VLSI System Design

ELE301P

LAB - 1 - Report

Praveen B R

COE19B007

Submission Date: 25/08/2021

INDEX

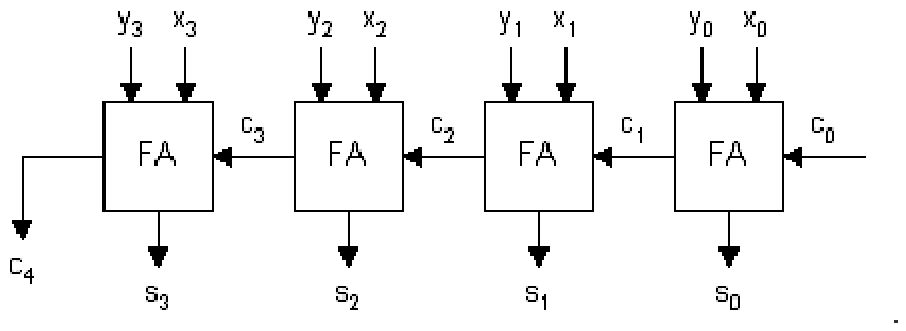
1. Q1) 4 Bit Ripple Carry Adder
    1.1. Code
    1.2. Results
2. Q2) Full Adder
    2.1. Dataflow model
    2.2. Structural model
    2.3. Behavioural model
3. Q3) Other Questions
    3.1. Question 1
    3.2. Question 2
    3.3. Question 3

Q1) 4 bit Ripple Carry Adder

Objective:
To implement a 4 bit Ripple Carry Adder using verilog in structural style

Theory:

Ripple Carry Adder is a combinational logic circuit. It is used for the purpose of adding two n-bit binary numbers. It requires n full adders in its circuit for adding two n-bit binary numbers. Each full adder takes the carry-in as input and produces carry-out and sum bit as output. It does not allow us to use all the full adders simultaneously.



Code:

Main Code

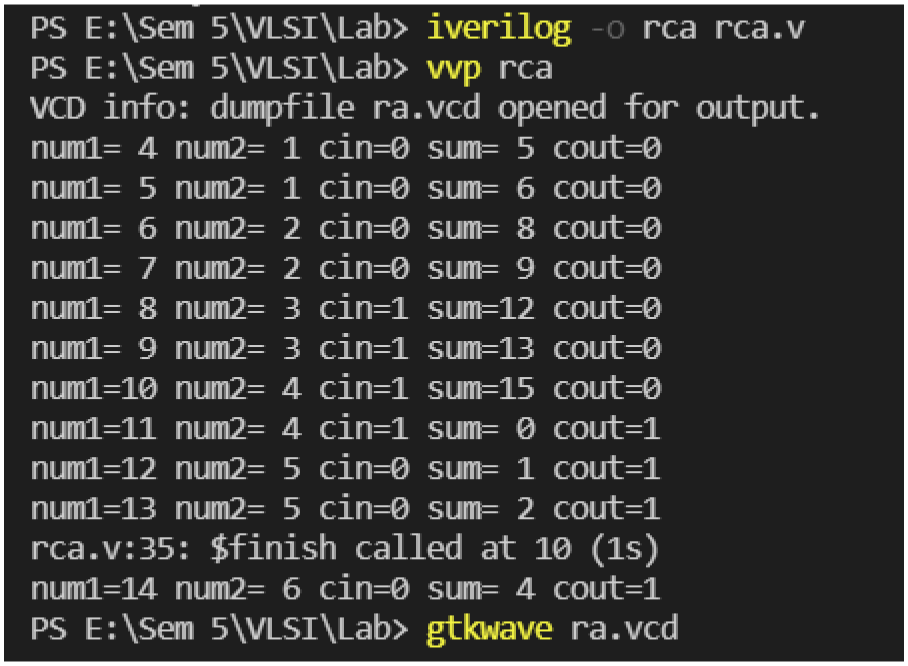
```
module full_adder(a,b,cin,sum,cout);

input a,b,cin;
output cout,sum;
wire s1,s2,s3;

xor(s1,a,b);
and(s2,a,b);
xor(sum,s1,cin);
and(s3,s1,cin);
or(cout,s3,s2);
endmodule

module ripple_carry_adder(a,b,cin,cout,sum);
input [3:0]a,b;
input cin;
output [3:0]sum;
output cout;
wire t[3:0];
full_adder f1(a[0],b[0],cin,sum[0],t[0]);
full_adder f2(a[1],b[1],t[0],sum[1],t[1]);
full_adder f3(a[2],b[2],t[1],sum[2],t[2]);
full_adder f4(a[3],b[3],t[2],sum[3],cout);
endmodule
```

