

## Lab 2 Full-Adder

### CMPE 125

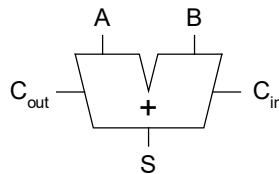
#### Introduction

In this lab you will design a simple digital circuit called a *full adder*. Along the way, you will learn to use the Altera field-programmable gate array (FPGA) tools to enter a schematic, simulate your design.

#### Background: Adders

An adder, not surprisingly, is a circuit whose output is the binary sum of its inputs. Since adders are needed to perform arithmetic, they are an essential part of any computer. The full adder will be an integral part of the microprocessor that you design in later labs.

A full adder has three inputs ( $A$ ,  $B$ ,  $C_{in}$ ) and two outputs ( $S$ ,  $C_{out}$ ), as shown in Figure 1. Inputs  $A$  and  $B$  each represent 1-bit binary numbers that are being added, and  $S$  represents a bit of the resulting sum.



**Figure 1. Full adder**

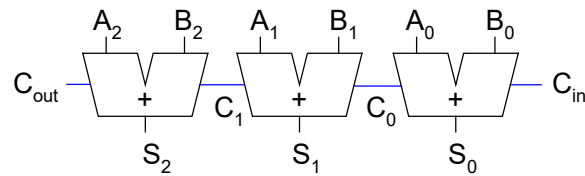
The  $C_{in}$  (carry in) and  $C_{out}$  (carry out) signals are used when adding numbers that are more than one bit long. To understand how these signals are used, consider how you would add the binary numbers 101 and 001 by hand:

$$\begin{array}{r} 1 \\ 101 \\ + 001 \\ \hline 110 \end{array}$$

As with decimal addition, you first add the two least significant bits. Since  $1+1=10$  (in binary), you place a zero in the least significant bit of the sum and carry the 1. Then you add the next two bits with the carry, and place a 1 in the second bit of the sum. Finally, you add the most significant bits (with no carry) and get a 1 in the most significant bit of the sum.

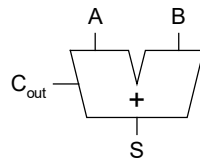
When a sum is performed using full adders, each adder handles a single column of the sum. Figure 2 shows how to build a circuit that adds two 3-digit binary numbers using three full adders. The

$C_{out}$  for each bit is connected to the  $C_{in}$  of the next most significant bit. Each bit of the 3-bit numbers being added is connected to the appropriate adder's inputs and the three sum outputs ( $S_{2:0}$ ) make up the full 3-bit sum result.



**Figure 2. 3-bit adder**

Note that the rightmost  $C_{in}$  input is unnecessary, since there can never be a carry into the first column of the sum. This would allow us to use a half adder for the first bit of the sum. A half adder is similar to a full adder, except that it lacks a  $C_{in}$  and is thus simpler to implement. To save you design time, however, you will only build a full adder in this lab.



**Figure 3. Half adder**

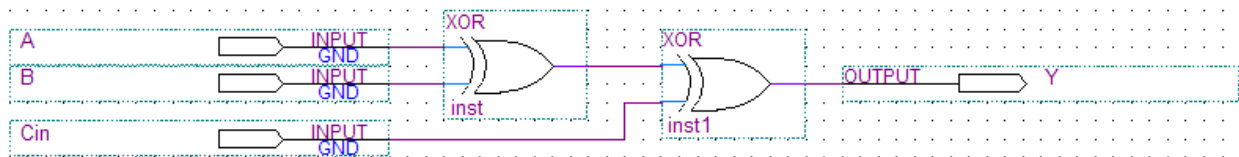
## 1. Design

A partially completed truth table for a full adder is given in Table 1. The table indicates the values of the outputs for every possible input, and thus completely specifies the operation of a full adder. As is common, the inputs are shown in binary numeric order. The values for  $S$  (sum) are given, but the  $C_{out}$  (carry out) column is left blank. **Complete the table by filling in the correct values for  $C_{out}$  so that adders connected as in Figure 2 will perform valid addition.**

Inputs			Outputs	
$C_{in}$	$B$	$A$	$C_{out}$	$S$
0	0	0		0
0	0	1		1
0	1	0		1
0	1	1		0
1	0	0		1
1	0	1		0
1	1	0		0
1	1	1		1

**Table 1.** Partially completed truth table for full adder

From the truth table, we now want to implement our design using logic gates. The sum output ( $S$ ) can be produced from the inputs by connecting two 2-input XOR gates as shown in Figure 4. You should convince yourself that this circuit produces the outputs for  $S$  as given in the table.



