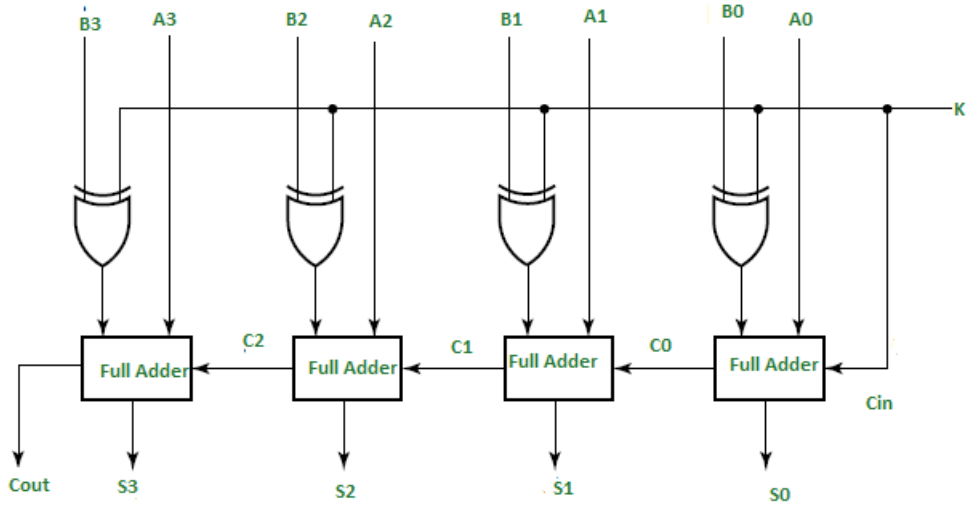
Experiment 2

4 BIT ADDER

A binary adder is a digital circuit which adds 2 or more numbers. The inputs and outputs are represented in binary form. Typically, an n-bit adder performs addition using two n-bit inputs along with a carry input and returns an n+1 bit output where the $n+1^{th}$ bit is carry-out.

A 4-bit adder has two 4-bit inputs and a carry in. The circuit can accept inputs ranging from 0000 to 1111 while the carry input can take the values of 0 or 1. The circuit can be realised using four full adders as shown below.



4-Bit adder using full adder

Each full adder performs addition of two single bit inputs and carry forward from the previous stage. The gate level realization of a full adder is as below.

A	B	C in	Sum	C out
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

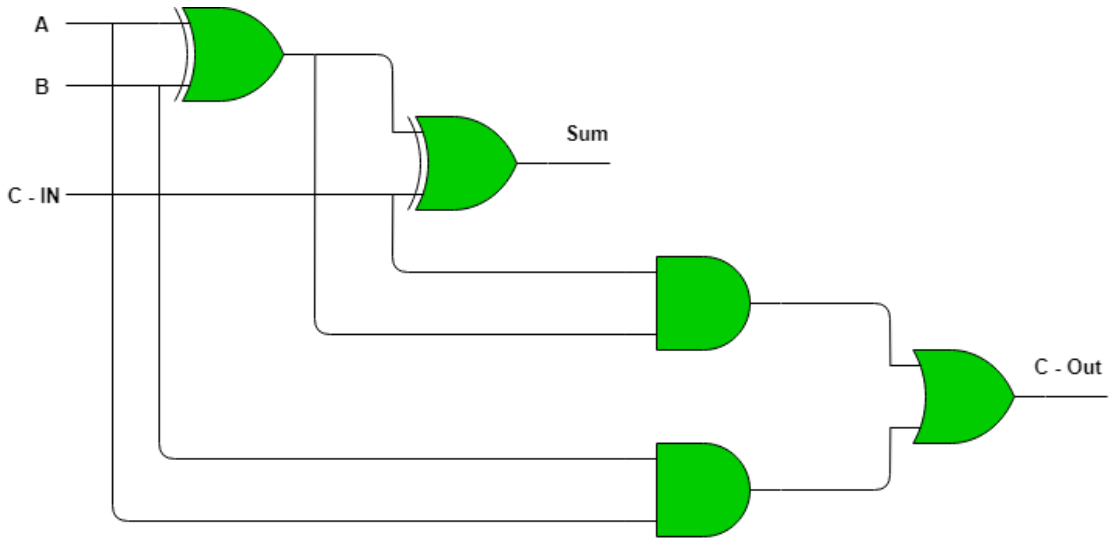
Truth table for full adder

A, B \ Cin	0	1
0, 0	0	1
0, 1	1	0
1, 1	0	1
1, 0	1	0

K Map for Sum

A, B \ Cin	0	1
0, 0	0	0
0, 1	0	1
1, 1	1	1
1, 0	0	1

K Map for Carry out



Full adder

Sum = A ⊕ B ⊕ C in

Carry = AB + BC + CA

As we know the module full adder is being used multiple times in the circuit, we first create a full adder module in Verilog using behavioural, dataflow and structural forms and then use those full adder modules to create the 4-bit adder.

