LAB 05: 8-bit Ripple Carry Adder

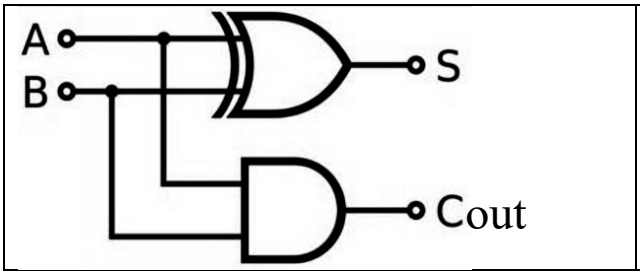
Purpose: In this project you are asked to define and test a Ripple Carry Adder (Part III)

Objectives:

- **Continue to get familiar with EDAPlayground**
- **Continue to learn HDL (Verilog) basics**
- **To understand arithmetic logic unit: half and full adder**
- **To understand arithmetic logic unit: 8-bit Ripple Carry Adder**

Part 1: Define and Test a Half Adder (Refer to previous lab)

The Half Adder is a digital building block with 2 inputs (A, B) and 2 outputs (S, Cout). The Half Adder logic must be modeled next. The circuit below shows the Half Adder logic circuit and the equivalent boolean equations:



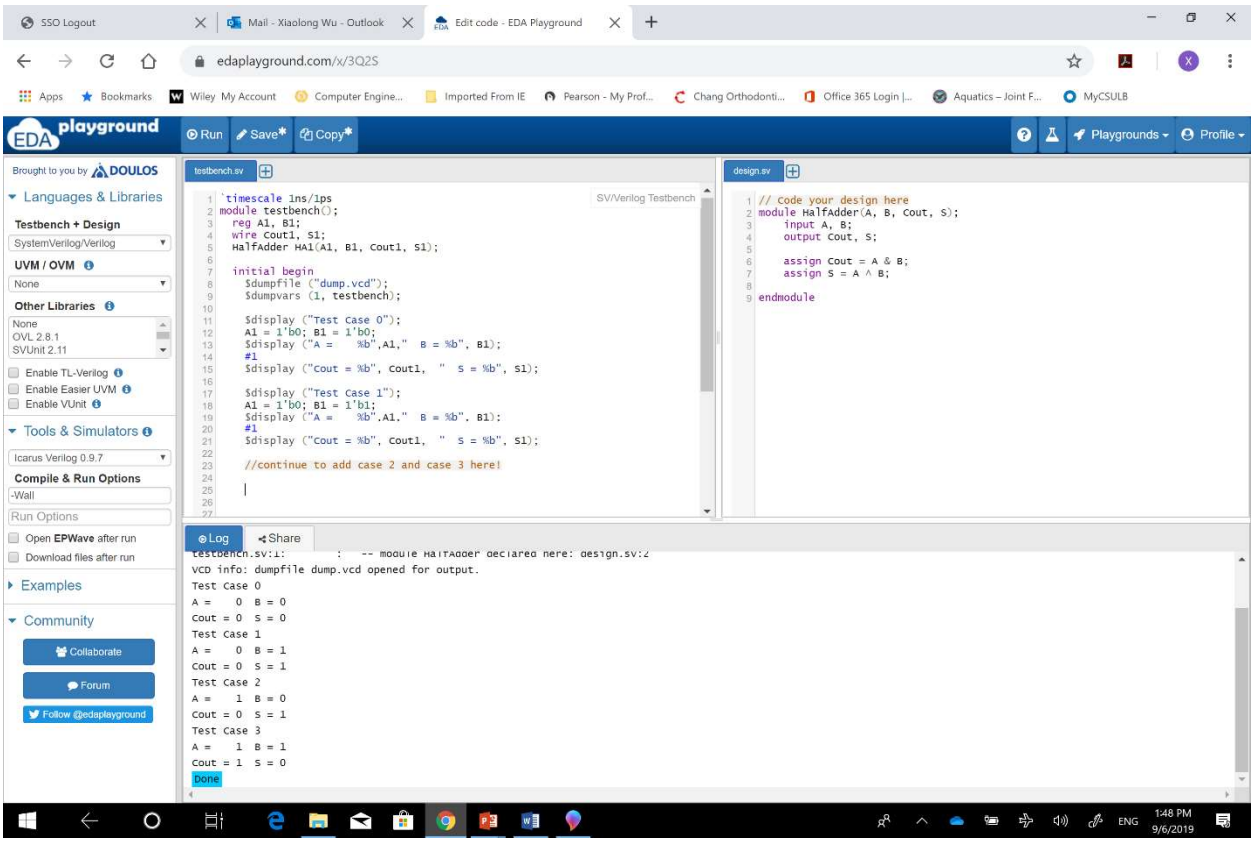
S = A xor B

Cout = A and B

Binary Operator Symbols in Verilog

~	NOT
*, /, %	mult, div, mod
+, -	add,sub
<<, >>	shift
<<<, >>>	arithmetic shift
<, <=, >, >=	comparison
==, !=	equal, not equal
&, ~&	AND, NAND
^, ~^	XOR, XNOR
|, ~|	OR, NOR
?:	ternary operator

This completes the verilog module definition of the half adder. Next it must be tested to ensure it works correctly.



