LAB 2 – Mapping your Circuit to FPGA

Goals

- Transfer your design idea to the Xilinx FPGA starter board. See your circuit function.
- Learn how to interface to the components on the Starter Board
- Design a 4-bit adder using hierarchical schematics.

To Do

- The first step is to design a simple 1-bit adder circuit.
- In the next step you instantiate 4 copies of this instance to make a 4-bit adder.
- We then describe Xilinx ISE and how to connect it with the FPGA board.
- Finally, we program the FPGA and get the circuit to run on the FPGA board.

Follow the instructions. Paragraphs that have a gray background like the current paragraph denote descriptions that require you to do something.

- To complete the lab you have to hand in a report. The last page of this document has been formatted so that it can be used for the report. Fill in the necessary parts and return the report to your assistant during the following exercise session.

Introduction

In the previous lab we created a small circuit using the Xilinx ISE. We did not, however, see what our circuit did; we just used a kind of drawing program. In this laboratory exercise, we see how the drawing can be transferred to the Starter board and we observe our circuit working.

Our first design was fairly small. In this laboratory, we make something slightly larger. In this exercise, we use a modular technique. We first design a small circuit that is able to add 1-bit numbers. We then combine multiple instances of this small circuit to make a larger 4-bit adder. Such modular design techniques are very important in allowing us to build large and complex circuits.

Binary addition

One of the later lectures is devoted to the design of arithmetic circuits. We give a small summary here about a straightforward method to add two binary numbers. Essentially the binary addition is the same as the decimal addition operation that you are used to. The only difference is that a digit in a binary number can only be 0 or 1, instead of the numbers from 0 to 9.
Consider the simple example given below:

```
  1 1
  8 7 3
+ 3 6 2
  1 2 3 5
```

We start from the right and first add $3 + 2$, which results in $5$. Since this number is less than $10$, we can directly write it as it is. The next digit is $7 + 6$, which results in $13$. Now we cannot directly write the result since it exceeds the largest possible number we can use in a digit. We write $3$ to the result, and have a ‘carry’ that we move over to the next digit. In the third digit, we now have $1 + 8 + 3 = 12$ since we need to also add the carry from the right side. This result also produces a carry, which also gives the final result.

Now let us look at the same addition when expressed as binary numbers.

```
  1 1 1 1
  0 1 1 0 1 1 0 1 0 0 1
+ 0 0 1 0 1 1 0 1 0 1 0
  1 0 0 1 1 0 1 0 0 1 1
```

The principle is the same. We start from the right. Add each digit. $1 + 0$ is $1$, $0 + 1$ is $1$, $0 + 0$ is $0$ and so on. Once we try to add $1 + 1$ we realize that the result $(2)_{10}$ can not be expressed in one binary digit $(10)_2$. So we have one ‘carry’ bit that is moved to the left.

**One bit addition**

Looking at the example, we see that to add two one-bit numbers $A$ and $B$ we would need two outputs. One ‘Sum’ output that is the result of the addition, and an additional ‘CarryOut’ output that indicates that the addition overflowed and produced a carry bit to the right. Such a circuit is called a half-adder.

The problem with the half adder is that it produces a ‘CarryOut’ to the right, but cannot accept a corresponding ‘CarryIn’ from the left. A circuit that can add three inputs $A$, $B$, CarryIn and has two outputs one Sum, and a CarryOut is called a full-adder.

By sequentially connecting the CarryOut signal of one stage to the CarryIn stage of the following digit we can build much larger adders easily.

Fill out the truth table for a full adder circuit given in the report sheet. We use the inputs $A$, $B$, $Cl$ and the outputs are called $S$ and $CO$. Derive the Boolean equations for both outputs and apply what you have learned to come up with a simplified circuit schematic for the full adder circuit.

Now we have everything in place to finish the 1-bit full adder in Xilinx ISE.
Start ISE and start a ‘New Project’ (you can call it Lab2). Follow the instructions in Lab1, create a schematic (you can call it FA) and draw the full adder using 2-input logic gates. Do not forget to save your circuit.

We now start a new schematic that includes four instances of the full adder (FA) that we have just designed. But before we can use the FA like any other logic-gate, we need to create a symbol for it. This is slightly tricky---see Figure 1 for an example.

