
Objective: To practice building large combinational circuits from subblocks.

.....

1 Getting the files

Download the files for this lab from

<http://www.eee.hku.hk/~elec3342/fa22/handout/elec3342lab3.tar.gz>

.....

2 Building Adders ... One Bit at a Time

In this part, you will first build a single-bit full adder (FA) using VHDL. Later in this lab, you will be reusing this full adder to build larger adders.

2.1 Full Adder – Preliminary A full adder (FA) adds two 1-bit inputs, a and b , with a single bit carry-in ci , and produces an 1-bit output s and a carry-out bit co . The function of a FA can be explained with a simple half adder (HA). A HA is the same as an FA except there is no carry input.

The function of an HA is to simply add two inputs a and b . Since $a, b = \{0, 1\}$, the output can only take the values 0, 1, or 2. If the output is 2, then it must produce an additional carry-out bit to the position to the left. As a result, the function of a half adder can be specified by the following truth table:

a	b	co	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Another way to look at the function of a half adder is that the carry-out bit (co) and sum bit (s) forms a two-bit output. (Notice how combining co and s together shows the values 0, 1, 1, 2 in binary.)

The function of a full adder is similar to that of a half adder with the additional carry-in (ci) input. The ci input takes the value from the carry out of the bit position to the right.

Complete the truth table for a full adder below. Note that the first 4 rows where carry-in is ‘0’ are the same as that of a half adder. When carry-in is ‘1’, it is essentially adding an additional 1 to the result. (The maximum output may therefore be 3.) You may treat co and s as a two-bit output like in the case of the half adder above.

ci	a	b	co	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

2.2 Implementayion in VHDL Now open Vivado:

- Create a new project called `test_fulladder`.
- Add the file `rtl/full_adder_1b.vhd` to design sources.

Open `full_adder_1b.vhd` in Vivado. You will see we already specified the library and defined the entity together with interface. But the architecture body is still empty. Design your FA by completin the architecture body in the file.

2.3 Check Yourself

Open the elaborated schematics. Refer to Lab 2 if you don't remember how to examine the elaborated design. Check the elaborated design from Vivado to confirm that it is working as expected?

2.4 Testing Your Full Adder Although you can visually inspect the gates one by one on a small 1-bit FA like above, the task becomes very challenging if you need to verify the function of large circuits. For verifying and debugging the function of complex circuits, a typical hardware design flow relies of the use of *testbenches*. Once setup, a testbench provides the input to your module as stimulus automatically. It can also be setup to verify the output of your circuits to check if it is behaving correctly.

In this lab, we have provided a testbench that you can add to your project by:

- Close your elaborated design.
- Click on **+**.
- Select **Add or create simulation sources**. [NOTE: it is *simulation* source.]
- Add `tb/full_adder_1b_tb.vhd` and click **Finish**.

Your source hierarchy should now look like this:

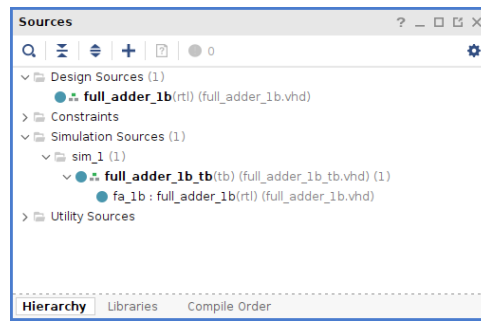


Figure 1:

Now run the behavioral simulation and check the waveform. Refer to Lab 1 if you don't remember how to run the simulation. Your waveform should look like this:

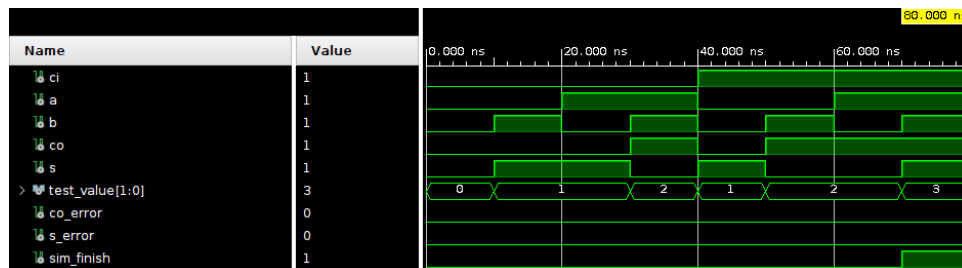


Figure 2:

This testbench tests all possible input combinations of *ci*, *a* and *b*. The corresponding output *co* and *s* of your module are also shown in the waveform. You can check if the output of your module is what you expected one by one.

We also provide two indicator signal *co_error* and *s_error*. They will tell you if your output has error. If you see value 1 on them, that means your output is not correct. Then you should go back and check your source file.

If you don't see any errors and you see the pulse on *sim_finish* at the end. That means your full adder is functioning correctly.

