V. Evaluate my Learnings

Name: REIMARC G. CORPUZ Score: _____

Course/Sec: BSCPE 3GF Time/Day: 6:11/Monday

Date Sub: NOV. 15, 2022

A. Answer the following questions accordingly.

1. What are the basic components of a module? Which components are mandatory?

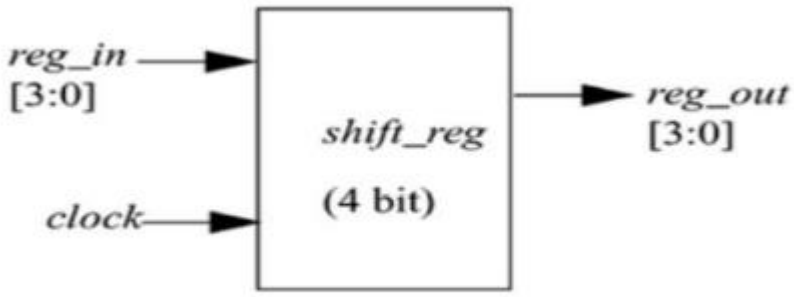
The basic components of a module are Module Name, Port List, Port Declarations, Parameters, Declarations of wires, regs and other variables, Data flow statements, Instantiation of lower-level modules, always and initial blocks, Tasks and functions, endmodule statement.

The components that are mandatory are module, module name, and endmodule.

2. Does a module that does not interact with its environment have any I/O ports? Does it have a port list in the module definition?

The module that does not interact with its environment have I/O ports. And it doesn't have a port list in the module definition.

3. A 4-bit parallel shift register has I/O pins as shown in the figure below. Write the module definition for this module shift_reg. Include the list of ports and port declarations. You do not need




```
module shift_reg(reg_out,reg_in,clock);
    output reg [3:0]reg_out;
    input [3:0]reg_in;
    input clock;
endmodule
```

4. Declare a top-level module stimulus. Define REG_IN (4 bit) and CLK (1 bit) as reg register variables and REG_OUT (4 bit) as wire. Instantiate the module shift_reg and call it sr1. Connect the ports by ordered list.

```
module stimulus;
        reg [3:0]REG_IN;
        reg CLK;
        wire [3:0]REG_OUT;
        shift_reg sr1(REG_OUT,REG_IN,CLK);
endmodule
```

5. Connect the ports in Step 4 by name.

```
shift_reg(.reg_out(REG_OUT),.reg_in(REG_IN),.clock(CLK));
```

6: Write the hierarchical names for variables REG_IN, CLK, and REG_OUT.

stimulus.REG_IN, stimulus.CLK, stimulus.REG_OUT

7: Write the hierarchical name for the instance sr1. Write the hierarchical names for its ports clock and reg_in.

stimulus.sr1,

stimulus.sr1.clock

stimulus.sr1.reg_in

B. Write the structural modeling of Full adder here module name YOURINITIALS_fa (CMCC_fa):

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

        input a,b,cin;
        output sum,cout;
        wire t1, t2, k;
```

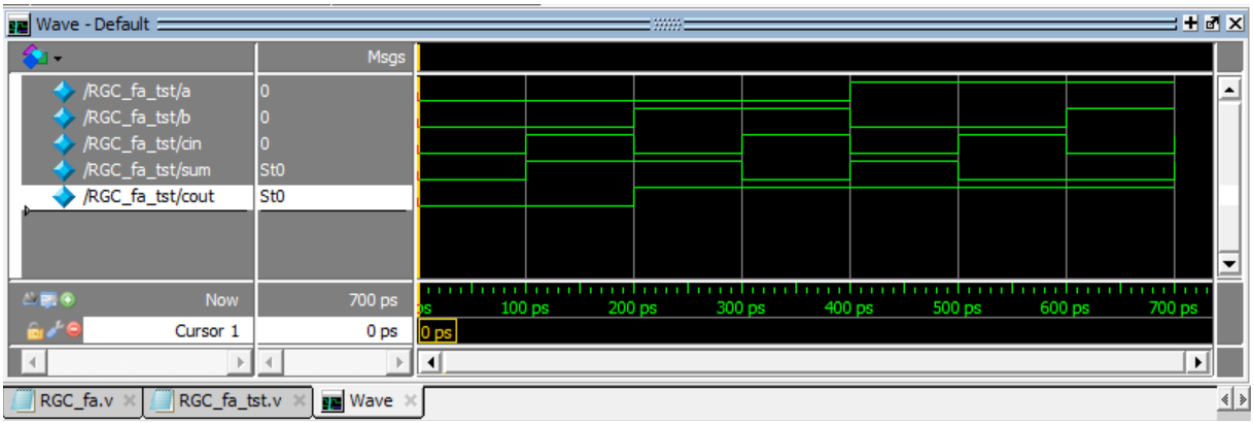
```
    xor(sum,a,b,cin);
    xor(k,a,b);
    and(t1,a,b);
    and(t2,k,cin);
    or(cout,t1,t2,k);