On the left side of the GUI, select the ‘Design’ tab, so that you can see the design hierarchy. You should be able to see your schematic (in the figure it is named FA) under the FPGA description (xc3s200-4ft256). If you select the schematic, the window just below shows several possible actions. Next step is to expand ‘Design Utilities’, which reveals further actions. The first of these is ‘Create Schematic Symbol’. Run this action by right clicking on the action and selecting ‘Run’ from the context menu. The symbol generation is rapid; you see a message in the console below when symbol is generated.

![Figure 1. Creating a symbol from the schematic](image)

Now we can start a new schematic, and use our newly generated symbol. To start a new schematic you can select

Project->New Source

from the menu.

Start a new schematic (you can call it Adder). You see that the Symbols panel now shows
a new entry that looks like ‘<path/to/your/project/lab2>’. Once you select this entry, you should be able to select the full adder symbol you have just generated. Each full adder can add two 1-bit numbers. So you need to place four of these full-adders in your new schematic.

We have two 4-bit numbers (A and B) in our adder. The output is the sum (S) of these two 4-bit numbers and is 5-bits wide, four Sum bits of the individual full adders plus the Carry Out bit of the final (left most, most significant) full adder. All but the first (right most, least significant) full adders have their Carry In signals connected to the Carry Out of the full adder to their left. The first (right most, least significant) full adder does not have anything to its right, so there is no Carry In entering this full adder. We can connect the Carry In input to logic-0 (or ground, the symbol for this being called ‘gnd’). You can see an example in Figure 2.

![Figure 2. An example schematic for a 4-bit adder](image)

In the description (and the figure) the organization is so that the least significant bit is on the right and the most significant bit is on the left. This is partly for didactic purposes; the organization mimics the way we write numbers on paper. The default symbols created by Xilinx ISE would favor the cells to be placed vertically, you are free to choose whatever arrangement you want, the circuit would still function the same way. However it is good practice to keep to general conventions.

At this point you can actually finish the schematic. If you remember in the last exercise we needed a similar number of connections to the outside world. While this may be acceptable for smaller circuits, this involves a lot of unnecessary work when designing 32-bit wide circuits. (In the end, we actually design a 32-bit processor). Instead of making individual connections we can group the inputs and outputs to a so-called bus.

To draw a bus, you need to first draw a wire, and add an I/O marker to the input and name the port (by going to the select mode, and double clicking on the I/O Marker). For example, you could name it A(3:0). This indicates that your signal A has 4 bits. On the schematic, you see that your wire changed to a somehow thicker line, indicating a bus.
This bus contains actually four separate wires that are called A(0), A(1), A(2), and A(3). If you want to connect to this bus you need to add a tap. You can select the icon from the left side of the drawing area to add a tap. If you need to rotate your bus taps, you can use “Ctrl R” or the “Rotate” icon. Make the tap side (left side in the small bus tap figure shown above) of each bus tap attached to the bus line (thick wire). On the other side of each tap, add a wire to each tap. Each bus tap is related to a bit of the bus and will connect to the logic gates.

You have now indicated that you will tap to the bus, but it is not yet clear which signal you want to tap. You can do by double-clicking on the wire attached to each bus tap to change the name in the requester that comes up (the names will be A(0), A(1), A(2), or A(3) in this specific example). The problem with this method is that you will not see the names of the wires, which might be a problem later on. An alternative is to add the visible name of each wire by clicking on

Add→Net Names

or click the “Add Net Names” icon in the toolbar. If you look at the left hand side of the IDE you see the properties window. It is likely that all of the window is not visible, but when you scroll down you should be able to see the options in Figure 3.
You can simply select ‘Name the Branch’s Net’ and you can either type in the ‘Name’ you want to tap. But there is a more elegant way. First go and select the option ‘Pick up names of bus members by clicking on a branch’. As the name implies, this allows the tool to find out what the net is called once you click on the bus. The cursor changes to display ‘Click on a branch’ and you are able to click on the desired bus. If you also have selected the option ‘Decrease the name’ at the bottom of the properties (as seen in Figure 3), you are able to name the first net (in this case A(3)) by clicking on the appropriate tap, and the name automatically decreases to A(2), and you can continue clicking on the wire taps. The finished bus would look like Figure 4. Note that the name A(3:0) is just an example, you can replace the name as you see fit.
With these hints it should be easy to finish the schematic. There is one little problem remaining. The first (least significant) full adder does not have a carry in signal coming in. If you try to leave the input unconnected, you get an error when completing the schematic. One solution is to tell the schematic editor that we connect this input to logic-0 (or ground). The symbol for this is called ‘gnd’.