Test bench

```
module rca_tb;
reg [3:0]a,b;
reg cin;
wire [3:0]sum;
wire cout;
ripple_carry_adder ra(a,b,cin,cout,sum);
initial begin
#0 a=4'b0100;b=4'b0001;cin=0;
#10 $finish;
end
initial begin
$monitor("num1=%d num2=%d cin=%b sum=%d cout=%b",a,b,cin,sum,cout);
end
initial begin
$dumpfile ("ra.vcd");
$dumpvars (0,ra);
end
always begin #1 a+=1;
end
always begin
#2 b+=1;
end
always begin
#4 cin+=1;
end
endmodule
```

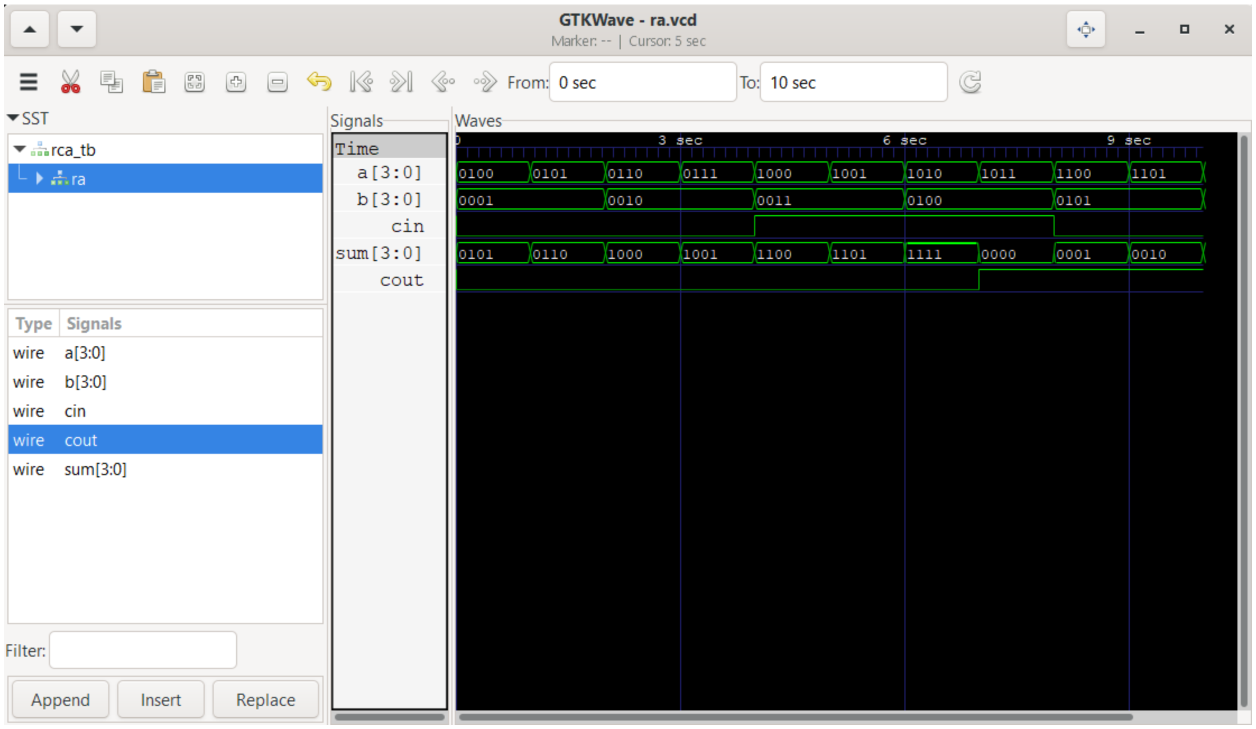
Results:

Terminal



```
PS E:\Sem 5\VLSI\Lab> iverilog -o rca rca.v
PS E:\Sem 5\VLSI\Lab> vvp rca
VCD info: dumpfile ra.vcd opened for output.
num1= 4 num2= 1 cin=0 sum= 5 cout=0
num1= 5 num2= 1 cin=0 sum= 6 cout=0
num1= 6 num2= 2 cin=0 sum= 8 cout=0
num1= 7 num2= 2 cin=0 sum= 9 cout=0
num1= 8 num2= 3 cin=1 sum=12 cout=0
num1= 9 num2= 3 cin=1 sum=13 cout=0
num1=10 num2= 4 cin=1 sum=15 cout=0
num1=11 num2= 4 cin=1 sum= 0 cout=1
num1=12 num2= 5 cin=0 sum= 1 cout=1
num1=13 num2= 5 cin=0 sum= 2 cout=1
rca.v:35: $finish called at 10 (1s)
num1=14 num2= 6 cin=0 sum= 4 cout=1
PS E:\Sem 5\VLSI\Lab> gtkwave ra.vcd
```

Waveform



Conclusion:
Thus, a 4 bit Ripple Carry Adder has been created using verilog in structural style.

Application:
4 bit Ripple Carry Adders can be used to add two 4 bit binary numbers.

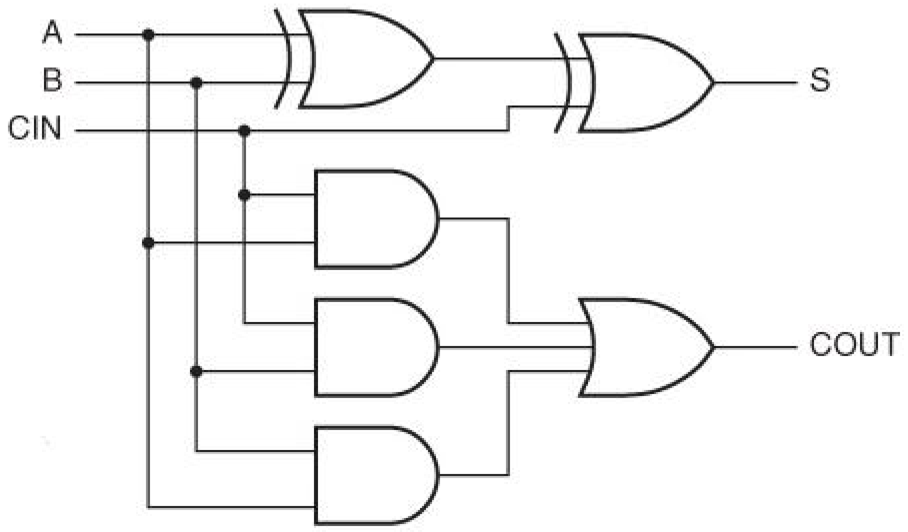
Q2) Full Adder using different styles

Objective:
To show simulation waveforms of different modellings of Full Adder

Theory:
A full adder circuit is central to most digital circuits that perform addition or subtraction. It is so called because it adds together two binary digits, plus a carry-in digit to produce a sum and carry-out digit. It therefore has three inputs and two outputs.The different models used in verilog are,