Behavioural Modeling

The behavioural modelling of the circuit consists of the following steps

1. Module instantiation
2. Defining inputs and outputs
3. Finding input-output relation
4. Assigning outputs as direct functions of inputs

```
module adder(
    input a,
    input b,
    input cin,
    output reg sum,
    output reg cout
    );
```

Module instantiation with input and output defining

```
always@*
begin
if ({a, b, cin} == 3'b000) begin
sum = 0;    cout = 0;
end

else if ({a, b, cin} == 3'b001 || {a, b, cin} == 3'b010 || {a, b, cin} == 3'b100) begin
sum = 1;    cout = 0;
end

else if ({a, b, cin} == 3'b011 || {a, b, cin} == 3'b101 || {a, b, cin} == 3'b110 ) begin
sum = 0;    cout = 1;
end

else if ({a, b, cin} == 3'b111) begin
sum = 1;    cout = 1;
end

end
```

Assigning input output relations using conditional statements

We get the conditional statements by analysing the behaviour of outputs with each input combinations. We have defined a single module with the above codes. Now we need to connect four such full adders to realize a 4-bit adder.

```
module adderbehav(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output [3:0] Sum,
    output Cout);

wire c1,c2,c3;

adder a0(A[0],B[0],Cin,Sum[0],c1);
adder a1(A[1],B[1],c1,Sum[1],c2);
adder a2(A[2],B[2],c2,Sum[2],c3);
adder a3(A[3],B[3],c3,Sum[3],Cout);

endmodule
```

We initialise a high-level module with full adders as lower-level modules and connect those modules to realize 4-bit adder. The two 4-bit inputs and a 4-bit output is initialized as 4-bit ports. The bit wise wiring is done between the full adders.

Dataflow Modeling

The dataflow modelling of the circuits consists of the following steps

1. Module instantiation
2. Defining inputs and outputs
3. Finding operations being performed
4. Applying operation into inputs to get output

```
1  module full_adder(
2       input a,b,cin,
3       output sum,carry);
4
```

Module and input instantiation

```
4
5  assign sum = a ^ b ^ cin;
6  assign carry = (a & b)|(b & cin)|(cin & a);
7
```

Assigning input output relation with various operations

```
9
10 module rippleca(
11     input [3:0]a,
12     input [3:0]b,
13     input cin,
14     output [3:0] sum,
15     output c4);
16
17 wire c1,c2,c3;
18
19 full_adder fa0(a[0],b[0],cin,sum[0],c1);
20 full_adder fa1(a[1],b[1],c1,sum[1],c2);
21 full_adder fa2(a[2],b[2],c2,sum[2],c3);
22 full_adder fa3(a[3],b[3],c3,sum[3],c4);
23
24 endmodule
```

Module and inputs are initialized. The higher order module is realized using full adders as lower order modules. Each lower order modules are wired internally.

Structural Modeling

The structural modelling of the circuits consists of the following steps

1. Module instantiation
2. Defining inputs and outputs
3. Finding logical operations performed
4. Realizing logical operations using logical components

These modelling may contain a single module or multiple modules which maybe custom or commercially available standard modules.

```
2
3  module adder(
4      input A,
5      input B,
6      input Cin,
7      output S,
8      output Cout
9      );
10
```

Module and input instantiation

```
10
11 wire w1, w2, w3;
12
13 xor xor_1(S, A, B, Cin);
14 and and_1(w1, A, B);
15 and and_2(w2, A, Cin);
16 and and_3(w3, B, Cin);
17 or or_1(Cout, w1, w2, w3);
18
```

Realisation of module functions using logic components

The components are wired using wires. The gate functions are pre-defined in Verilog.

```
20
21 module struct(
22      input [3:0]a,
23      input [3:0]b,
24      input cin,
25      output [3:0] sum,
26      output cout);
27
```

Module and input instantiation for higher order module

```
27
28 wire c1, c2, c3;
29
30 adder a0(a[0], b[0], cin, sum[0], c1);
31 adder a1(a[1], b[1], c1, sum[1], c2);
32 adder a2(a[2], b[2], c2, sum[2], c3);
33 adder a3(a[3], b[3], c3, sum[3], cout);
34
35 endmodule
```

Higher order module realization

The circuit models are then tested with a testbench as simulation source which contains test cases taking different values for inputs. Testbench contains module instantiation and event instantiation.