endmodule
```

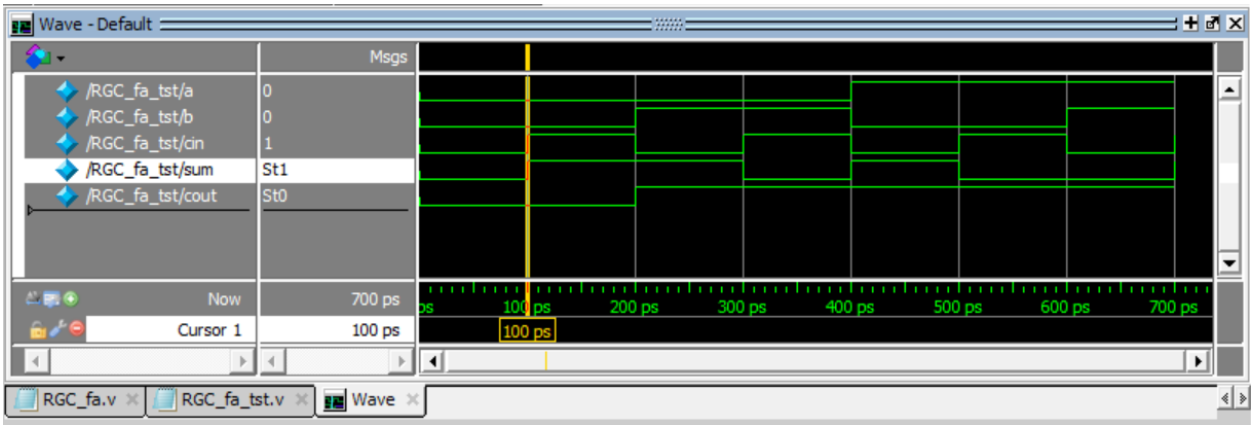
C. An example of a test bench created for the 1 bit full adder is shown below. Modify it and Draw the expected waveform using the test bench.

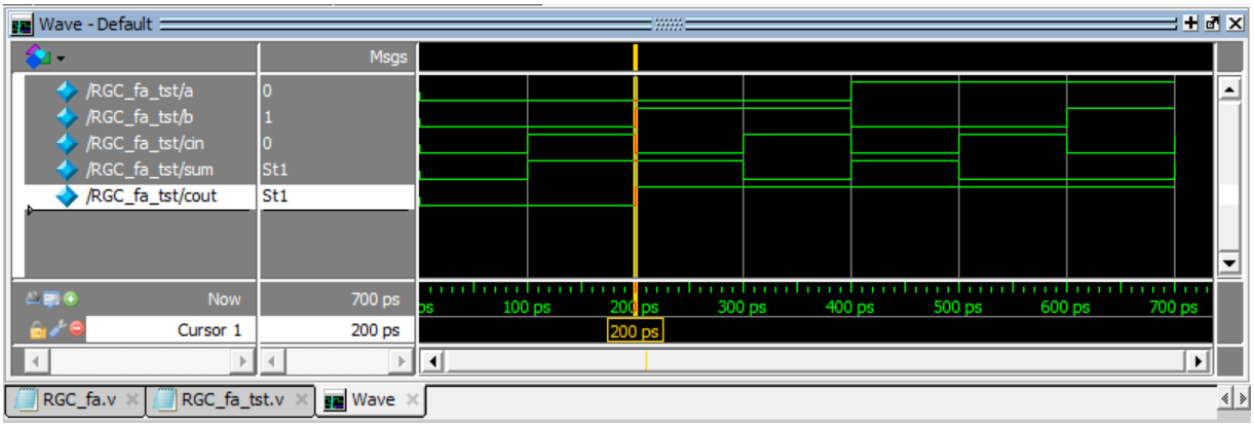
```
module RGC_fa_tst();
reg a, b, cin;
wire sum, cout;

