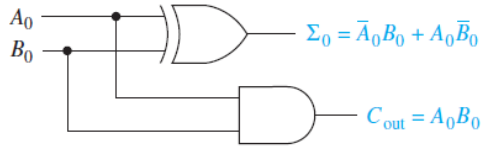
CSULB
CECS225

Model a ripple carry adder and review module instantiation.

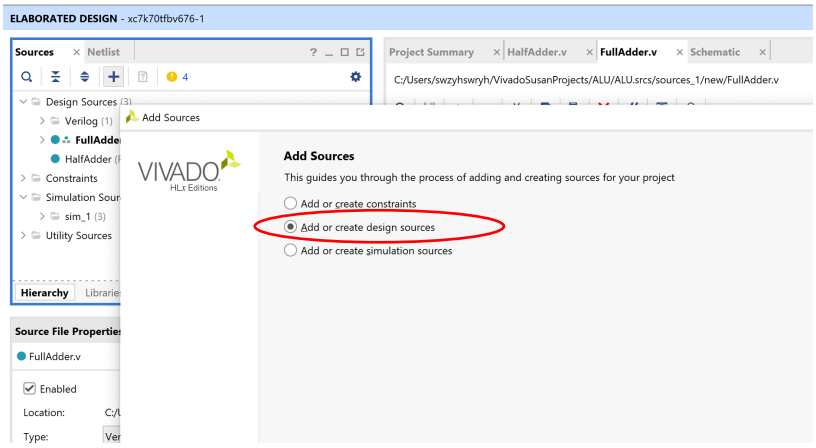
**PROCEDURE:** First create the half adder using an XOR gate and an AND gate.



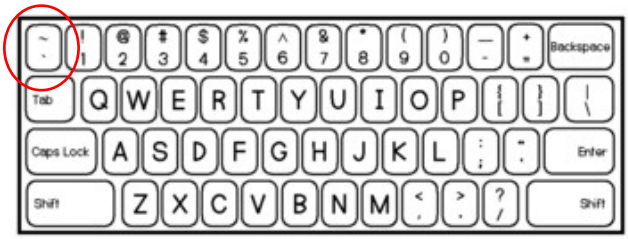

1. Complete the skeleton below to create the Half Adder Verilog module:

```
module HalfAdder(
   input a,
   input b,
   output c,
   output s
   );
   xor x1(          );
   and a1(          );
endmodule
```

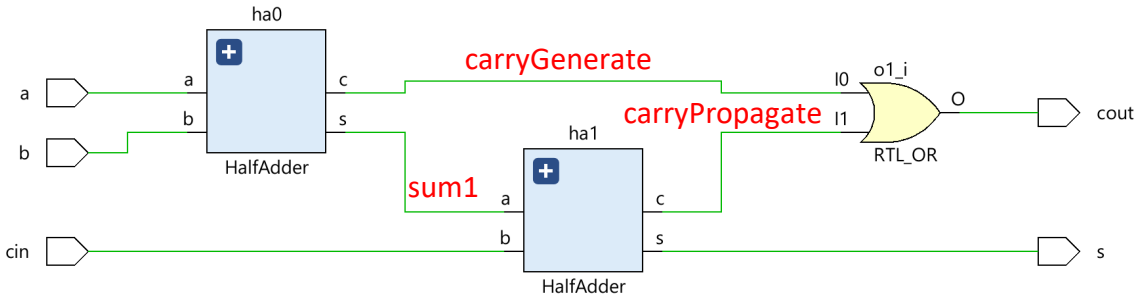
2. From the Flow Navigator, click add Sources and add another design file. Or click the + button to create a new file, enter the name **FullAdder.v** as shown below




Before the module definition in the new file include the following line
**`include "HalfAdder.v"**



Then finish the fullAdder module according to the circuit depicted below, observe the interconnections:



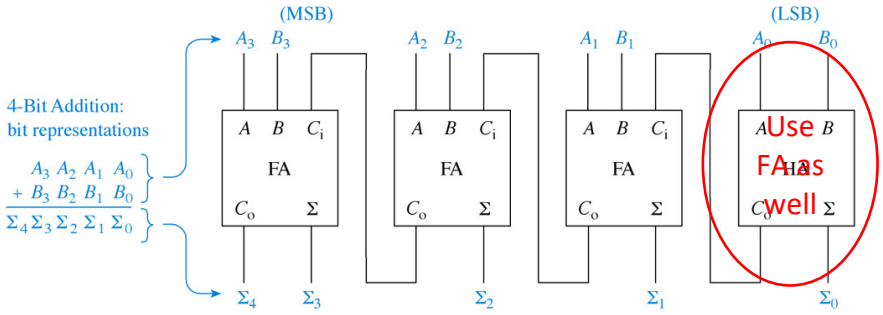

```
21
22  `include "HalfAdder.v"
23
24  module FullAdder(
25      input a,
26      input b,
27      input cin,
28      output s,
29      output cout
30      );
31      wire carryGenerate, carryPropagate, sum1;
32      HalfAdder
33          ha0(                                ),
34          ha1(                                );
35      or
36          o1(                                 );
37
38  endmodule
39
```

```
`include "HalfAdder.v"

module FullAdder(
  input a,
  input b,
  input cin,
  output s,
  output cout
  );
  wire carryGenerate, carryPropagate, sum1;
  HalfAdder
    ha0 (                          ),
    ha1 (                          );
  or
    o1 (                           );
endmodule
```