**Figure 4.** Schematic for sum logic

Using only two-input logic gates (AND, OR, XOR) and inverters (NOT), design a circuit that takes  $A$ ,  $B$ , and  $C_{in}$  as its inputs and produces the  $C_{out}$  output. Try to use the fewest number of gates possible. Sketch your schematic.

## 2. Schematic

Now that you know how to produce both the sum ( $S$ ) and carry out ( $C_{out}$ ) outputs using simple logic gates, you will now construct a working full adder circuit using real hardware. One way to test your circuit before building it in hardware is to enter the schematic representation of your logic into a software package. You can then simulate the circuit and test that it works the way you expect it to. Some software packages are then capable of programming the schematic into an integrated circuit.

First, you will learn how to start a new project. Start the Quartus software from the Start menu. If asked about the look and feel, choose Quartus II.

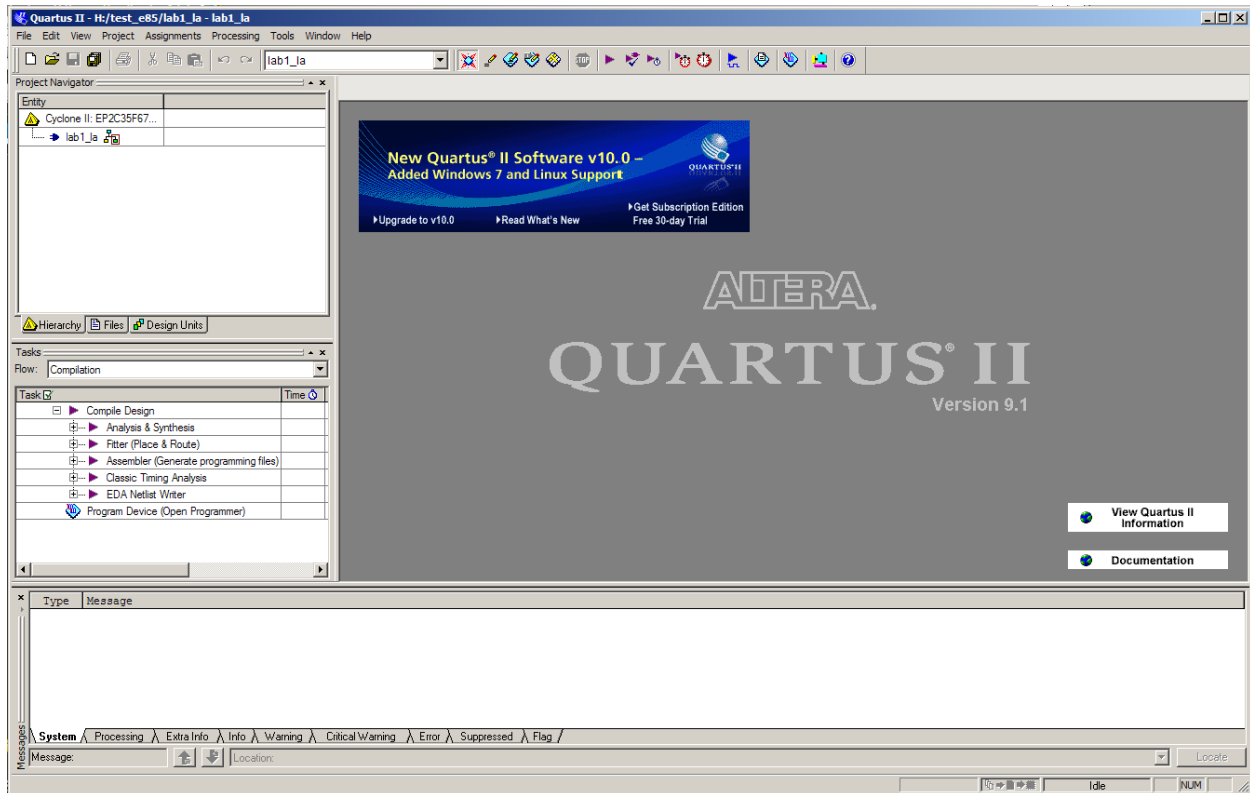
In the Getting Started Window, click on Create a New Project. In the New Project Wizard, set the working directory to a good place in your home directory. Name the project lab2\_xx, where xx is your name. Make sure there are no spaces or unusual characters in the path or file name; the tools may complain or silently misbehave if it has trouble with the file name. If prompted about whether to create the directory, say Yes.

