Introduction to Electronic System Design (DAT093) Lab 0: Tool and task kickstarter

Sven Knutsson, Lars Svensson

Version 2.1, August 30, 2018

1 Introduction

In this initial lab session, you will carry out several tasks:

- Design and verify a one-bit full-adder circuit.
- Use your full-adder as a component to construct a four-bit adder circuit, and verify that design.
- Develop the four-bit adder into a four-bit counter, and verify that design.

The adder is a *combinational* circuit, where the output values depend only on the present inputs (after some inevitable circuit delay). In contrast, the counter is a *sequential* design, where a clock signal paces execution, and input values arbitrarily far back in time may affect the outputs.

Verification will be carried out by means of provided *test benches*, which specify a behavior by detailing the expected outputs for given combinations of inputs. For each test bench proper (a VHDL file), there is also a *do file* which specifies the simulator commands to be run, including commands that show informative error messages in case the design behaves in an unexpected manner. Test benches that specify expected outputs are highly useful for small designs such as these, and make the checking a go/no-go question[1] provided that the design has the correct

[1] In case of bugs, you must of course not change the provided test benches to make your design pass the test! If you wish to carry out further tests than those provided, make a copy of the test bench and/or the *do* file and modify those. In the lab, we will test your design using the unaltered test benches.

numbers and types of input and output connections.

2 Preparation

You should already have received a pointer to a VHDL video tutorial from the FPGA manufacturer Altera[2]. If you did not watch it yet, do so before coming to the lab. If your previous viewing was your first introduction to VHDL, it would be a good idea to watch it once more.

It is assumed that you are familiar with two's-complement representation of integers. A quick brush-up may be found in Wikipedia and in any undergraduate textbook on digital design. You may also refer to companion documents in this course, and to the VHDL lecture slides.

3 Getting started

For these and most other tasks in this lab series, we will use the Windows computers available in the course lab. These computers already have QuestaSim installed with a shortcut icon on the desktop (most computers at Chalmers don't have this software installed).

- Log into the system, using your Chalmers ID and CDKS password. Your first login may take extra time due to setup and initialization; be patient.
- Double-click on the QuestaSim icon to launch it.

An integrated design environment (IDE) window with multiple panes appears. Only the bare essentials of QuestaSim will be described here; refer to the companion documents and to the built-in tool help system for more details.

Most designs comprise several files: in addition to the VHDL code files, there will typically be test bench files, simulation results, and maybe other temporary files. As is common also in software IDEs, all these files and documents are collected in a *project*.

- Select **File→New Project...** A dialog window opens, allowing you to select a name for the project and a file system location[3]. Experience suggests

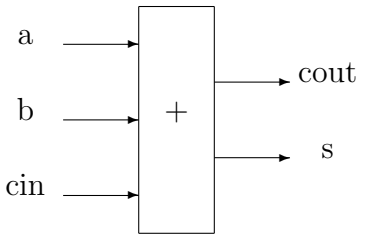
[2] `https://www.youtube.com/watch?v=k8Y0fWOpbg8`

[3] Do not store your files on the `C:` volume, as this is local to the computer you happen to

to use the same name for project and file-system folder. In the following, we assume that you use the name `lab0a` for your first project.

- Click **OK** to create the project. The middle pane of the IDE has a **Library** tab and a **Project** tab. Select the **Project** tab and verify that your new project is the current one. A new dialog has also opened to let you add files to or create files for the project.

4 A simple full adder



a	b	cin	s	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 1: FA block diagram and truth table.

A full adder (FA) is a logic block with three one-bit inputs (`a`, `b`, and `cin`) and two one-bit outputs (`s` and `cout`). Its block diagram and truth table are shown in Figure 1. Your next task is to design an FA in VHDL and verify it using the provided test bench.

- In the QuestaSim project dialog, click **Create New File** and specify the file name `full_adder.vhdl`. Click **OK** and verify that the file appeared in the file list for the project.
- Enter the VHDL code for the entity declaration in the new file. Use the entity name `full_adder`, as this is what the test bench expects. Use the type `STD_LOGIC` for input and output signals, again to fit with the test bench. Save the file.

