Lab 1: Combinatorial adder

Introduction

In this this lab exercise you will learn to specify, simulate and implement combinatorial logic in an FPGA using the Xilinx Vivado design environment. The lab includes several steps and a certain amount of repetition; this is intended to help you become familiar with the FPGA design flow.

In this lab, you will design and implement the following:

- Two-input logic gates (AND, OR, XOR)
- A half adder (using your AND and XOR gates as components)
- A full adder (using your half adder and OR gates)
- A parallel 4-bit adder, using your full adder.

Task 1: Setting up the project

It is assumed that you have already downloaded and set up Vivado, either in the Linux VM on the course website or your own stand-alone installation. Ask the instructor if you need assistance**. The installation is large and time consuming, so make sure it is set up and working well in advance!**

Start by setting up a vhdl director for your course work. In Linux this can be done on the command line by typing:

mkdir vhdl

There should be no spaces in the path names. In your vhdl folder, create a subdirectory for this lab exercise (e.g. ~/vhdl/lab1). To do this in the command line, type:

`cd vhdl` (to navigate to the vhdl subdirectory)
`mkdir lab1`

Once you have a working directory for the exercise, start Vivado and create a new project:

- From the opening window, click 'Create New Project...' to bring up a new project wizard.
- In the wizard, give the project a new name (e.g. Lab1), and point the project location and working directory to the 'lab1' subdirectory that you just created. If you do this, you don't need to create a project subdirectory.
- Create an RTL project, because you are starting the project from the beginning and creating/adding your own sources.
- You don't need to add or create sources in this wizard, but make sure to specify VHDL for both the target and simulator languages
- Similarly, it is not necessary to add intellectual property cores (IP) or user constraints. These can be added later.

- When asked to choose a default Xilinx part, you need to select the FPGA that the design is targeted for. For the FPGA on the Digilent board:

Product category: General purpose
Family: Artix-7
Package: cpg236 (236-pin ball-grid array)
Speed grade: -1
Temperature grade: C
There will be three remaining devices displayed. Choose xc7a35tcpg236-1

When you reach the end of the wizard, review the project summary and click 'Finish' to open the new project. The Vivado IDE will open, and you will be able to use the Flow Navigator (on the left by default) to manage settings and execute the different steps of the design flow.

Task 2: Logic gates

This part of the lab introduces the different steps of the FPGA design flow, including design entry, simulating the design, and implementing it in hardware. You will need to produce code for two types of gates (AND, XOR).

These instructions will take you through the steps for a 2-input AND gate. Once you have successfully finished, you should also write (and test) a separate VHDL for the XOR gate.

Step 1: Creating a new Design entry

When the Vivado IDE is open, you can use the Flow Navigator (on the left by default) to manage settings and execute the different steps of the design flow.

In the Flow Navigator, begin by launching the "Add Sources" wizard under Project Manager. Choose "Add or create design sources" and click "Next". You will then have the option to add files or directories, or create a new file. Choose "Create File". In the pop-up:

- Select source type 'VHDL Module'.
- Give the file a suitable name (for example `my_and`).
- Set the file location "Local to Project" (default) and click "OK".

After clicking "Finish" you will be prompted to define the module properties:

- Default names are given for the entity and architecture names.
- Define the ports. In our case, specify two "in" ports (a,b) and one "out" port (q). You can make changes later in the editor.
- Review your choices and finish.

If you have done everything correctly, you will see a top-level VHDL design (`my_and`) in your Design Sources hierarchy. Open and view the VHDL source, which should contain an entity declaration and an empty architecture. Check that the entity statement has correctly-declared input and output ports (a,b,q), and edit them if necessary.
In the the architecture, write the AND gate functionality using the native VHDL **and** operator. Save the file when you are finished.

Step 2: Simulating your design

The next step is to simulate your gate using a VHDL test bench. A blank VHDL test bench template is available from the Athena coure page (Resources/Downloadable resources/VHDL code). Save the file to your local project directory under a convenient name (for instance, `my_and_tb.vhd`).

In your "Sources" window you will notice that your `my_and` design file is listed under both "Design Sources" and "Simulation Sources". Right-click "Simulation Sources" and select "Add Sources", which will launch a wizard that will allow you to import and add the test bench file as a simulation source.

Open the test bench file in the editor window. Declare your my_and module as a component, and declare std_logic signals "a" "b" and "q". Also, instantiate your my_and component as the "unit under test" (UUT), connecting signals a, b and q with the appropriate ports.

In the stimulus process, provide appropriate inputs to my_and for testing that the gate works correctly. A example of a simple stimulus might look like thisx:

```
a <= '0';
b <= '0';
wait for 10 ns;
a <= '1';
wait for 10 ns;
a <= '0';
b <= '1';
wait for 10 ns;
a <= '1';
wait;
```

There are no clocks in this simple design, but the test bench template includes a simple clock stimulus process anyway. You may choose to ignore or remove this code.

After saving and checking that there are no syntax errors, you can now run the simulation. In the Flow Navigator, locate and launch "Run Simulation" (choosing "Behavioural" simulation). The simulation will compile and run, and XSim will open a new simulation view.

Here you can see the design hierarchy, visible signals, and a waveform viewer. Zoom out and/or scroll through the waveform window to check that your design works correctly.

Step 3: Adding hardware constraints to the design

When you are satisfied with your design, it is time to start implementing the AND gate in the FPGA. The first step is to specify which I/O pins on the device should be connected to ports a, b and q. Under "RTL Analysis" in the Flow Navigator, launch "Elaborated Design" to analyze the design and begin constraint assignment and project planning.

