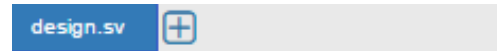


CECS 225: LAB 4

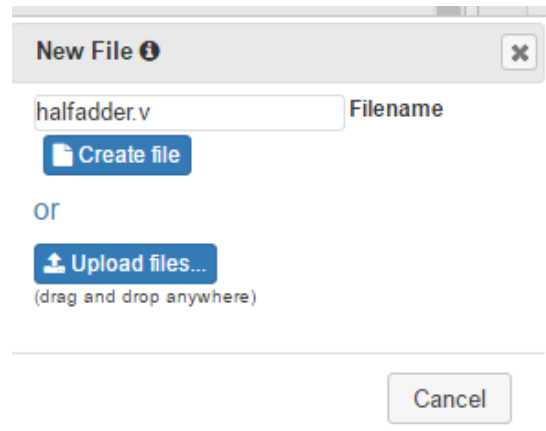
OBJECTIVE: Model a ripple carry adder and learn module instantiation.

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

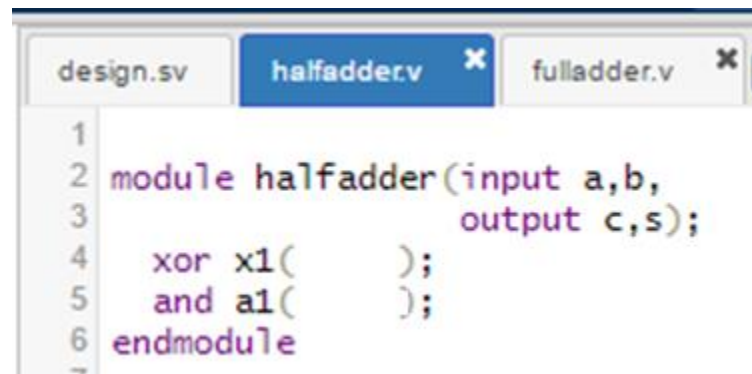
1. Click the + button to create a new file



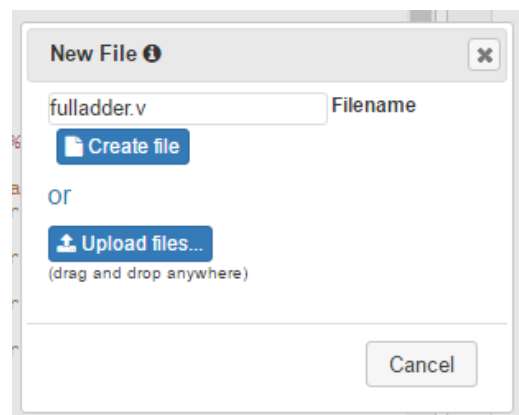
2. Enter the name **halfadder.v** and then click **Create file**



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



4. Click the + button to create a new file, enter the name **fulladder.v** as shown below:

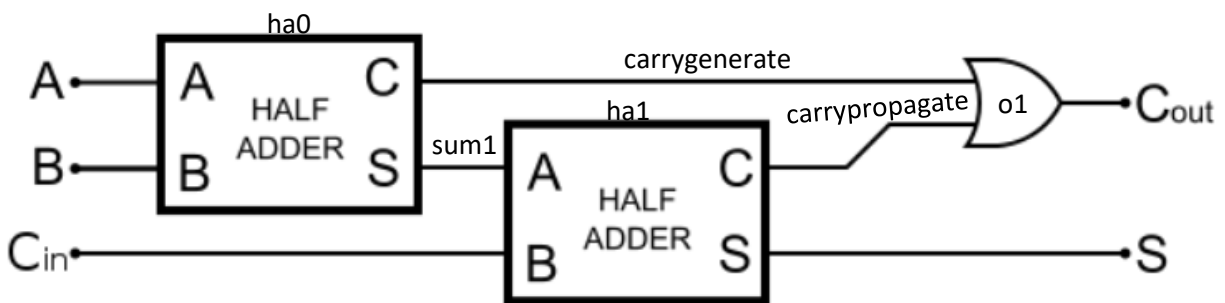


CECS 225: LAB 4

5. Enter the source code for fulladder.v as shown below:

```
design.v  halfadder.v  fulladder.v  +
1
2 `include "halfadder.v"
3 module fulladder(input a,b,cin,
4                   output cout, s);
5
6   wire carrygenerate, carrypropagate, sum1;
7
8   halfadder
9   ha0(a,b,carrygenerate, sum1),
10  ha1(sum1, cin, carrypropagate, s);
11  or
12  o1(cout, carrygenerate, carrypropagate);
13 endmodule
14
15
```

The module above is created according to the circuit depicted below, observe the interconnections:



Explanation: the **include** directive on line 2 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 8, 9, and 10 of **fulladder.v**. The **halfadder** module is called by name on line 8, two instances are created on lines 9 and 10.