Complete the 4-bit full adder schematic and show the completed schematic to an assistant. Attach a print out of the schematic to your report.

**Connecting to the Starter Kit Board**

After successfully drawing the schematic of your circuit, you will implement your design using a field programmable gate array (FPGA). We discuss FPGAs in detail later in the course. But, in general, an FPGA is a reconfigurable chip that allows you to map a given schematic (essentially a netlist of interconnected logic gates) in itself. Depending on the type an FPGA can map anywhere from a few hundred to more than a million gates.

Throughout the exercises, we be using the Spartan 3 Starter Board from Digilent, Inc., which is shown in Figure 5.
This board includes, among other devices, a Spartan 3 FPGA that we can program, some switches, some light-emitting diodes (LEDs), and some other cool features. The printed circuit board seen in Figure 5 has all these blocks electrically connected. The FPGA has 256 pins, some of which are needed for the power connections and other system connections that are necessary for proper operation. More than 170 of the connections of the FPGA can be freely programmed. In the starter board many of these pins are directly connected to one or the other peripheral device. You can learn more about the Spartan 3 Starter Board from:

http://www.digilentinc.com/Products/Detail.cfm?Prod=S3BOARD

The connections of the FPGA are on the bottom side, and they are organized in a grid fashion. For 256 connections a 16 x 16 grid is used. These pins are named with one letter (for the Y coordinate) and a number (for the X coordinate). So the pin A1 is the bottom left corner, A2 is the pin to the left of it and so on.

In this lab we have two four-bit inputs that we can map to the eight switches labeled SW7 to SW0, and a 5-bit output which we can map to the LEDs. From the board document we can find out to which FPGA pins the switches and the LEDs are connected (page 19). If you look on the board you see that the pin numbers are also written next to the name of the LEDs and Switches. For example, the right most switch (SW0) is connected to pin F12.

What we now need to do is to tell the Xilinx ISE which of our inputs and outputs should be connected to which pin exactly. This is done using a User Constraint File (UCF). Typically, a UCF contains also timing related constraints, telling ISE how fast the circuit should be etc., but at this moment we do not need these. We can generate a UCF file the same we generate a new source file by selecting
Project → New Source

and by selecting ‘Implementation Constraints File’ as the Source Type. Once you have generated the new file, you can select this file in the hierarchy browser on the left side. Directly below the hierarchy browser, there is the ‘Processes’ window that lists only an entry called ‘User Constraints’. If you expand this entry you see a further option that says ‘Edit Constraints (Text)’. See Figure 6 for an example.

![Figure 6. Starting the UCF editor](image)

If you double click on this entry you should start a text editor. Using the text editor enter the following constraints.

```
NET "A[0]" LOC = "H13";
NET "A[1]" LOC = "H14";
NET "A[2]" LOC = "G12";
NET "A[3]" LOC = "F12";

NET "B[0]" LOC = "K13";
NET "B[1]" LOC = "K14";
NET "B[2]" LOC = "J13";
NET "B[3]" LOC = "J14";

NET "S[4]" LOC = "P13";
NET "S[3]" LOC = "N14";
NET "S[2]" LOC = "L12";
NET "S[1]" LOC = "P14";
```
NET "S[0]" LOC = "K12";

Each line essentially connects one of your inputs or outputs to a corresponding pin, which in turn connects your circuit to the peripherals on the board once it is downloaded to the FPGA.

**Programming the FPGA**

Now we are almost at the end, what we have to do is to compile (map) our design on to the FPGA. This process consists of several independent steps (we do not discuss them in detail here), but you can use a short cut:

Select your circuit (Adder) in the hierarchy browser, and then going to the ‘Processes’ window below and double click on the ‘Generate Programming File’. The programming file is the binary data that is downloaded to the FPGA. To generate the programming file, all other steps have to be completed. If you are lucky, there should soon be a green check signal next to the ‘Generate Programming File’ entry. This means that everything has compiled correctly. If there are problems try again and then contact an assistant.

