

# Introduction to Accumulators and FPGAs

---

## Introduction

---

This lab will introduce concepts of arithmetic circuits by introducing adders and accumulators. We will also introduce basic concepts on FPGAs. We will develop an adder by means of hierarchical, iterative design. The final system will be divided into subsystems or modules described on independent VHDL files. We will also expand the concepts learn in previous labs to create, synthesize and download a CPLD project; to FPGAs.

## Goals

---

After completing this lab, you will:

- Be able to design and implement adders and accumulators.
- Learn to use several instantiations of a component declaration.
- Understand D2XL board capabilities.
- Make differences between CPLDs and FPGAs.

## Design Description : Accumulator

---

The accumulator we will design must:

- Have a four bits wide synchronous input.
- Have an eight bit wide output which will be displayed using one out of four seven segment digit displays available in the board.
- Includes reset capabilities.
- Must include a way to signal the system when a new input is ready to be read.

## Design Analysis

---

A basic half adder logic diagram is shown in figure 1, including its equations and truth table. It is called a half adder because it does not include a “*carry in*” within its inputs. Note that the circuit is made of simple XOR and AND gates.

A basic full adder logic diagram is shown in figure 2, including its truth table. A full adder does include the “*carry in*” as an input. It is build by two half adders and an additional OR gate. Note that figure 2 is showing a full 1 bit adder. In order to build full X-bit adders (where X represents any integer) we must put together several 1-bit full adders. There are several ways to do so. In this lab we will follow a “*ripple carry adder*” approach. A full 4-bit adder schema is shown in figure 3.

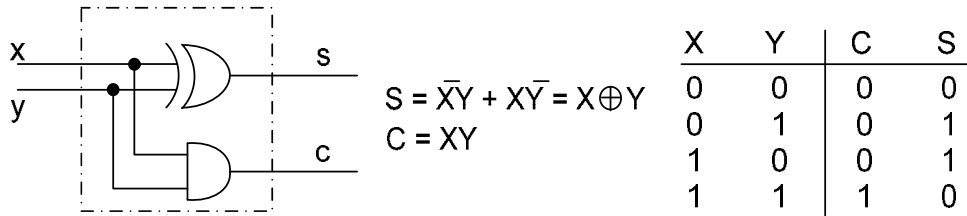


Figure 1; half adder logic diagram, equations and truth table

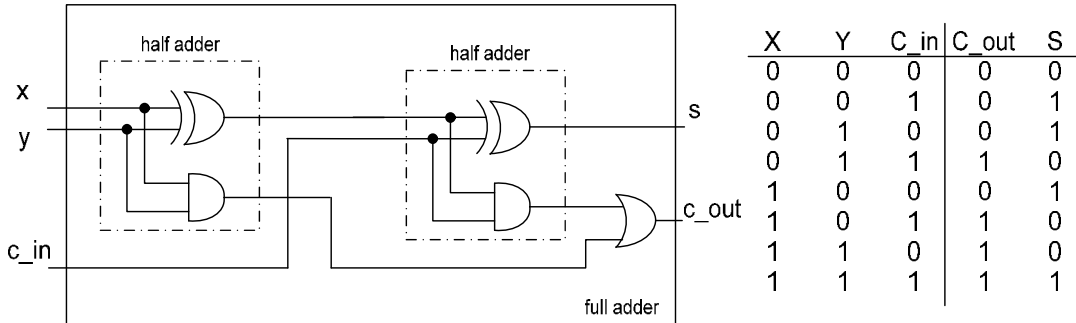


Figure 2; Logic diagram of a 1-bit wide full adder and truth table.

Other techniques will have different configurations corresponding to different trade off between delays, size of the circuit and complexity. For more references please go to chapter 5 of your text book.