Step 2: Create a Half Adder Verilog Test Fixture. To test a module for correct functionality, a set of inputs will be provided to produce an expected set of outputs. A **Verilog Testbench** is used to test a Verilog source module.

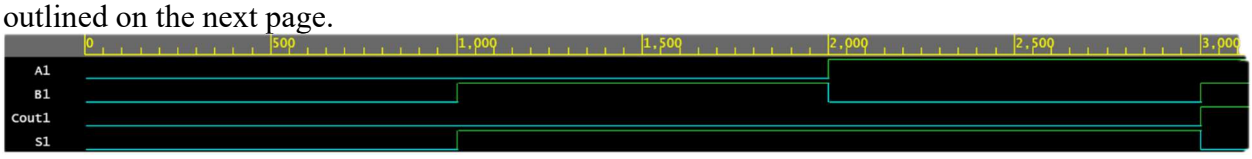
Create the Half Adder test script. To test a module for correct functionality, a set of inputs will be provided to produce an expected set of outputs. For simple modules like the half adder a truth table is used to show the outputs that can be expected from a set of inputs.

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The Cout output column shows that Cout equals 1 only when A equals 1 and B equals 1.

The S output column shows that S equals 1 when the value of A is not equal to the value of B.

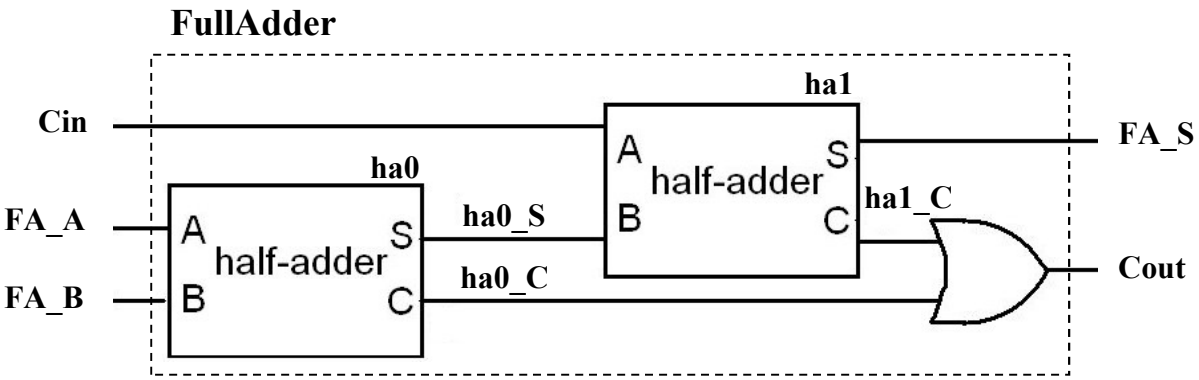
Simulation results are shown as waveforms. Steps to view and interpret simulation results are outlined on the next page.



Part 2: Define and test a Full Adder

(Read Chapter 5 page 240 of the textbook for an explanation of the Full Adder.)

Step 1: The plan! A Full Adder will be created using **hierarchical design**. A Full Adder can be made by using two instances of the **Half Adder** and an **OR** gate as shown below.