2.5 Checkoff 1

Get your modified `full_adder_1b.vhd` ready. You will be reusing this file later and uploading this file to Moodle at the end of this lab.

3 A 4-bit Adder

In this part, you will implement a 4-bit adder by using the full adder blocks you created above as a building blocks.

3.1 Implementation in VHDL

- Use the same project you created in previous section.
- Add the file `rtl/full_adder_4b.vhd` to design sources.

Similar to previous section, we already defined the library and entity. But you may notice some difference. Now the ports *a*, *b* and *s* are defined as `STD_LOGIC_VECTOR (3 DOWNTO 0)`. You will also note that the component definition of a FA has already been added for you.

3.2 Additional Signal Before you can instantiate 4 FAs, you need to think ahead how they should be connected. Refer to the lecture note. You will notice that the carry out of an FA needs to be connected to the carry in of the next FA. This additional connection is a *individual signal* node in VHDL that you need to define. You define such signals (node) in the architecture declaration. In the file `full_adder_4b.vhd`, create these additional signals that you need to connect the FAs that you will instantiate next. You can name them anything you want.

3.3 Instantiate and Connect To instantiate FA in your design, you need to use the following instantiation syntax in the architecture body:

```
<instance_name> : <component_name> port map (
    <component_signal_name> => <local_signal_name>,
    ...
);
```

- **instance_name** : a unique name to identify this instance.
- **component_name**: the name of the component to instantiate. For this part of the lab, it should be the entity name of the 1-bit adder you defined above (`full_adder_1b`).
- **component_signal_name => local_signal_name**: it says that the port `component_signal_name` on the component should be connected to the signal called `local_signal_name`. In this case, you want to put down something like `a => your_signal` to connect `your_signal` to the port `a`.

Once you have correctly instantiate 4 FAs in your system, Vivado will detect your design hierarchy automatically. You will see four `full_adder_1b` under the top `full_adder_4b` if you instantiate the `full_adder_1b` correctly:

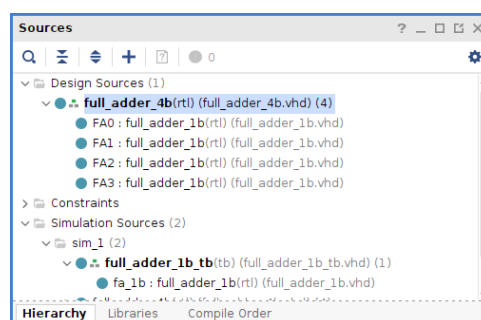


Figure 3:

3.4 Check Yourself

Open the elaborated design like before. Do you see 4 FAs instantiated? Are they connected through the new signals you defined above? Try to look *inside* an FA by descending the hierarchy. Do you see the FA implementation you saw above?

3.5 Test your circuit We also provide the testbench for the 4-bit adder. You can add `rtl/full_adder_4b_tb.vhd` to *simulation* sources. Now you have two testbench files in one simulation set. You need to right-click on `full_adder_4b_tb.vhd` and click **Set as top**. The top file will be bolded:

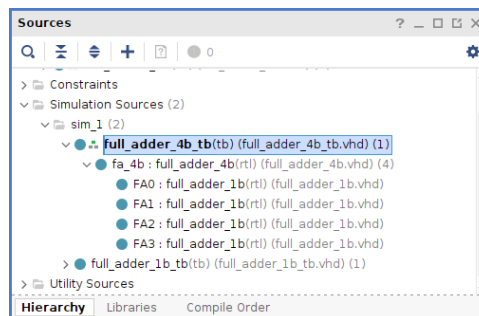


Figure 4:

Now you can run the simulation and check your waveform:

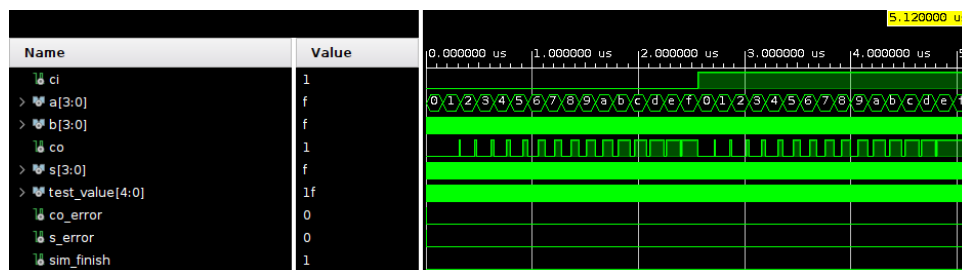



Figure 5:

Make sure you see the pulse on *sim_finish*. If not, your simulation is not finished. You need to click on  to let it finish.

3.6 Checkoff 2

- Take a screen capture of your elaborated 4-bit adder
- Get your modified `full_adder_4b.vhd` ready.

3.7 Submission Submit the files you have prepared from Checkoff 1 and 2 on Moodle.