RGC_fa uut ( .a(a), .b(b), .cin(cin), .sum(sum), .cout(cout));

initial begin
    a = 0;
    b = 0;
    cin = 0;
#100 a = 0; b = 0; cin = 1;
#100 a = 0; b = 1; cin = 0;
#100 a = 0; b = 1; cin = 1;
#100 a = 1; b = 0; cin = 0;
#100 a = 1; b = 0; cin = 1;
#100 a = 1; b = 1; cin = 0;
#100 a = 1; b = 1; cin = 1;
```

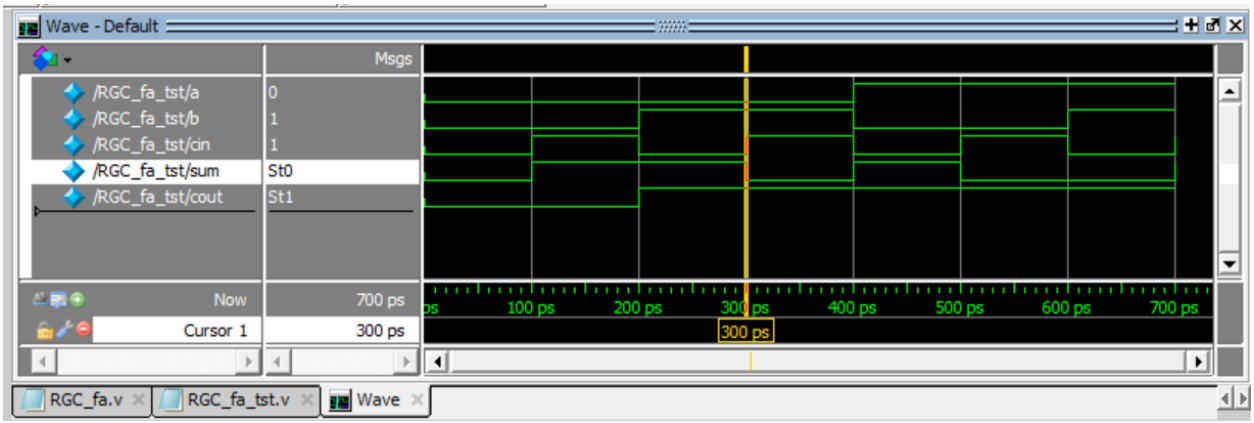
```
end
endmodule
```

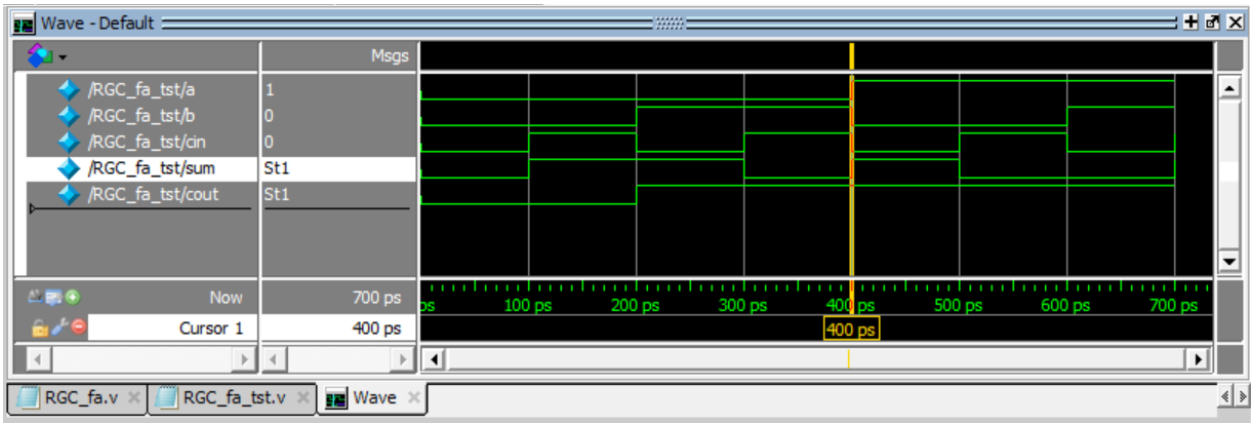
For letter C...Draw the expected waveform here:

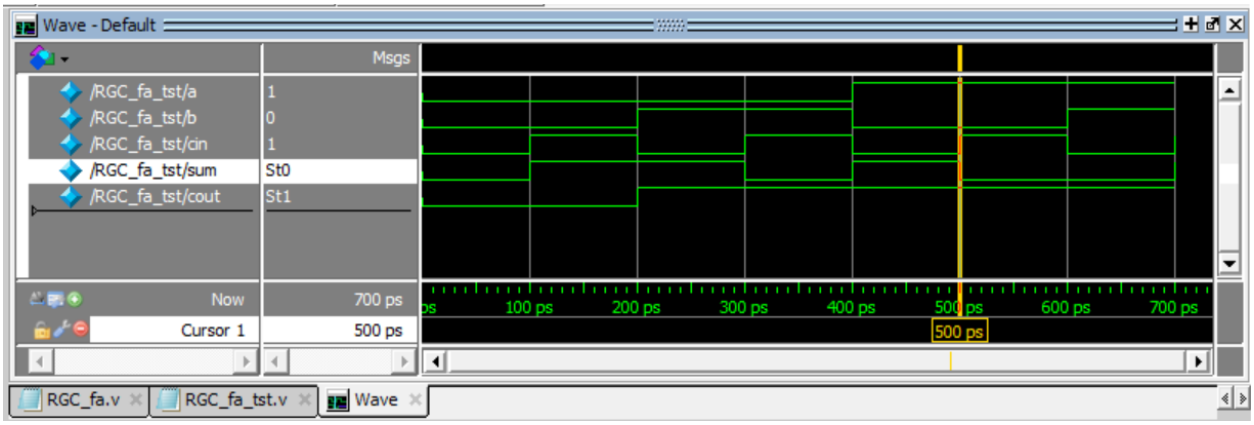


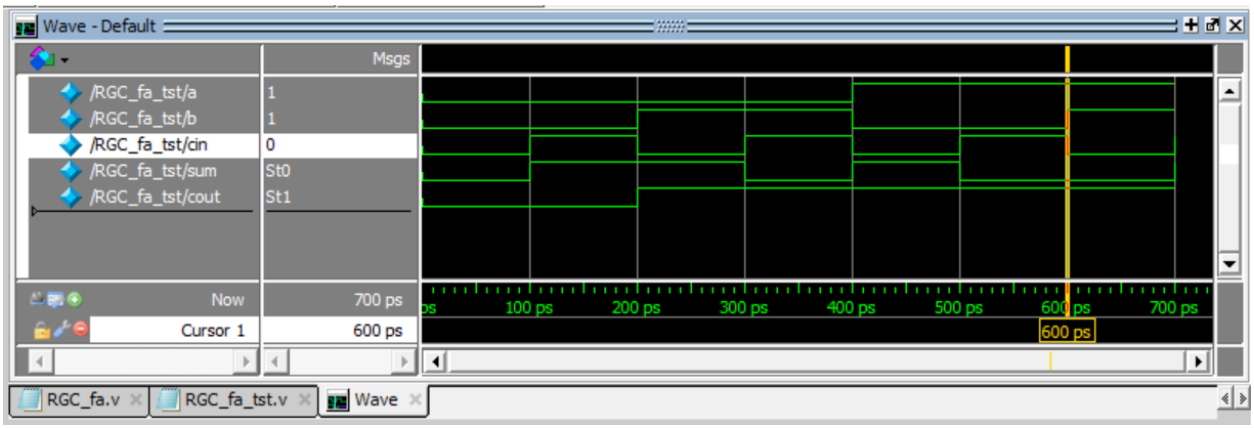


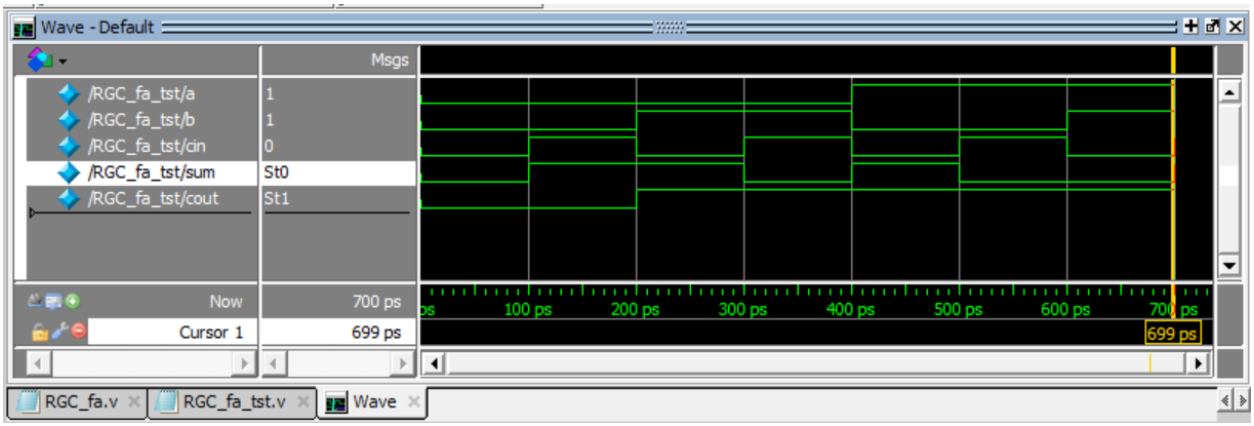



Wave - Default
Msgs
/RGC_fa_tst/a
/RGC_fa_tst/b
/RGC_fa_tst/cin
/RGC_fa_tst/sum
/RGC_fa_tst/cout
0
1
1
St0
St1
Now
700 ps
Cursor 1
300 ps
100 ps
200 ps
300 ps