The circuit has been annotated with extra labels for easy translation into Verilog. Below is the Verilog module to model everything within the dotted box above.

design.sv

```
// Code your design here
module HalfAdder(A, B, Cout, S);
    input A, B;
    output Cout, S;

    assign Cout = A & B;
    assign S = A ^ B;

endmodule


module FullAdder(Cin,FA_A, FA_B,FA_S, FA_Cout);
    input FA_A, FA_B, Cin;
    output FA_S, FA_Cout;

    wire ha0_S, ha0_C, ha1_C;

    HalfAdder ha0 (     .A (    FA_A    ),
                        .B (    FA_B    ),
                        .Cout ( ha0_C   ),
                        .S (    ha0_S   )
                  );

    HalfAdder ha1 (     .A (        ),
                        .B (        ),
                        .Cout (     ),
                        .S (        )
                  );
  //YOU NEED TO FILL OUT INPUTS/OUTPUTS FO HA1
    assign FA_Cout = ha0_C | ha1_C; //THIS IS THE CARRY OUT FOR THE FULLADDER

endmodule
```

Take note of how the labels in the diagram correlate to the labels in the Verilog module. There are two instances of the HalfAdder module created in Part 1 of this lab. The instances have instance labels **ha0** for HalfAdder zero and **ha1** for HalfAdder one. Variables in Verilog are referred to as *signals*. A Verilog convention known as named port mapping is used to connect inputs and outputs of the HalfAdder to signals within the FullAdder. The

port signals from the HalfAdder are preceded with the dot operator and signals from the FullAdder go in the following parenthesis. *Local* signals that are not inputs or outputs within the FullAdder must be declared using the wire keyword. Finally the OR gate is represented by the assignment statement where the OR logic operation.

design.sv

```
// Code your design here
module HalfAdder(A, B, Cout, S);
    input A, B;
    output Cout, S;

    assign Cout = A & B;
    assign S = A ^ B;

endmodule


module FullAdder(Cin,FA_A, FA_B,FA_S, FA_Cout);
    input FA_A, FA_B, Cin;
    output FA_S, FA_Cout;

    wire ha0_S, ha0_C, ha1_C;

    HalfAdder ha0 (     .A (    FA_A    ),
                        .B (    FA_B    ),
                        .Cout ( ha0_C   ),
                        .S (    ha0_S   )
                  );

    HalfAdder ha1 (     .A (        ),
                        .B (        ),
                        .Cout (     ),
                        .S (        )
                  );
  //YOU NEED TO FILL OUT INPUTS/OUTPUTS FO HA1
    assign FA_Cout = ha0_C | ha1_C; //THIS IS THE CARRY OUT FOR THE FULLADDER

endmodule
```

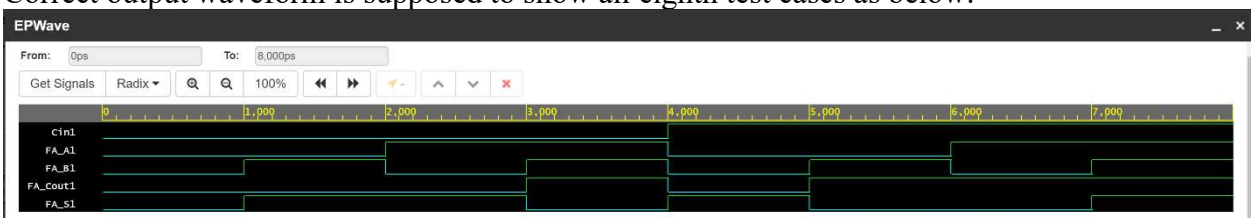
Use the given truth table on the left to make your test cases.

C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

testbench.sv

```
1  `timescale 1ns/1ps
2
3  module testbench();
4    reg FA_A1, FA_B1, Cin1;
5    wire FA_S1, FA_Cout1;
6
7
8    FullAdder FA1( Cin1, FA_A1, FA_B1, FA_S1, FA_Cout1);
9
10   initial begin
11     $dumpfile ("dump.vcd");
12     $dumpvars (1, testbench);
13
14     $display ("Test Case 0");
15     Cin1 = 1'b0; FA_A1 = 1'b0; FA_B1 = 1'b0;
16     $display ("Cin = %b",Cin1, "    FA_A = %b",FA_A1,  "   FA_B = %b", FA_B1);
17     #1
18     $display ( "FA_Cout = %b", FA_Cout1, "     FA_S = %b", FA_S1);
19
20
21
22     //CONTINUE TO ADD THTE REMAING TEST CASES
23
24   end
25
26 endmodule
27
```