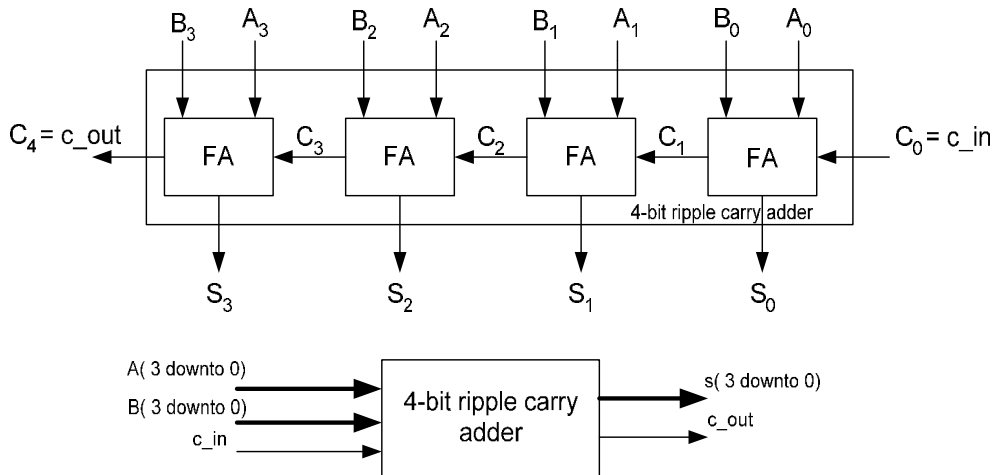


Figure 3; Block diagram of a 4-bit wide ripple carry adder.

## Creating, synthetizing and simulating a Half Adder (HA)

Step 1

As you have done previously, create a new directory for your project with the following format: **C:\ece238\spring2005\student\_name\lab7**. Download the files **half\_adder\_tb.vhd**, **full\_adder\_tb.vhd**, **four\_bit\_reg.vhd**, **accumulator\_tb.vhd**, **disp\_controller.vhd** and **top\_accumulator.ucf** provided on the website and copy them in the newly created directory. Make sure you are saving them without a ".txt" extension but a ".vhd" extension.

1. Open ISE software; Go to **Start Menu** → **Programs** → **Xilinx ISE 6** → **Project Navigator**. (or you can also look for the ISE icon on your desktop)

2. Create counter project; In the Project Navigator, select **File** → **New Project** and setup options as in figure 4. Click **OK**.

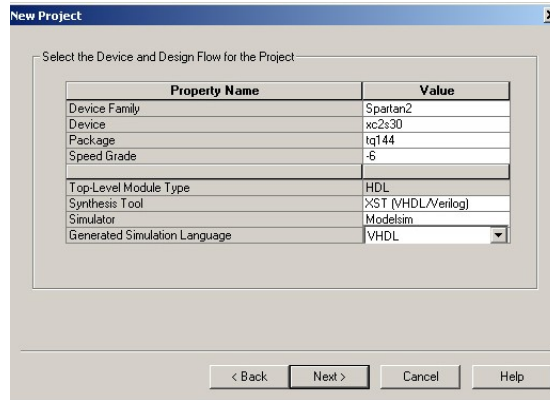


Figure 4; Creation of a project. The format for **Project Location** must be **“C:\ece238\spring2005\student\_name\lab7”**

3. Creating a new source file; Go to **Project** → **New Source**, select **VHDL Module** and enter the name **“half\_adder”** as shown in figure 5. You will get a window as shown in figure 6, to define the VHDL source; fill out the architecture name as **structural** and declare the inputs and outputs of the half adder circuit following the logic diagram in figure 1. Click **Next** → **Finish**. The result will be a new source file where the skeleton is ready for you to fill out with the description of the circuit.

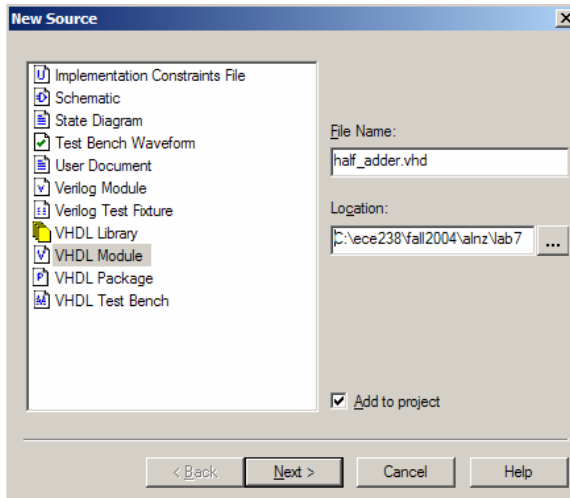


Figure 5.

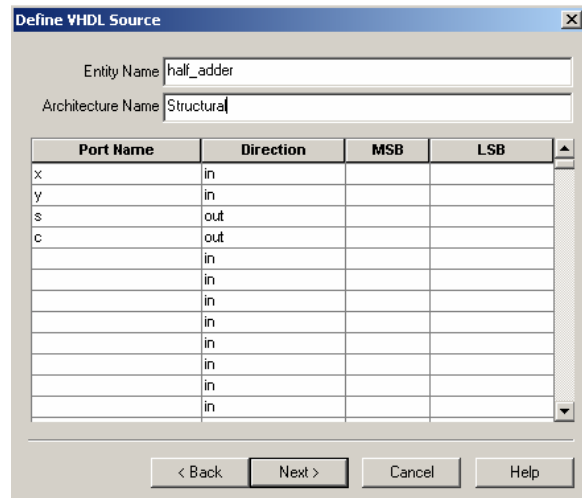


Figure 6.