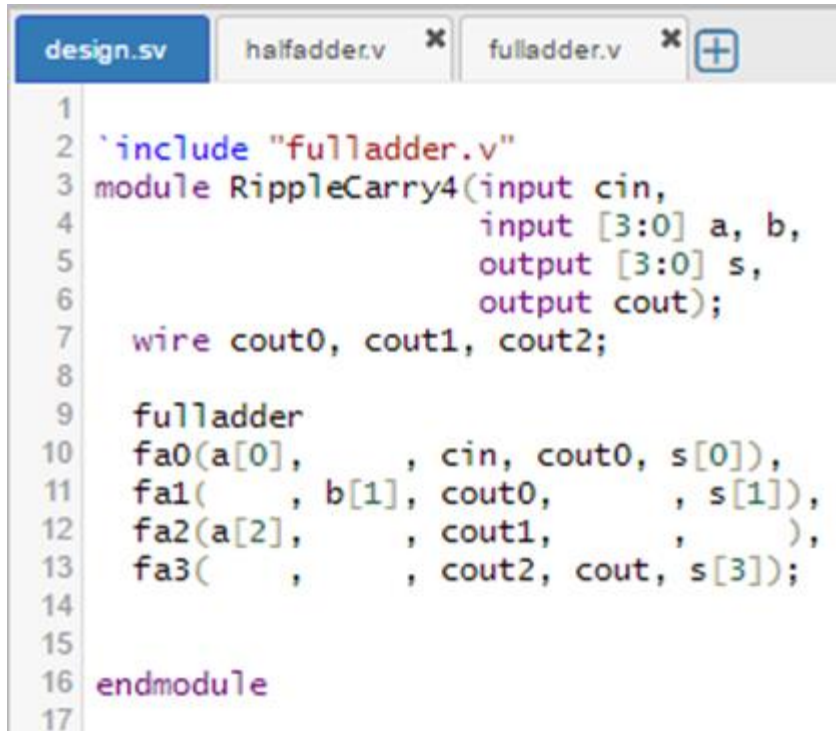
Module instantiation works just like using gate level primitives except the order of inputs and outputs may vary. The **halfadder** port list is (**input a**, **input b**, **output c**, **output s**) so variables must be put on the **halfadder** instance port list in the order that is determined by the **halfadder** module definition.

CECS 225: LAB 4

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.

In EDA playground, the **top level module** will automatically be recognized as the contents of **design.sv**.

6. Complete the module skeleton that is given below according to the block diagram given in class:



```
1
2 `include "fulladder.v"
3 module RippleCarry4(input cin,
4                     input [3:0] a, b,
5                     output [3:0] s,
6                     output cout);
7     wire cout0, cout1, cout2;
8
9     fulladder
10    fa0(a[0],      , cin, cout0, s[0]),
11    fa1(      , b[1], cout0,      , s[1]),
12    fa2(a[2],      , cout1,      , ),
13    fa3(      ,      , cout2, cout, s[3]);
14
15
16 endmodule
17
```

Once the RippleCarry4 module has been completed, it is time to test the design.

CECS 225: LAB 4

Testbench code is given below.

```
testbench.sv
1
2 module RippleCarry4Tester();
3
4     reg cin;
5     reg [3:0] a,b;
6     wire [3:0] s;
7     wire cout;
8
9     RippleCarry4 r(cin,a,b,s,cout);
10
11     integer i;
12
13     initial begin
14         // Dump waves
15         $dumpfile("dump.vcd");
16         $dumpvars(1);
17
18         cin = 0;
19         for(i = 0; i < 8; i = i + 1)
20             begin
21                 #1 {a,b} = $random;
22                 #5 $display("a = %d, b = %d, s = %d, cout = %b",a,b,s,cout);
23             end
24         $finish;
25     end
26
27 endmodule
28
```

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

```
VCD info: dumpfile dump.vcd opened for output.
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
Finding VCD file...
./dump.vcd
[2017-03-05 17:32:05 EST] Opening EPWave...
Done
```

CECS 225: LAB 4

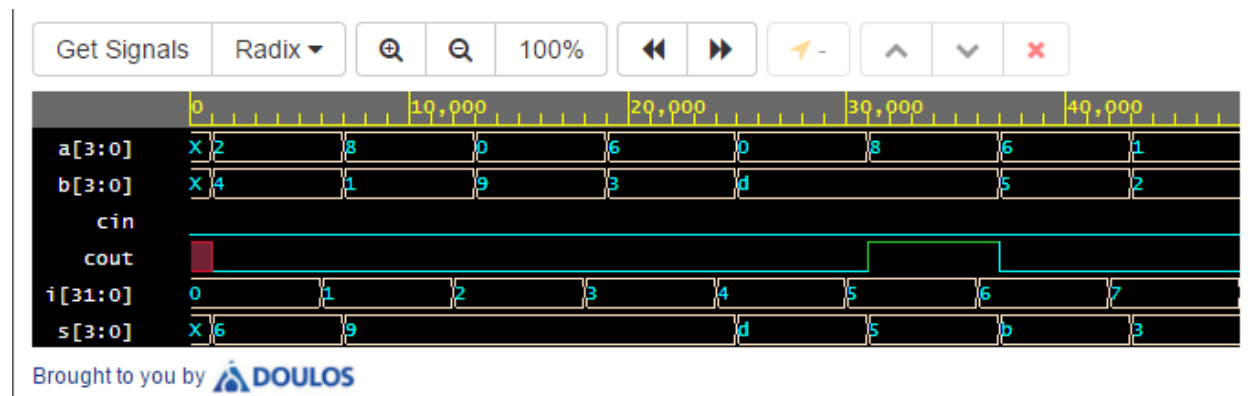
If your module did not work correctly get assistance from the instructor or a reputable student to troubleshoot your design. When designs become increasingly complex then so does the troubleshooting. Lines 15 and 16 of the testbench will create a waveform dump file for in depth signal analysis. Change line 16 of the testbench to **\$dumpvars(0);** and then you can add signals from each module to the waveform viewer once simulation is run. One additional step to launch the waveform viewer is to enable the following checkbox before clicking **Run**.

Run Options

☒ Open EPWave after run

☐ Download files after run

Waveforms of a properly working RippleCarry4 appear as follows:



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

- Copy the contents of your **RippleCarry4** module to a file named **RippleCarry4.txt**
- upload **RippleCarry4.txt** to the beachboard dropbox for RippleCarry4

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.