In the “Schematic tab” you will see the RTL schematic of the design you have created (essentially a single AND gate). Above the schematic you should see that the design uses one (logic) cell, three I/O ports and three nets (connections). If you click on the link to the 3 I/O ports, a new window will be launched that will allow you to assign these pins to physical I/O pins on the FPGA.

Expand the list of ports to see the details. The column 'Package Pin' is empty at the beginning; here is where you specify where the I/Os are placed. Look at your Digilent FPGA board to help decide which ones to use. For inputs `a` and `b`, the switches on the bottom are a good choice. For example, SW0 and SW1 are connected to FPGA pins V17 and V16, respectively. For output `q`, choose one of the LEDs above the row of switches. For example, LD0 is connected to FPGA pin U16.

The FPGA I/O pins on the Digilent FPGA board run at 3.3V, so in the “I/O Std” column, choose the LVCMOS33 standard for all pins in your design.

When you are finished, click the "Save" icon at the top of the IDE window to save your design constraints. You will be prompted to provide a name for your constraint file; choose an appropriate name like “My_And”.

Step 4: Implementing the design

There are three main implementation steps:

- Synthesis
- Implementation
- Bitstream generation (for programming)

Launchers for each of these steps are located in the Flow Navigator. You can execute them one at a time, or you can run all in sequence by clicking "Generate Bitstream". If you select a later step in the chain, Vivado will first run all previously uncompleted steps in order.
The final product will be an FPGA configuration file (`my_and.bit`) that you can download to the FPGA.

Before downloading the file to the FPGA, it may be useful to look at some of the synthesis results. In the flow manager, you will find different schematic types. Look at the following two to see the difference.

- RTL schematic: RTL Analysis > Elaborated Design > Schematic
- Technology schematic: Synthesis > Synthesized Design > Schematic

Examine the design summary as well to see how many resources were used:

- Technology schematic: Synthesis > Synthesized Design > Report Utilization

Step 5: Programming the FPGA

Connect the Digilent board to a USB port on your computer with the provided cable and turn on power. Make sure the jumper JP1 (next to the large USB port) is connected across the center two pins to select JTAG programming mode.

Double-click on "Open Hardware Manager" in the Flow Navigator. Near the top there will be an "Open Target" link, and a pop-up menu will let you select "Auto Connect". In the Hardware window you will then see a programming chain, beginning with the computer you are using as host (localhost), followed by the programming device (Digilent/serial #), and then the FPGA (xc7a35t).

Right-click on the FPGA and select "Program Device....". You will see a popup window where you can select the programming (.bit) file, and an optional "debug probes" file (which we will ignore for now).

After you have selected your.bit file, click on the "Program" button. If you have done everything correctly, the configuration file will be loaded to the FPGA. Test your design by by turning your input switches on and off in different combinations and seeing which combinations cause the output LED to light up.

Congratulations! You have now completed your first FPGA design. Once you have completed your AND gate, create and implement a second VHDL moduls for an XOR gate (`my_xor.vhd`). You can copy and modify `my_and.vhd` to do this. It fine to have both files in the same project; simply choose one of the files as the “top module” when you run the implementation.

Task 3: Half adder and Full adder

Now that you are familiar with the design process, close the existing project and create a new project named 'full-adder'. Import the two logic gate files you have already created into the new project.

Create a new top-level VHDL module (full_adder.vhd) with three inputs (a, b, and carry_in) and two outputs (s and carry_out). You will also need to create a second VHDL source (half_adder.vhd), with two inputs (a, b) and two outputs (s, carry).

In your 'half-adder' architecture, declare and instantiate your my_and and my_xor components, and use them to produce the outputs "s" and "carry" as covered in lecture. The output s is an exclusive OR (XOR) of the inputs a and b. carry is the AND of the two inputs.

In the 'full-adder' architecture, you will need to declare and instantiate your half-adder twice (again as covered in lecture). You will need to declare some internal signals in this top-level design. To produce the carry output you will need the logical OR output of the carry bits from the two half-adders. This can be easily done with a single line of VHDL.

Create a new test bench and simulate full-adder, testing test all combinations of a, b, and carry_in. Finally, implement and test your design on the Digilent FPGA developer board. Use three switches as inputs (a, b, carry_in), and two LEDs as outputs (s and carry_out).

Task 4: Parallel adder

In this final part, you will create a parallel adder that adds two 4-bit values, as described in lecture.

Create a new project "adder_4b" with top-level ports:
```
a_in, b_in, s_out : std_logic_vector(3 DOWNTO 0)
carry_in, carry_out : std_logic
```

Import the VHDL modules from your previous design to the project. Instantiate four full-adder components and connect them correctly, using internal signals for the carry chain.

Simulate your design. It is not necessary to test all 512 possible input combinations, but testing a few is important. For very complex designs, full simulations of become impossible, so the designer must choose tests that are likely to expose potential errors. In this case, make sure to choose input examples that test the carry chain.

Implement the design using eight switches (a_in, b_in) and five LEDs (s_out and carry_out) as inputs and outputs. carry_in can be connected to one of the four input buttons on the side of the switches. Check that the adder works correctly.

Show your work to the instructor when you are finished.

Ideas for improvement

The BASYS3 FPGA board has 16 switches, so it is possible to extend your parallel adder up to an 8-bit version. Can you use generics and generate statements to write an *n*-bit adder with flexible width?