Click Next to go to the Add Files page.

Click Next to go to the EDA (Electronic Design Automation) Tool Settings. You can skip this part and set up the simulation tool as in lab 1. Click Next and Finish to create your new project.

The Quartus window will open in a moment. You may wish to maximize the window. You will see three main panes, as shown in Figure 5 (and can bring them up from the View → Utility Windows menu if you accidentally close one):

- **Project Navigator:** Lists the current project's sources file and the chip in use.
- **Tasks:** Lists the processes to perform on the source selected in the Sources pane. For example, we will use this pane later to simulate your completed schematic.
- **Messages:** Lists the output of current processes, errors, and warning at the bottom of the screen. Keep an eye on these messages; important warnings appear here.




**Figure 5. Quartus II window**


We will describe some of the options for using these resources, but we also recommend exploring these resources on your own to become familiar with Quartus' capabilities. Use the Help menu for additional information.

Quartus has a basic and strikingly ugly schematic editor that we will use. It is not particularly sophisticated because designers today primarily use hardware description languages (HDLs) instead of schematics. However, understanding schematics is an important first step to mastering HDLs.

Create a new schematic by choosing File → New and selecting Block Diagram / Schematic File, and click OK. A new schematic window named Block1.bdf will appear.

First, place your logic gates. Click on the Symbol Tool icon (shaped like an AND gate). Expand the list of libraries in the upper left of the Symbol window by clicking on the arrow icons . Look under primitives → logic and choose xor. Click OK, then click twice on the schematic window to place two xor gates. Leave some room between the gates to draw a wire later. Press the Esc key or right click and choose Cancel to get out of the placement mode.

Click on the Symbol Tool again and choose primitives → pin → input. Place three input pins on the left side. Leave some space between the pins and the gates so that you can wire them together later. Then choose an output pin and place it on the right. Double click on one of the input pins and change its name to **A**. Leave the default value unchanged at VCC. Rename the other inputs to **B** and **Cin**. Rename the output to **S**.

Use the Orthogonal Node Tool () to wire the gates together. Click and drag to connect the pins to gates and the two gates together. At this point, your schematic should resemble Figure 4. It's

a good idea to click on the wire between the two XOR gates and give it a unique name such as *n1* or *mid* in case you need to debug later. (If you are using version 10 or higher, you can name a wire by right clicking on the wire, selecting Properties, and adding the name).


If you need to make corrections, use the Selection Tool to grab and move gates or wires. Zoom in and out by using the View menu or holding the Ctrl key while turning the mouse wheel. Use delete and undo as necessary.

Choose File → Save and save your schematic as lab2\_xx.bdf.

You are now ready to complete your schematic of the full adder by drawing the logic for  $C_{out}$  that you designed in Part 1. Draw the necessary logic gates and wires to complete the circuit. Use the existing input terminals for *A*, *B*, and *C<sub>in</sub>*, and add an output terminal for *C<sub>out</sub>*. The symbols you may use to draw your logic gates are as follows: and2, and3, or2, or3, not, and xor.

Remember, do not add a second set of input ports for *A*, *B*, and *C<sub>in</sub>*. Instead, note that you can connect multiple wires to the same input ports (or you can connect wires to other wires to create branches).

Select the Files tab in the Project Navigator pane to see a list of files of the project (presently just lab2\_xx.bdf). If you need to reopen the file later, double-click on it here.