- Dataflow model
- Structural mode
- Behavioural model

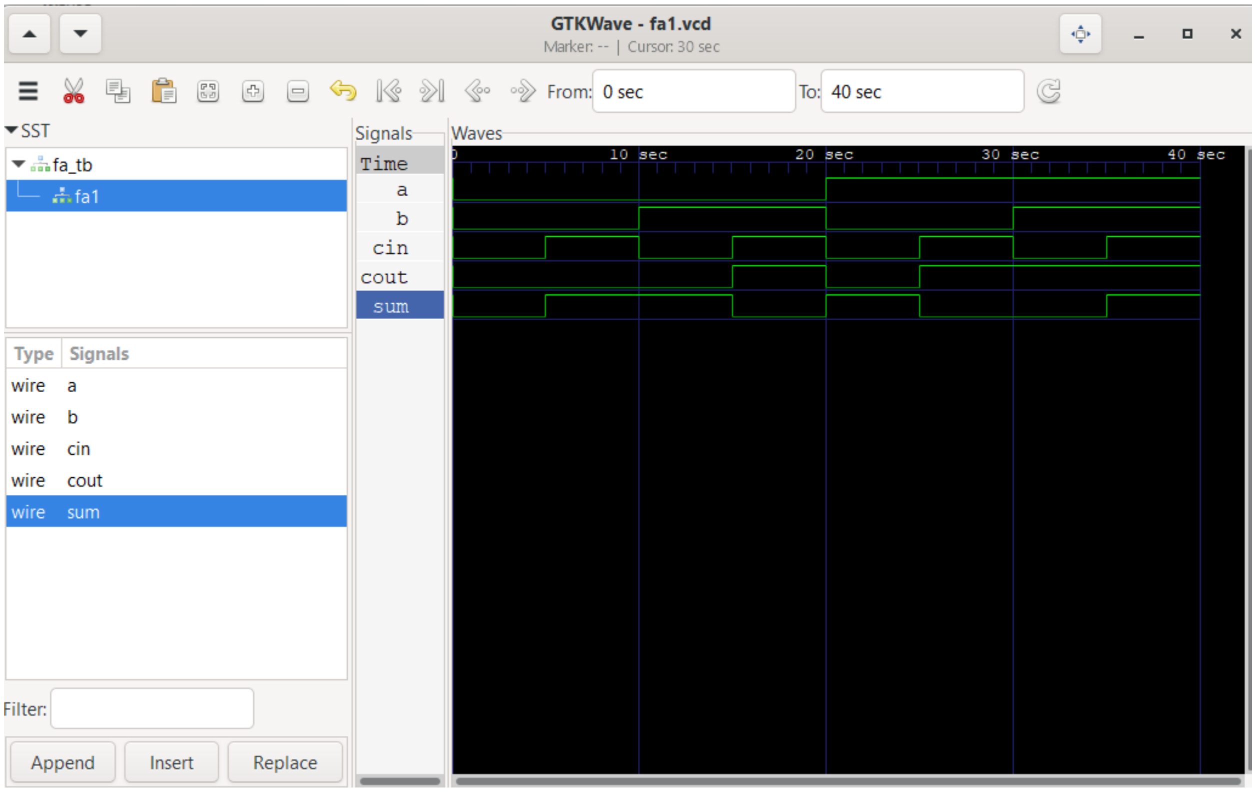
Dataflow model:



Code

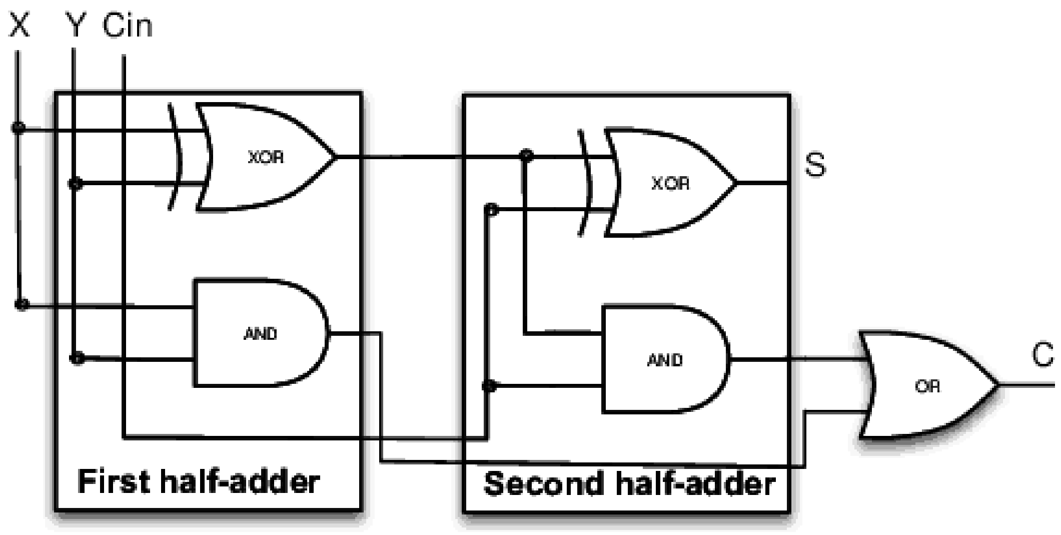
```
module fa_dataflow(a,b,cin,sum,cout);

input a,b,cin;
output cout,sum;

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

endmodule

module fa_tb();

reg ip1,ip2,ci;
wire s,co;

initial
begin
    ip1=0; ip2=0; ci=0; #5;
    ip1=0; ip2=0; ci=1; #5;
    ip1=0; ip2=1; ci=0; #5;
    ip1=0; ip2=1; ci=1; #5;
    ip1=1; ip2=0; ci=0; #5;
    ip1=1; ip2=0; ci=1; #5;
    ip1=1; ip2=1; ci=0; #5;
    ip1=1; ip2=1; ci=1; #5;
    $finish;
end

fa_dataflow fa1 (ip1,ip2,ci,s,co);

initial begin
    $dumpfile ("fa1.vcd");
    $dumpvars (0,fa1);
end

endmodule
```