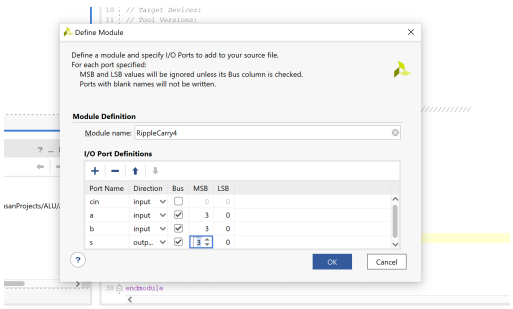
**Explanation:** the **include** directive copies the source code from **HalfAdder.v** into **FullAdder.v** so that the **Halfadder** module can be "instantiated" or used.
The **halfadder** *module instantiation* occurs on lines 32, 33, and 34 of **FullAdder.v**. The **HalfAdder** module is called by name on line 32, *two instances* are created on lines 33 and 24.
Module instantiation works just like using gate level primitives except the order of inputs and outputs may vary.
The **HalfAdder** *port list* is (**input a, input b, output c, output s**) so variables must be put on the **HalfAdder** *instance port list* in the order that is determined by the **HalfAdder** module definition.

The Ripple Carry Adder (**RippleCarry4**) will be the **top level module** for this lab. A **top level** module can be thought of as a main function. In Verilog the top level module is at the top of the hierarchy i.e. it typically contains all of the sub modules in a project/design.

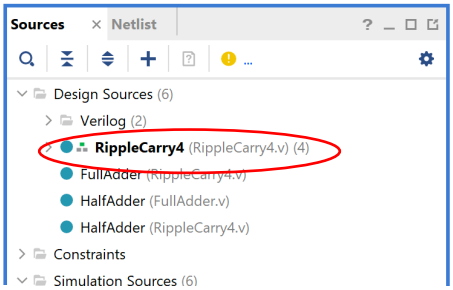
3. Complete the module skeleton that is given below according to the block diagram **but use a full adder for the LSB instead of HA**:




This is how you declare vectors (buses)




Notice, when you synthesis how your last module will automatically become the **top level module** (It will be recognized as the contents of **design.sv**)




```
`include "FullAdder.v"
module RippleCarry4(
   input cin,
   input [3:0] a,
   input [3:0] b,
   output [3:0] s,
   output cout
   );

   wire cout0, cout1, cout2;

   FullAdder
   fa0(a[0],      , cin, s[0], cout0 ),
   fa1(     , b[1],      , s[1], cout1),
   fa2(a[2],      , cout1,     ,      ),
   fa3(     ,      ,      ,     ,      );

endmodule
```

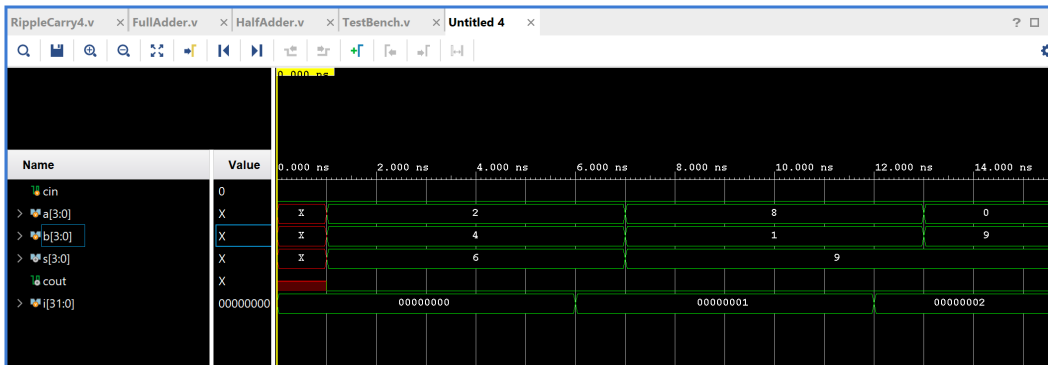
Once the RippleCarry4 module has been completed, it is time to test the design. Testbench code is given below:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Name: Susan Nachawati
//
// Create Date: 10/26/2020 01:38:07 PM
// Design Name: RippleCarry4
// Module Name: TestBench
//
//
//////////////////////////////////////////////////////////////////////////////////


module TestBench();
    reg cin;
    reg [3:0] a, b;
    wire [3:0] s;
    wire cout;

    RippleCarry4 rut(cin, a, b, s, cout);

    integer i;
    initial
    begin
        $display ("4-bit Ripple Carry");

        cin = 0;
        for(i=0; i <8; i = i+1)
            begin
                #1 {a, b} = $random;
                #5 $display("a = %d,  b= %d, s = %d, cout = %b", a, b, s, cout);
            end //for
    end//initial
endmodule
```

If everything works correctly then the following console output will be produced:

```
4-bit Ripple Carry
a =  2,  b=  4, s =  6, cout = 0
a =  8,  b=  1, s =  9, cout = 0
a =  0,  b=  9, s =  9, cout = 0
a =  6,  b=  3, s =  9, cout = 0
a =  0,  b= 13, s = 13, cout = 0
a =  8,  b= 13, s =  5, cout = 1
a =  6,  b=  5, s = 11, cout = 0
a =  1,  b=  2, s =  3, cout = 0
```

And your timing diagram should look like this:



If your module did not work correctly get assistance from the instructor or a reputable student to troubleshoot your design.

**WHAT TO TURN IN:** Once your RippleCarry4 module is working correctly:

- Copy the contents of your **RippleCarry4** module to a file named **with a screenshot of your output and timeline.**
- **Explain your briefly while voice recording your screen, simulate your code, show the timeline being generated, and the output explain your output. Upload to YouTube.**
- Save a link to your video in the same file as your code and the output. Upload as a pdf.

**NOTE:**
Keep the source files from this lab as they will be used in future Labs!
halfadder, fulladder, and RippleCarry4 modules will be used in upcoming labs.