In the project file list, your newly created file is highlighted with a blue question mark (?), indicating that compilation has not yet been carried out.

use at the moment; in the next session, you may be seated at another computer! Instead, use your directories on the network file system, which should be mapped to volume `Z:`.

- Right-click on the file name and select **Compile→Compile Selected**. If all goes well, the file highlight marker changes into a green checkmark, and green status messages appear in the **Transcript** pane at the bottom of your window. If errors occur, the file highlight marker changes into a red X, and red error messages appear in the **Transcript** pane.
- If there were errors, double-click on the error messages in the **Transcript** pane starting from the top[4], inspect the code and fix the errors, save the file, and recompile. Repeat until all errors have been eliminated.

The next step is to verify that your entity is in accordance with the test bench. You need to include the test bench files in your design project and recompile the files together.

- Download the test bench file `full_adder_tb3.vhdl` from the course homepage and place it in your project directory.
- Right-click in the project window, select **Add To Project→Existing file...**, and select the test bench file. Click `OK`.
- Right-click in the project file list pane and select **Compile→Compile All**. Correct any errors as before (without modifying the test bench file—see footnote on page 1!).

Once a correct entity has been created, it is time to create the FA architecture. There are many ways to implement the behavior encoded in the Figure-1 truth table. If you cannot decide, we suggest that you start with the *selected signal assignments* shown in the video tutorial (keywords `WITH`, `SELECT`, and `WHEN`) and optionally try alternatives when you have verified that one.

- Double-click on `full_adder.vhdl` again to open the file.
- Add code for the `architecture` of the full adder to the file. Save the file and close it.
- Compile all files as before, and fix any compile-time errors.

When the compile-time errors have been fixed, the design may be simulated to find out if it behaves as expected by the test bench.

[4]As in most software programming, you may view the compilation process as sequential, such that one error may cause or hide errors further ahead in the source code. Starting from the top is usually the quickest way to address all compilation errors.

- Add the *do* file `full_adder_tb3.do` to your project in the same way as you added the test bench file above. (Note that by default, the file selection dialog only lists files with certain filename extensions, and that `.do` is not among thouse extensions. Make sure to select **All Files** to see the *do* file.)
- Select **Simulate→Start Simulation...** A dialogue opens to let you choose what to simulate. Your compiled designs are in the library named `work`. Open that library, select the testbench entity, and click **OK**. The middle pane of the QuestaSim main window splits in two, where the right-hand window shows the signals in the design.
- Right-click on the *do* file in the project file list and select **Execute**. A waveform window opens to show the results of the simulation.

The test bench has been designed to flag all unexpected behavior with messages in the **Transcript** pane of the main window.

- Inspect the **Transcript** pane. If any errors were flagged, study the waveforms and your code in order to identify the error. Edit your architecture source code to correct the error, save the file, recompile, and simulate again. Repeat until all errors have been eliminated.

5 A ripple-carry adder

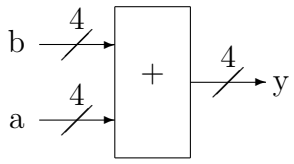


Figure 2: Four-bit ripple-carry adder block.

The next task is to design a four-bit ripple-carry adder using the FA as a component. The desired entity is illustrated in Figure 2. An example of a block diagram of a ripple-carry adder is shown in Figure 3. *Note* that you will need to modify this diagram to have the correct number of FA components and to handle the carry signals in accordance with the entity!

- Create a new design directory and a new project `lab0b` for the new task.
- Download the provided test bench files `ripple_adder_tb3_4_bit.vhdl` and `ripple_adder_tb.do` and add them to your project.