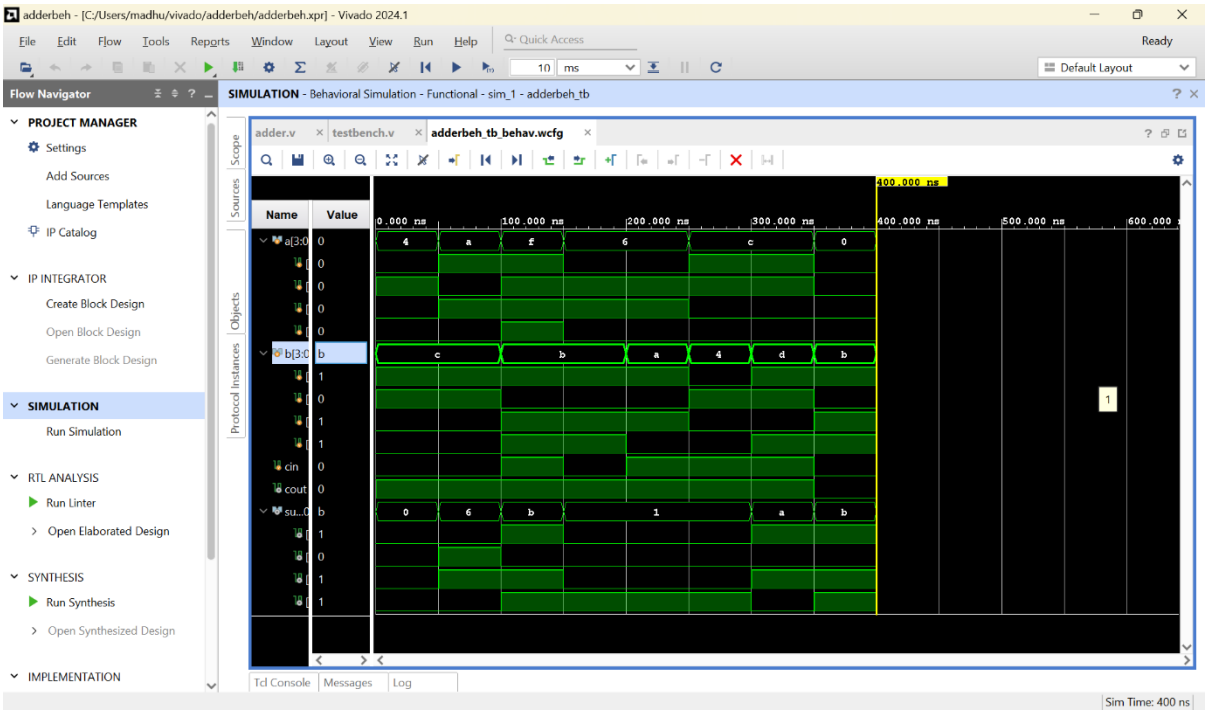
```
2
3  module adderstruct_tb;
4
5  reg [3:0] a, b;
6  reg cin;
7  wire [3:0] sum;
8  wire cout;
9
10 struct testbench(a, b, cin, sum, cout);
11
```

Module and input, output instantiation

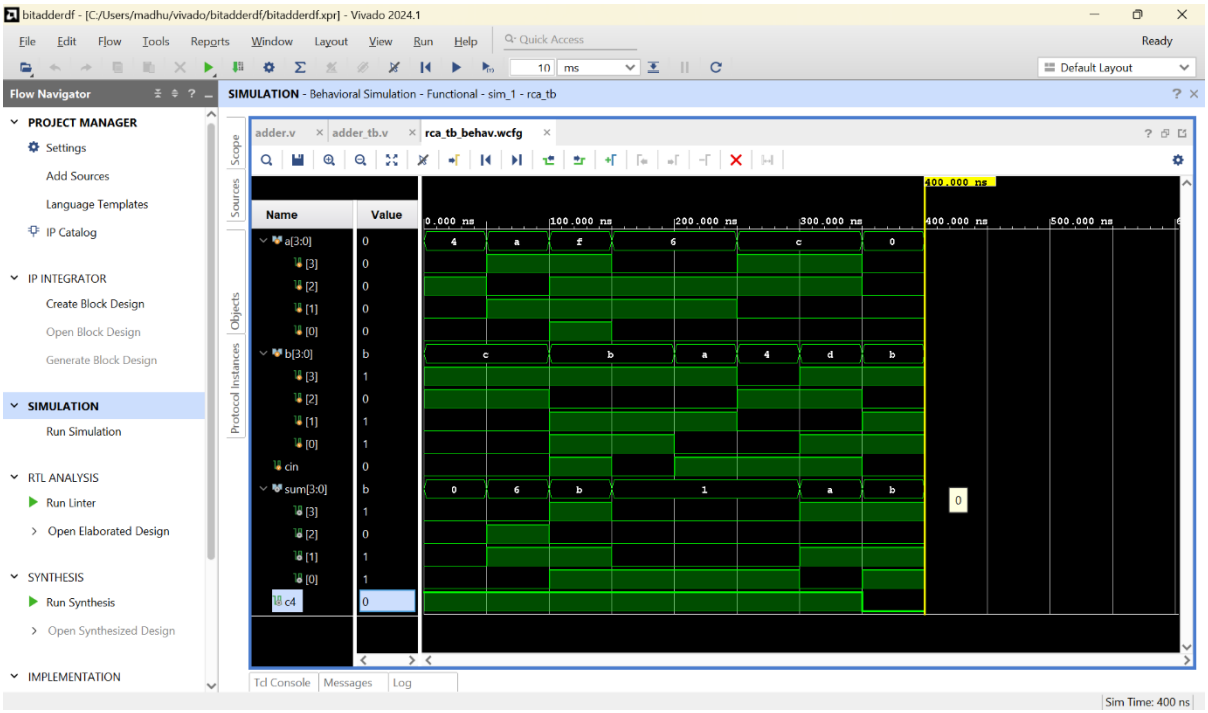
```
11
12 initial
13 begin
14
15 a = 4'b0100;    b = 4'b1100;    cin = 1'b0;    #50
16 a = 4'b1010;    b = 4'b1100;    cin = 1'b0;    #50
17 a = 4'b1111;    b = 4'b1011;    cin = 1'b1;    #50
18 a = 4'b0110;    b = 4'b1011;    cin = 1'b0;    #50
19 a = 4'b0110;    b = 4'b1010;    cin = 1'b1;    #50
20 a = 4'b1100;    b = 4'b0100;    cin = 1'b1;    #50
21 a = 4'b1100;    b = 4'b1101;    cin = 1'b1;    #50
22 a = 4'b0000;    b = 4'b1011;    cin = 1'b0;    #150
23
24 $finish();
25
26 end
```