To check your design, click on Start Compilation  in the Task pane (Processing→Start Compilation). You'll see a compilation report indicating five pins and 2 logic elements. Review the warnings and errors carefully. You may get the following warnings that are harmless:

- Feature LogicLock only available with subscription.
- Ignored location or region assignments
- Found output pins without load capacitance
- Found invalid Fitter assignments
- Reserve All Unused Pins not specified

If you see other warnings or errors, track down their root cause before they lead you to grief later.

### 3. Simulation

One motivation for drawing your full adder schematic in Quartus is that you can now use the software to simulate the operation of the circuit. It is a good idea to verify the correctness of your design before actually building the circuit in hardware. In this part of the lab, you will simulate the design using **ModelSim**.

ModelSim expects a description of a circuit in a hardware description language (HDL) such as Verilog. To convert your schematic to Verilog, open the schematic and choose File → Create / Update → Create HDL Design File for Current File. Choose Verilog HDL. Your file should be written to lab2\_xx.v. Watch for and correct any warnings or errors that arise.

Now fire up ModelSim SE 6.6b from the Windows start menu. Maximize the ModelSim window when it opens. If prompted, you may wish to associate file types with ModelSim but do not want to use Jumpstart.

Choose File → New → Project. Name the project lab2\_xx and put it in the directory where you are working. Accept the default library name of “work.” Then click “Add Existing File” and add lab2\_xx.v.

You should see lab2\_xx.v in the ModelSim project pane. Double-click on it to view it. The file should list the inputs and outputs and the wires (using default names if you didn’t name them yourself). It should then have a series of “assign” statements describing the gates. & indicates AND. | indicates OR. ^ indicates XOR. In future labs you will learn to write Verilog yourself.

Choose Compile → Compile All to compile the Verilog code into a form that ModelSim can simulate. Watch for and correct errors in the transcript pane. Then choose Simulate → Start Simulation. Click on Work to expand the library, and choose lab2\_xx as your module to simulate. Uncheck “enable optimization” because it sometimes hides information that is useful during debugging. Click Ok.

ModelSim will open more panes including sim and Objects that help you select signals for the waveform viewer. In the sim pane, be sure lab2\_xx is selected. In the objects window, you’ll see all the inputs, outputs, and internal wires. Shift-click to select them all. Then right-click and choose Add → To Wave → Selected Signals. A Wave pane will pop up with the signals.

## 4. Verification

Now it is time to apply the inputs. In the transcript pane at the bottom, type

```
force A 0
force B 0
force Cin 0
run 100
```




This will set all three inputs to 0 and simulate for 100 ns. (Note that Verilog is case-sensitive; “A” and “a” are different.) You should see all the inputs and outputs at a low level in the Wave pane. Next, raise A:

```
force A 1
run 100
```

You’ll see A rise. If your design is correct, S will also rise.

Continue with the six other patterns of inputs to check your truth table.

If you have errors, you may want to look at the internal nodes to track down the problem. Fix the schematic, then regenerate the Verilog file. Recompile and restart the simulation in ModelSim.

If the waveform is not visible, click on the + button in the top right corner of the “wave-default” pane to the right of the main ModelSim window (or choose View→Wave from the menu). Click the “Zoom Full” icon in the taskbar  to see the whole waveform of the simulation results. You can also use the “Zoom In” and “Zoom Out” icons:  . Check and see that the output values ( $S$  and  $C_{out}$ ) are correct. If not, go back and fix your schematic and resimulate. When the output values are correct, you have a working full adder! Save an image of the waveform. Make sure the entire waveform is visible, select File→Export→Image..., and save the file. If needed, you can also print the waveform. Choose File→Print to print a copy of your waveforms to turn in. You can choose the start and end times in the bottom right of the print dialog box.

What to turn in:

1. Design the full-adder with schematics, compile and verify. Convert the schematic to Verilog automatically and simulate using the auto-generated Verilog code.
2. For verification of the design, use the commands in section 4. Verification to verify the logic in the truth table
3. Implement the full-adder directly in Verilog code, compile and verify. Simulate the direct Verilog implementation.
4. Verify the functional verification of the Verilog only implementation using ModelSim process outlined in step 2.
5. Put all this in a report per the report guideline on Canvas.