400 ps
500 ps
600 ps
700 ps
RGC_fa.v
RGC_fa_tst.v
Wave


Wave - Default
Msgs
/RGC_fa_tst/a
/RGC_fa_tst/b
/RGC_fa_tst/cin
/RGC_fa_tst/sum
/RGC_fa_tst/cout
1
0
0
St1
St1
Now
700 ps
Cursor 1
400 ps
100 ps
200 ps
300 ps
400 ps
500 ps
600 ps
700 ps
RGC_fa.v
RGC_fa_tst.v
Wave


Wave - Default
Msgs
/RGC_fa_tst/a
/RGC_fa_tst/b
/RGC_fa_tst/cin
/RGC_fa_tst/sum
/RGC_fa_tst/cout
1
0
1
St0
St1
Now
700 ps
Cursor 1
500 ps
100 ps
200 ps
300 ps
400 ps
500 ps
600 ps
700 ps
RGC_fa.v
RGC_fa_tst.v
Wave


Wave - Default
Msgs
/RGC_fa_tst/a
/RGC_fa_tst/b
/RGC_fa_tst/cin
/RGC_fa_tst/sum
/RGC_fa_tst/cout
1
1
0
St0
St1
Now
700 ps
Cursor 1
600 ps
100 ps
200 ps
300 ps
400 ps
500 ps
600 ps
700 ps
600 ps
RGC_fa.v
RGC_fa_tst.v
Wave


Wave - Default
Msgs
/RGC_fa_tst/a
/RGC_fa_tst/b
