Implement a 4-bit ripple carry adder using full adders.

CODE:

```
`timescale 1ns / 1ps

module ripple_carry_adder_4bit(
input [3:0] a, b,
input cin,
output [3:0] sum,
output carry);
wire c1, c2, c3;
full_adder uut1(a[0],b[0],cin, sum[0], c1);
full_adder uut2(a[1], b[1], c1, sum[1], c2);
full_adder uut3(a[2],b[2], c2, sum[2], c3);
full_adder uut4(a[3], b[3], c3, sum[3], carry);
endmodule
```

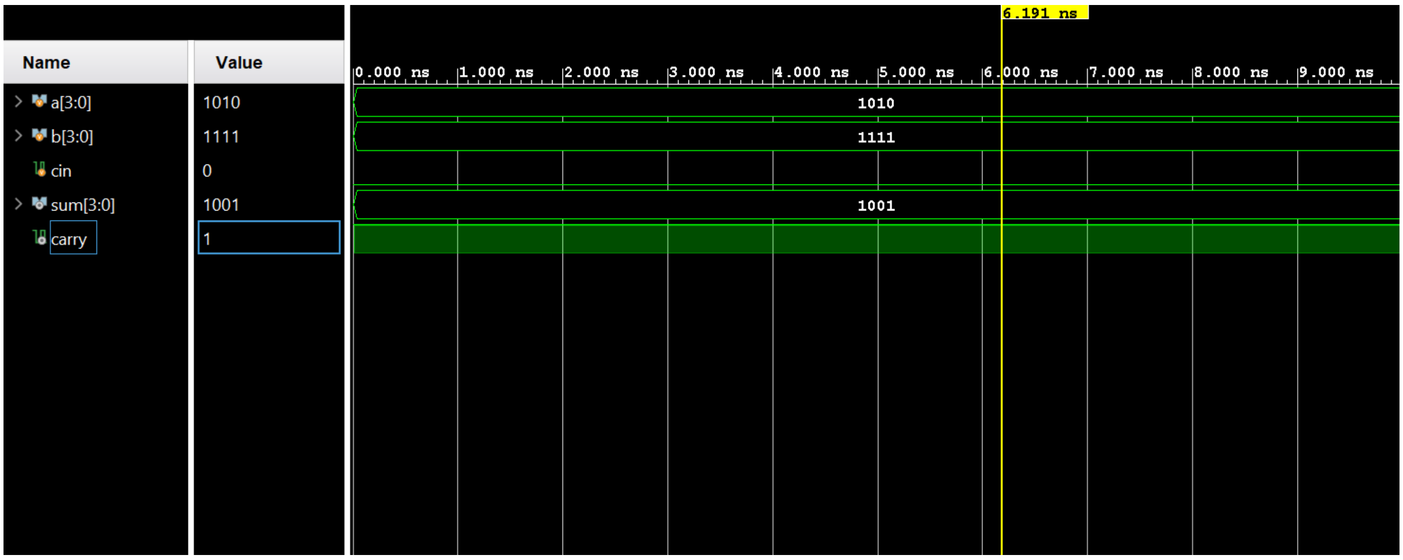
#full_adder

```
`timescale 1ns / 1ps

module full_adder(
input a,b,cin,
output s,cout
    );
assign s = a ^ b ^ cin;
assign cout = (a&b)|(b&cin)|(a&cin);
endmodule
```

TESTBENCH:

```
`timescale 1ns / 1ps


module ripple_carry_adder_tb();

reg [3:0]a,b;
reg cin;
wire [3:0]sum;
wire carry;

ripple_carry_adder_4bit uut(a,b,cin,sum,carry);

initial begin
    a=4'b1010; b=4'b1111; cin=0; #10
    $stop;
end
endmodule
```

SIMULATION RESULT:



Name	Value
a[3:0]	1010
b[3:0]	1111
cin	0
sum[3:0]	1001
carry	1

ERROR FACED:

```
module ripple_carry_adder_4bit(
input [3:0] a, b,
input cin,
output [3:0] sum,
output carry);
```

Instead of output [3:0] sum I was using output reg [3:0] sum which is wrong because sum is driven by combinational logic and not a procedural always block.

Key Differences: wire vs. reg

Type	When to Use?	Driven By?
wire	Used for combinational logic, driven by assign statements or module outputs.	Continuous assignments or module connections.
reg	Used for storage elements, driven inside an always block.	Procedural assignments (always block).