Correct output waveform is supposed to show all eighth test cases as below:




```
[2019-10-21 15:16:00 EDT] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
testbench.sv:1: warning: Some modules have no timescale. This may cause
testbench.sv:1:        : confusing timing results.  Affected modules are:
testbench.sv:1:        :   -- module FullAdder declared here: design.sv:13
testbench.sv:1:        :   -- module HalfAdder declared here: design.sv:3
VCD info: dumpfile dump.vcd opened for output.
Test Case 0
Cin = 0   FA_A = 0  FA_B = 0
FA_Cout = 0    FA_S = 0
Test Case 1
Cin = 0   FA_A = 0  FA_B = 1
FA_Cout = 0    FA_S = 1
Test Case 2
Cin = 0   FA_A = 1  FA_B = 0
FA_Cout = 0    FA_S = 1
Test Case 3
Cin = 0   FA_A = 1  FA_B = 1
FA_Cout = 1    FA_S = 0
Test Case 4
Cin = 1   FA_A = 0  FA_B = 0
FA_Cout = 0    FA_S = 1
Test Case 5
Cin = 1   FA_A = 0  FA_B = 1
FA_Cout = 1    FA_S = 0
Test Case 6
Cin = 1   FA_A = 1  FA_B = 0
FA_Cout = 1    FA_S = 0
Test Case 7
Cin = 1   FA_A = 1  FA_B = 1
FA_Cout = 1    FA_S = 1
Done
```

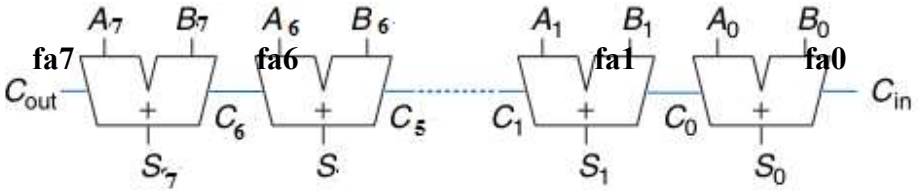
Part 3: Define and test a 8-bit RCA

RCA8 will have:

- two 8-bit inputs **A_8, B_8**
- a 1-bit input **Cin_1**
- a 1-bit output **Cout_1**
- an 8-bit output **S_8**

The internal structure of the RCA8 module follows Figure 5.5 of the textbook however there will only be eight Full Adders(numbered 0 through 7) tied together.

Ripple Carry Adder Diagram



Here is partial Veriloy code, you need to continue from the previous FA to fill out the rest!

```
// Code your design here
module HalfAdder(A, B, Cout, S);
        input A, B;
        output Cout, S;

        assign Cout = A & B;
        assign S = A ^ B;

endmodule


module FullAdder (Cin, FA_A, FA_B,FA_S, FA_Cout);
        input FA_A, FA_B, Cin;
        output FA_S, FA_Cout;

        wire ha0_S, ha0_C, ha1_C;

        HalfAdder ha0 (             .A (    FA_A  ),
                                    .B (    FA_B  ),
                                    .Cout ( ha0_C ),
                                    .S (    ha0_S )
            );

        HalfAdder ha1 (             .A (    Cin   ),
                                    .B (    ha0_S ),
                                    .Cout ( ha1_C ),
```

```
                                                .S (      FA_S   )
                                                );
        assign FA_Cout = ha0_C | ha1_C;

endmodule

//now we are going to create a 8-bit ripple carry adder
module RCA8 (A_8, B_8, Cin_1, Cout_1, S_8);
  input [7:0] A_8, B_8;
  input Cin_1;
   output Cout_1;
   output [7:0] S_8;

        wire c0, c1, c2, c3,c4, c5, c6;

        FullAdder fa0 ( .Cin    ( Cin_1   ),
              .FA_A   ( A_8[0] ),
              .FA_B   ( B_8[0] ),
              .FA_S   ( S_8[0] ),
              .FA_Cout(   c0   )
             );

        FullAdder fa1 (  .Cin    (  c0    ),
              .FA_A   ( A_8[1] ),
              .FA_B   ( B_8[1] ),
              .FA_S   ( S_8[1] ),
              .FA_Cout(   c1   )
             );

//Make sure add corresponding variables for fa2~fa7

        FullAdder fa2 (  .Cin    (),
              .FA_A   (),
              .FA_B   (),
              .FA_S   (),
              .FA_Cout(   )
             );

        FullAdder fa3 (  .Cin    (      ),
              .FA_A   (),
              .FA_B   (),
              .FA_S   (),
              .FA_Cout(  )
             );

        FullAdder fa4 (  .Cin    (  ),
              .FA_A   (),
              .FA_B   (),
```

```
                .FA_S   (),
                  .FA_Cout(      )
              );

        FullAdder fa5 (  .Cin    (  ),
                .FA_A   (),
                .FA_B   (),
                .FA_S   (),
                  .FA_Cout(      )
              );


        FullAdder fa6 (  .Cin    (   ),
                .FA_A   (),
                .FA_B   (),
                .FA_S   (),
                  .FA_Cout(      )
              );

  FullAdder fa7 (  .Cin    (  c6   ),
                .FA_A   ( A_8[7] ),
                .FA_B   ( B_8[7] ),
                .FA_S   ( S_8[7] ),
                   .FA_Cout( Cout_1 )
              );
endmodule
```