Waveform



Type	Signals
wire	a
wire	b
wire	cin
wire	cout
wire	sum

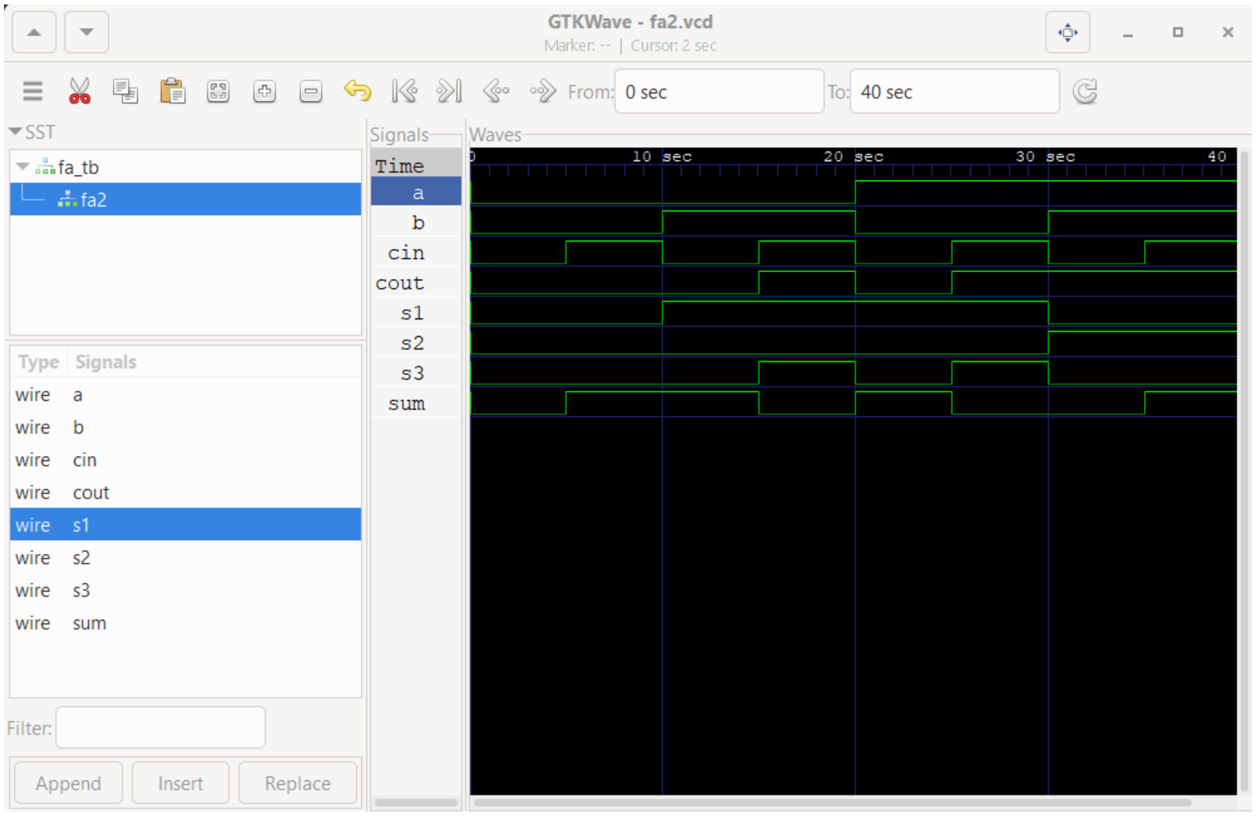
Structural model:



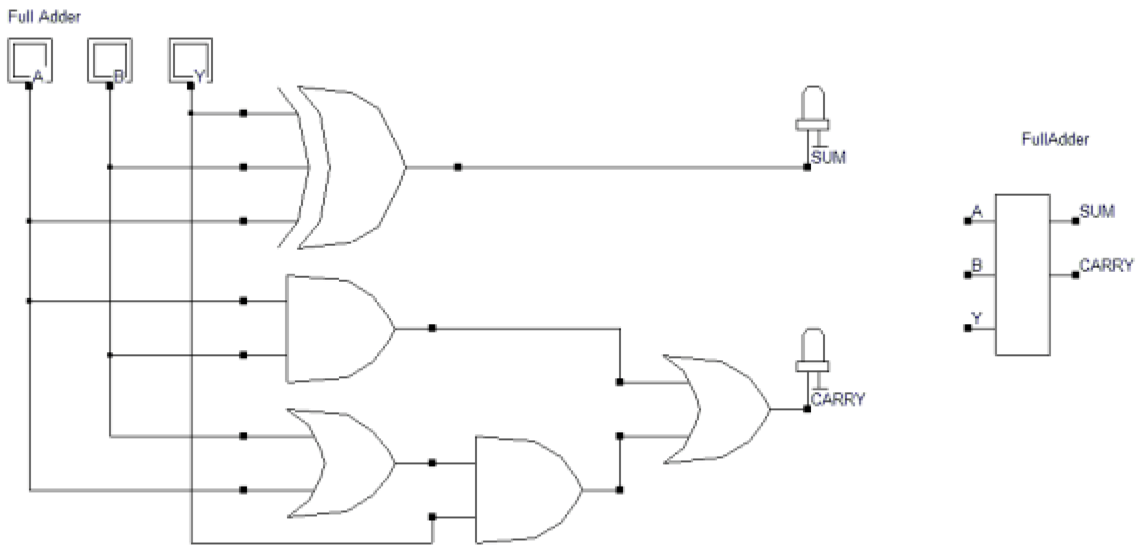
Code

```
module fa_structural(a,b,cin,sum,cout);

input a,b,cin;
output cout,sum;
wire s1,s2,s3;

xor (s1,a,b);
and (s2,a,b);
xor (sum,s1,cin);
and (s3,s1,cin);
or (cout,s3,s2);
endmodule

module fa_tb();

reg ip1,ip2,ci;
wire s,co;

initial
begin
    ip1=0; ip2=0; ci=0; #5;
    ip1=0; ip2=0; ci=1; #5;
    ip1=0; ip2=1; ci=0; #5;
    ip1=0; ip2=1; ci=1; #5;
    ip1=1; ip2=0; ci=0; #5;
    ip1=1; ip2=0; ci=1; #5;
    ip1=1; ip2=1; ci=0; #5;
    ip1=1; ip2=1; ci=1; #5;
    $finish;
end

fa_structural fa2 (ip1,ip2,ci,s,co);
initial begin
    $dumpfile ("fa2.vcd");
    $dumpvars (0,fa2);
end

endmodule
```