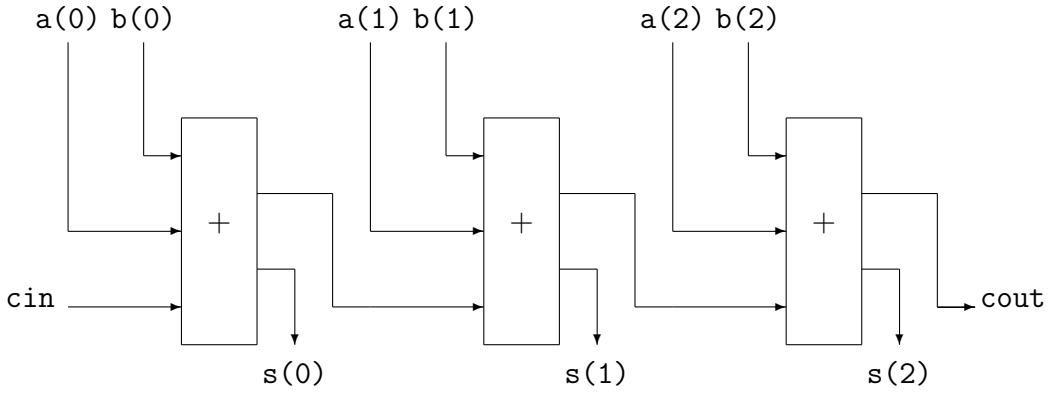


Figure 3: Example ripple-carry adder block diagram, using the full adder described in the previous section.

- Create the entity according to Figure 2. Refer to the test bench to get names and types right: name your entity `ripple_adder_4_bit`, and use the type `STD_LOGIC_VECTOR` for the input and output signals.
- Create the architecture for the ripple-carry adder in accordance with the entity. Use your previously-designed full-adder *as a component*—do not repeat the FA code multiple times in the ripple-carry adder! Use Figure 3 as inspiration and modify as needed.
- Verify the functionality of your ripple-carry adder using the test bench files. Fix bugs (if any).

6 A simple counter

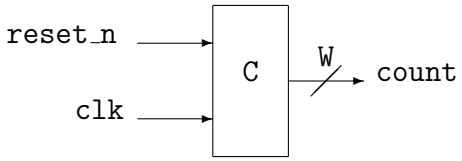


Figure 4: Up-counter block.

The last task of this lab session is to extend your ripple-carry adder from Section 5 to design an up-counter. This block counts the number of pulses at its clock input and increments its output value once per cycle, restarting from 0 after N cycles; N is a design-time parameter.

The entity is illustrated in Figure 4. In addition to the `clk` input and the `count` output, the `reset_n` input is an asynchronous active-low reset signal. When `reset_n` is deactivated, the counter should count `clk` pulses from 0 to $N-1$, triggering on the rising edge of `clk`. When `count` is $N-1$, the next pulse should return the state to 0 for the next lap. The width `W` of `count` is determined by the number of bits necessary to represent the range of values[5].

- Create a work directory `lab0c` and a new project of the same name.
- Create the entity needed for the counter, using the name `counter` and the types `STD_LOGIC` and `STD_LOGIC_VECTOR` for its external signals. Refer to the test bench files `counter_tb3.vhdl` and `counter_tb.do` to get the names and types right. Assume that $N = 13$.

Before you write the code for the counter architecture, spend a few minutes considering a suitable hardware implementation of the counter. Often, these few minutes will help you to quickly find a viable and understandable VHDL-code representation of a block. (Example: a counter must include some means to maintain the current value and some means to calculate the next value. As these would typically be implemented separately—with registers and with adder logic, respectively—it may be good to let the VHDL code reflect this practice and use a separate `process` statement for the registers.) Refer again to the video if you feel unsure about how to represent sequential behavior.

- Create the `architecture` for the counter, using your ripple-carry adder as a component. Use the type `STD_LOGIC_VECTOR` for the internal counting value. Check consistency with the test benches as above. Simulate your design in its test bench to detect bugs (if any). Correct any bugs and re-simulate, following the same general procedure as above.

7 Wrap-up

After completing this lab session, you are expected to be able to carry out the following tasks:

- Launch QuestaSim.
- Create projects, VHDL entities, and VHDL architectures in QuestaSim.

[5] As will be seen in a later lab, it is entirely possible to calculate the necessary width from the value of N. For simplicity, you may use a constant width of 4 here.

- Create simple combinational and sequential digital designs and verify their behavior, using provided test bench files and *do* files.

These skills will be used in subsequent lab sessions in the series. Then, the description of the different steps involved will not be as detailed as here.