/RGC_fa_tst/cin
/RGC_fa_tst/sum
/RGC_fa_tst/cout
1
1
0
St0
St1
Now
700 ps
Cursor 1
699 ps
100 ps
200 ps
300 ps
400 ps
500 ps
600 ps
699 ps
RGC_fa.v
RGC_fa_tst.v
Wave

D. Test the circuit by adding the following 4-bit binary values. (Negative numbers are given in two's complement notation. Verify results by converting sums back to their base-10 equivalent values.

		bay	dish
3+2=5	0011+0010=	_____	_____
5+4=9	0101+0100=	_____	_____
7−3=4	0111+1101=	_____	_____
3−5=−2	0011+1011=	_____	_____

Draw the expected waveform:

D)

		BINARY		DECIMAL
3 + 2 = 5	→	0011 + 0101 =	0101	→ 5
5 + 4 = 9	→	0101 + 0100 =	1001	→ 9
7 − 3 = 4	→	0111 + 1101 =	0100	→ 4
3 − 5 = −2	→	0011 + 1011 =	1110	→ −2

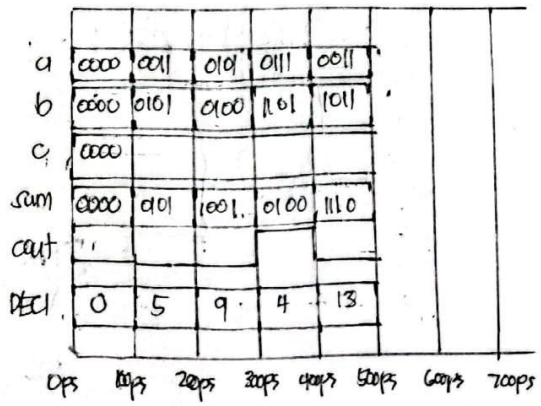
VERILOG STRUCTURE:

```
module fullADDER_4bit(sum,cout,a,b,c);
  input [3:0] a,b; input c;
  output wire [3:0] sum;
  output cout;
  FULLadder FA1(a[0],b[0],c,sum[0],cout1);
  FULLadder FA2(a[1],b[1],cout1,sum[1],cout2);
  FULLadder FA3(c[2],b[2],cout2,sum[2],cout3);
  FULLadder FA4(a[3],b[3],cout3,sum[3],c);
endmodule

module FULLadder(sum,cout,a,b,c);
  input a,b,c; output sum,cout;
  wire t1,t2,t3;
/*always @(DEC)
  begin
    case(DEC1)
      0: sum = 4'b0000;
      1: sum = 4'b0001;
      2: sum = 4'b0010;
      3: sum = 4'b0011;
      4: sum = 4'b0100;
      5: sum = 4'b0101;
      6: sum = 4'b0110;
      7: sum = 4'b0111;
      8: sum = 4'b1000;
      .
      .
      .
      15: sum = 4'1111;
  end*/
  XOR (sum, a,b,c);
  XOR (t1, a,b);
  AND (t2, a,b);
  AND (t3, t1, c);
  OR (cout, t1, t2, t3);
endmodule
```

TESTBENCH

```
module fullADDER_4bit_tst();
  reg [3:0] a,b,c; // DEC1
  wire [3:0] sum, cout;
module fullADDER_4bit(
    .sum(sum),
    .cout(cout),
    .a(a),
    .b(b),
    .c(c),
initial //.DEC1(c));
begin
  a = 4'b0000; b = 4'b0000; c = 0;
#100 a = 4'b0011; b = 4'b0101;
#100 a = 4'b0101; b = 4'b0100;
#100 a = 4'b0111; b = 4'b1101;
#100 a = 4'b0011; b = 4'b1011;
 end
endmodule
```

EXPECTED WAVEFORM:



E. Space Notes: Cite your resources to justify your answers here. APA format. Kindly write your constructive reflection on how to make this learner's module better□.

Reference

IEEE.(1996). Standard Hardware Description Language Based on the Verilog(R) Hardware Description Language [eBook edition]. IEEE. https://ieeexplore.ieee.org/servlet/opac?punumber=4246

Reflection:

The Verilog structures depict basic circuit components like logic gates. A wider circuit is defined by creating a code that connects such parts. This is referred to as the structural representation of logic circuits. However, it also describes a circuit using logic expressions. and programming constructs that specify the circuit's function but not its precise gate-level organization. The behavioral representation is what we refer to as this. We cannot define a function inside of another one in the C programming language. But we can refer to a function that are contained within other functions. Similar to that, no module definition can include another. Within its module and endmodule lines is a module definition. An alternative is a module definition. Instantiate other modules to include copies of them.

To declare an instance of a module inside another module, you write the name of your module (like HalfAdder in the example), give it a unique name (like HA1, HA2 in the example) and connect the ports. In the example code, the line: HalfAdder HA1 (.a(x),.b(y),.sum(s1),.carry(c1)); tells the synthesizer (like a compiler in C programming language) to create an instance of the half adder module called HA1, connect the inputs x and y of the full adder to the ports a and b of the instance and to connect sum and the carry_out of the half adder to the internal signals (wires) s1 and c1. The other line: HalfAdder HA2 (.a(cin),.b(s1),.sum(sum),.carry(c2)); is generating another instance called HA2, where signal s1 is connected to port b of the half adder, the input cin is connected to port a, the carry_out port is connected to an internal signal c2, and the sum port is connected to the output sum. The keyword wire is used for internal signal, which are neither inputs nor outputs. "|" indicates a bit-wise "or"ing.

```
module HalfAdder(input a,b, output sum, carry_out);
        assign sum = a^b; // sum = a xor b
        assign carry_out = a&b; // carry = a and b
endmodule
module FullAdder (input x,y,cin, output sum, carry_out);
        wire s1,c1,c2;
        HalfAdder HA1 (.a(x),.b(y),.sum(s1),.carry_out (c1));
        HalfAdder HA2 (.a(cin),.b(s1),.sum(sum),.carry_out (c2));
        assign carry_out = c1|c2;
endmodule
```

Link:

https://blog.cloudv.io/assets/files/CSCE239L_Experiment_4.pdf