Waveform



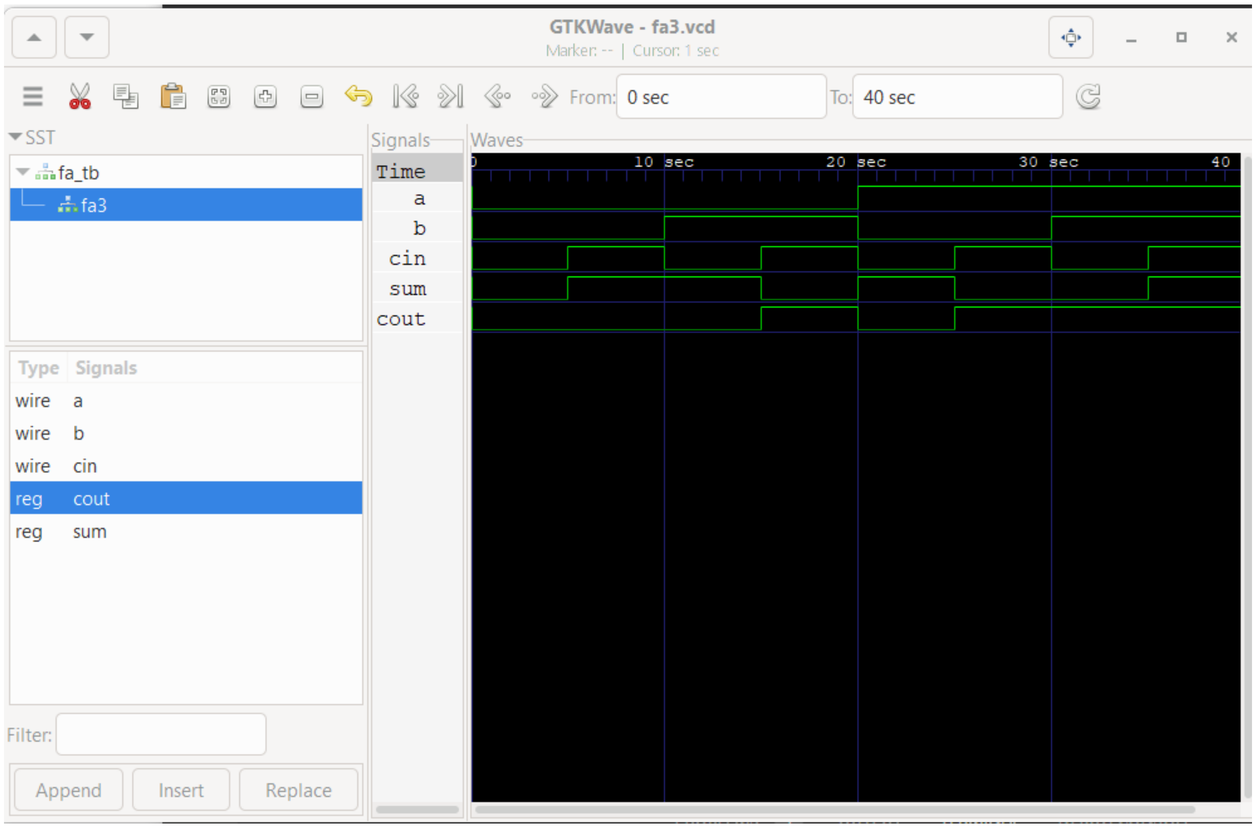
Behavioural model:



Code

```
module fa_behavioural(a,b,cin,sum,cout);

input a,b,cin;
output reg cout,sum;
always @(a,b,cin)
begin
    cout =(a&b|((a^b)&cin));
    sum = a^b^cin;
end
endmodule

module fa_tb();

reg ip1,ip2,ci;
wire s,co;

initial
begin
    ip1=0; ip2=0; ci=0; #5;
    ip1=0; ip2=0; ci=1; #5;
    ip1=0; ip2=1; ci=0; #5;
    ip1=0; ip2=1; ci=1; #5;
    ip1=1; ip2=0; ci=0; #5;
    ip1=1; ip2=0; ci=1; #5;
    ip1=1; ip2=1; ci=0; #5;
    ip1=1; ip2=1; ci=1; #5;
    $finish;
end

fa_behavioural fa3 (ip1,ip2,ci,s,co);

initial begin
    $dumpfile ("fa3.vcd");
    $dumpvars (0,fa3);
end

endmodule
```

Waveform



Type	Signals
wire	a
wire	b
wire	cin
reg	cout
reg	sum

<u>Conclusions:</u>
Thus, the different simulation waveforms of a full adder are shown.

Q3) Other Questions

Question 1:
Are there any statements in Verilog similar to "printf" in C to print the output or values of variables in the terminal? If yes, mention their syntax.

Answer:
Yes, there are functions in verilog similar to "printf" in C. They are,

Function	Syntax
Monitor	$monitor (<arguments>);
Display	$display (<arguments>);
Write	$write (<arguments>);
Strobe	$strobe (<arguments>);

Display: display the values and ends with new line
Monitor: It will call the display function whenever there is change in argument.
Write: display the values of arguments and does not end with a new line.
Strobe: it is similar to display but prints the values at the end of current time.

Question 2:
Suppose a is a 5-bit wire with a value 5'b11001 and b is a 4-bit wire. If we write "**assign b=a**" what will be the output?

Answer:
When the "**assign b=a**" is performed the left most bit or the so called most significant bit of **a** is dismissed and the next 4 bits will be assigned to **b**.And the value of **b** will be **b = 4'b1001**.

Question 3:
Suppose a=3'b101, b=4'b1010. Now I need c which is the concatenation of a and b i.e c=7'b1011010. Find the syntax to do the same.

Answer
Concatenation of a,b into c can be done using the syntax,
Syntax : **c={a,b}**