Here is the testbench code with the first test cases, please continue to add the rest!

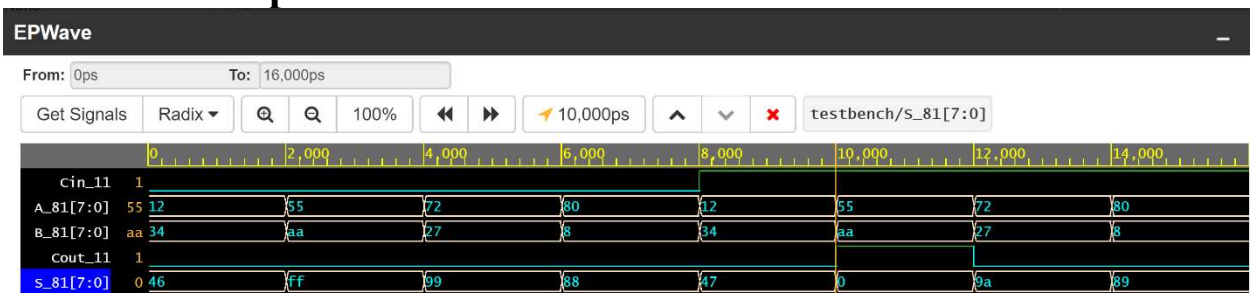
```
`timescale 1ns/1ps
module testbench();
  reg [7:0] A_81, B_81;
  reg  Cin_11;
  wire Cout_11;
  wire [7:0] S_81;

  RCA8 rca(A_81, B_81, Cin_11,Cout_11, S_81);

  initial begin
    $dumpfile ("dump.vcd");
    $dumpvars (1, testbench);

    $display ("Test Case 0");
    Cin_11 = 1'b0;  A_81 = 8'h12; B_81 = 8'h34;
    $display ("Cin_1 = %b",Cin_11, "   A_8 = %2h",A_81,  "  B_8 = %2h", B_81);     #2
    $display ( "Cout_1 = %b", Cout_11, "    S_8 = %2h", S_81);

    $display ("Test Case 1");
    Cin_11 = 1'b0;  A_81 = 8'h55; B_81 = 8'hAA;
```

```
    $display ("Cin_1 = %b",Cin_11, "   A_8 = %2h",A_81,  "  B_8 = %2h", B_81);     #2
    $display ( "Cout_1 = %b", Cout_11, "    S_8 = %2h", S_81);
```

You need to add the rest SIX testcases here!!!

```
  end
endmodule
```

Here is the output after the simulation!



Lab Report: Submit a single PDF or WORD to beachboard with the following contents.

- **Title Page**
  - CECS 225
  - Lab 5
  - Your Name
- **Section 1:** RCA Verilog module source code
- **Section 2:** RCA Verilog Test code
- **Section 3:** Function Table with 8 test cases

Inputs				Outputs	
Test Case	Cin	A_8	B_8	Cout	S_8
0	0	12	34		
1	0	55	AA		
2	0	72	27		
3	0	80	08		
4	1	12	34		
5	1	55	AA		
6	1	72	27		
7	1	80	08		

- **Section 4:** RCA Simulation Screenshot showing correct results (make sure the order of input/output variables match the truth table)