4. Declaring components; Following logic diagram in figure 1, you will need two kind of gates, a XOR gate and an AND gate. Figure 7 shows each gate and its component declaration. Remember that a component declaration goes between **“architecture structural of half\_adder is”** line and the **“begin”** line.
5. Connecting components; Component instantiation (section where you describe the connections between components) goes between the lines **“begin”** and **“end structural”** in the VHDL description of the system. Follow the logic diagram in figure 1 to describe the connections between the gates and inputs/outputs of the half adder. Remember from last

lab the format for component instantiation is as follows:

```
name_of_instantiation: name_of_component PORT MAP(
    name_of_inputs => ??,
    name_of_outputs => ??,
);
```

Note that by changing the “name\_of\_instantiation” you can use (instantiate) more than one component of the same kind. For instance, if we were to use two XOR gates, we should have two component instantiations; i.e. one with name “xor\_gate\_01” and another with name “xor\_gate\_02”.

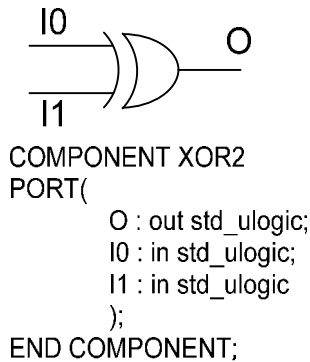


Figure 7; XOR gate and its component declaration.

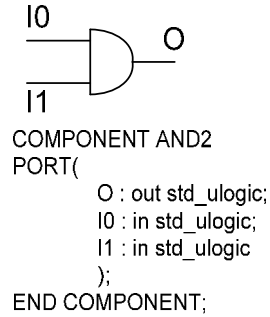


Figure 8; AND gate and its component declaration

Note that in the new VHDL source file created there are 4 lines commented. In our case, we ARE using Xilinx primitive components (XOR and AND gates are Xilinx primitives). As a result we do need to uncomment the last two lines of the comment.

6. Synthesize the project; Highlight the file half\_adder.vhd in the **Sources in Project** window, expand **Implement Design** and double click on **Synthesize** in the **Processes for Sources** window. Make sure there are no errors. Note that you will get a couple warnings stating the gates will be treated as black boxes, which is ok.
7. Add VHDL code to the project; Go to **Project** → **Add source**, select the half\_adder\_tb.vhd (test bench associated to half\_adder.vhd).
8. Run behavioral simulation; Highlight the file half\_adder\_tb.vhd in the **Sources in Project** window, expand **ModelSim simulator** and double click on **Simulate Behavioral Model** in the **Processes for Source** window. You should get a waveform similar to figure 8. Check it against the truth table for a half adder and make sure it complies.

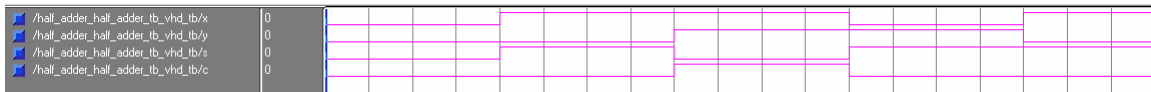


Figure 8; Waveform resulting from half adder simulation

## Creating, synthesizing and simulating a Full Adder (FA)

## Step 2

1. Creating a new source file; Go to **Project** → **New Source**, select **VHDL Module** and enter the name “*full\_adder*” as shown in figure 9. You will get a window as shown in figure 10, to define the VHDL source; fill out the architecture name as *structural* and declare the inputs and outputs of the full adder circuit following the logic diagram in figure 2. Click **Next** → **Finish**. The result will be a new source file where the skeleton is ready for you to fill it out with the description of the circuit.