Now we have the program file we need to download this information on to the FPGA. Make sure the Xilinx Spartan 3 Starter board is powered on and that the programmer from the board is connected to the USB port of the computer.

Now expand the ‘Configure Target Device’ option just below the Generate Programming File, and double-click on “Manage Configuration Project (iMPACT)”. An iMPACT window opens up with options for configuring the device, as shown in Figure 7.
This is a simple utility that is used to communicate with the board over the USB cable. Now we have to configure the options so that ISE can communicate with the board (there are different cables).

In the iMPACT window, double click on ‘Boundary Scan’, then right click the popped up blue words, “Right click to Add Device or Initialize”, in the right window, and choose “Initialize Chain”.

Xilinx ISE should automatically detect the XC3S200 Spartan 3 FPGA and an “Assign New Configuration File” window pops up. If iMPACT cannot detect the cable, select:

Output → Cable Setup

and make sure that “Digilent USB JTAG Cable” is selected and close the iMPACT window. Repeat the steps above.

What the graphic editor shows are two chips that are connected serially. The first device is the FPGA that we want to program. Once you turn off the power on the FPGA, it will forget the program you have downloaded. Therefore the board has an additional non-volatile memory (an EEPROM that will not lose its content once you power it off). The second device on the chain is this EEPROM. In our exercises we do not use this
capability, therefore we ‘bypass’ this device.

In the open window choose the .bit file (likely lab02_xx.bit). The second window is for the EEPROM, click “ByPass” on this second window. Finally right-click on the xc3s200 FPGA symbol to the left of the window, and select ‘Program’, as shown in Figure 8.

![Figure 8. Xilinx iMPACT programming window](image)

In the Programming Properties pop-up window, **make sure that the Verify box is unchecked**, and click OK. You should see a ‘Program Succeeded’ message. If not, fix your errors and reprogram.

One of the common problems is that jumper M2 on the board (Just above the last X in the Xilinx logo, see Figure 5) is shorted. Make sure that the jumper is not active.

Now you are ready to test your circuit on the board. Use the switches and verify that your circuit functions correctly. Show the working circuit to an assistant. If there are errors, you should find the problem and correct it.

**Last Words**

We have (hopefully) managed to program the FPGA and have translated our design idea from paper, to a schematic, and then to a physical working circuit.

It is quite possible that you have encountered errors during this exercise. The entire process consists of many steps, and at each step there are many parts where you could have made an error. From the lowest level the errors could be:

- You have made a mistake in the truth table.
- Equations were not derived correctly.
- Mistake in the schematic drawing (in either schematic).
- The UCF could have errors
- Problems with the programming cable (USB port not working)
- The configuration switches on the board were not correct
- Board was not powered

There are so many things that could go wrong; it is almost normal that some of you experience problems during the exercise. You should not feel frustrated. It is a rather complex process involving many steps, and it requires some experience to get it right.

When compared to the capabilities of the FPGA, the circuit we have programmed is very small (less than 1% of its resources would be used). For such a small circuit some of the steps of the program may seem a little bit complex. However, the environment is designed for much larger circuits, and some of its ‘quirks’ start to make sense once we start making larger circuits.

We do not use the schematic editor very much. You see that describing circuits using hardware description languages such as Verilog is much more practical for larger circuits. This will be the topic of the next exercise.
Part 1a

Complete the following truth table for the full-adder circuit

<table>
<thead>
<tr>
<th>CI</th>
<th>B</th>
<th>A</th>
<th>CO</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Part 1b

Derive Boolean equations for CO and S (using whatever method you like)

\[
CO = \text{ }
\]

\[
S = \text{ }
\]
Part 1c

Draw the schematic of the full adder circuit according to the equations you have derived.

Part 2

Show an assistant your 4-bit adder circuit (do not forget to attach a print to this report).

Part 3

Show an assistant that your circuit works.

Part 4

If you have any comments about the exercise please add them here: mistakes in the text, difficulty level of the exercise, anything that will help us improve it for the next time.