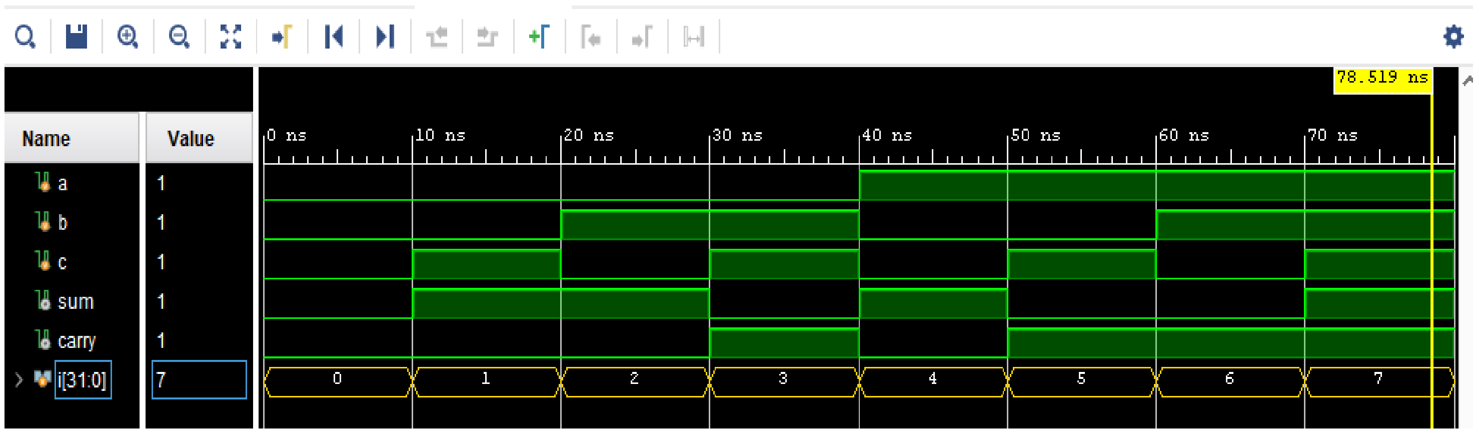


Name	Value
a	1
b	0
c	0
sum	1
carry	0
i[31:0]	4



Name	Value
a	1
b	0
c	1
sum	0
carry	1
i[31:0]	5





<u>Conclusion:</u> -

Hence , we have successfully designed and simulated Half-adder and Full-adder using gate-level modeling style in Verilog HDL. Also verified simulated waveforms with actual truth tables using Xilinx Vivado 2019.1