Sequential events

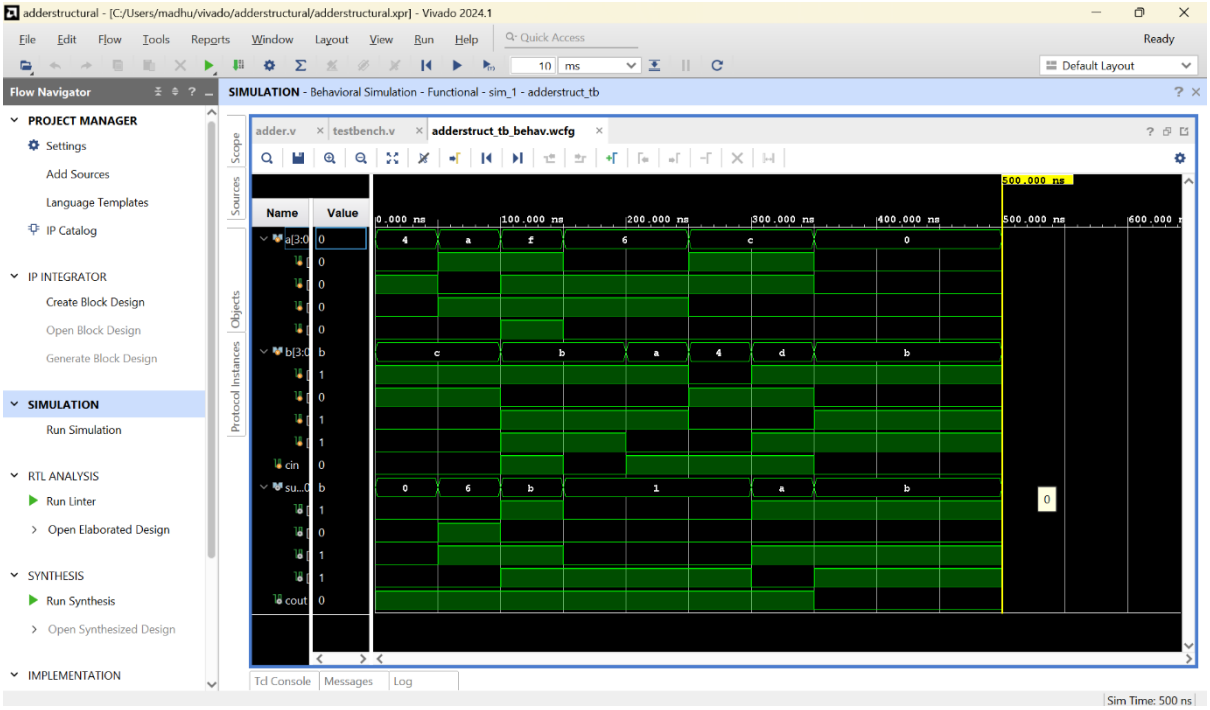
These sequential events push inputs to certain values being provided in the event assignment. Behavioural simulation output is realized using these inputs which can be viewed in the form of graph.



Behavioural Modeling output



Dataflow Modeling output



Structural Modeling output