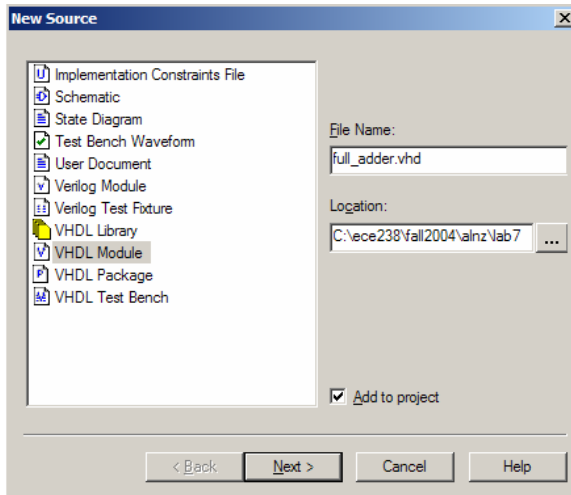


Figure 9

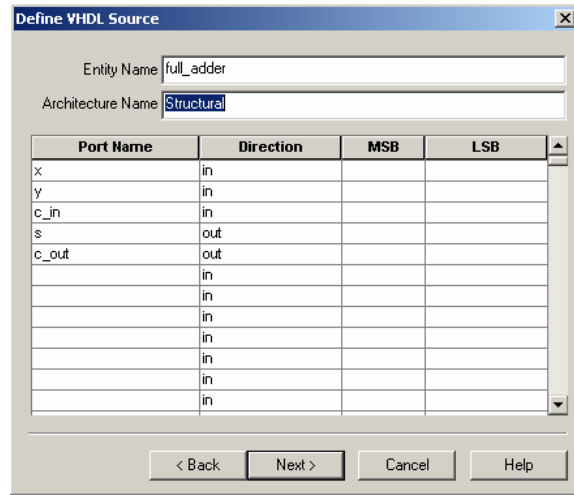


Figure 10

2. Declaring components; Following logic diagram in figure 2, you will need two kind of components, a half adder and an OR gate. Figure 11 shows the OR gate and its component declaration. Remember that a component declaration goes between “*architecture structural of full\_adder is*” line and the “*begin*” line.

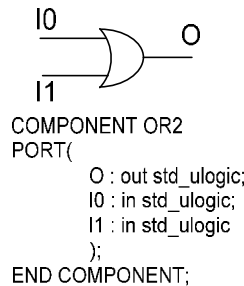


Figure 11; OR gate and its component declaration

3. Connecting components; Component instantiation (section where you describe the connections between components) goes between the lines “*begin*” and “*end structural*” in the VHDL description of the system. Follow the logic diagram in figure 2 to describe the connections between the components and inputs/outputs of the full adder.

Note that in the new VHDL source file created there are 4 lines commented. In our case, we ARE

using Xilinx primitive components (OR gate is Xilinx primitives). As a result we do need to uncomment the last two lines of the comment.

Note that when this file is completed and saved, the file hierarchy under Sources in Project window will change showing that half\_adder system is part of the full\_adder system.

4. Synthesize the project; Highlight the file full\_adder.vhd in the **Sources in Project** window, expand **Implement Design** and double click on **Synthesize** in the **Processes for Source** window. Make sure there are no errors. Note that you will get a couple warnings stating the gates will be treated as black boxes, which is ok.
5. Add VHDL code to the project; Go to **Project** → **Add source**, select the full\_adder\_tb.vhd (test bench associated to full\_adder.vhd).
6. Run behavioral simulation; Highlight the file full\_adder\_tb.vhd in the **Sources in Project** window, expand **ModelSim Simulator** and double click on **Simulate Behavioral Model** in the **Processes for Source** window. You should get a waveform similar to figure 12. Check it against the truth table for a half adder and make sure it complies.

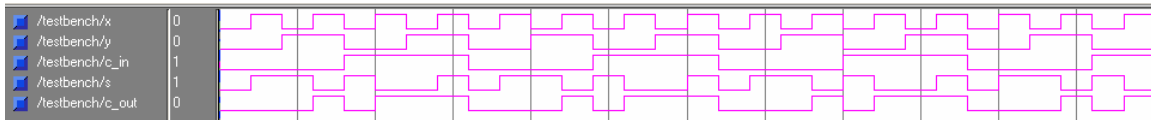


Figure 12; 1-bit wide full adder simulation.

## Creating, synthetizing and simulating a 4-bit Full Adder (FA) Step 3

1. Creating a new source file; Go to **Project** → **New Source**, select **VHDL Module** and enter the name “four\_bit\_full\_adder” as shown in figure 13. You will get a window as shown in figure 14, to define the VHDL source; fill out the architecture name as *structural* and declare the inputs and outputs of the full adder circuit following the logic diagram in figure 3. Click **Next** → **Finish**. The result will be a new source file where the skeleton is ready for you to fill it with the description of the circuit.

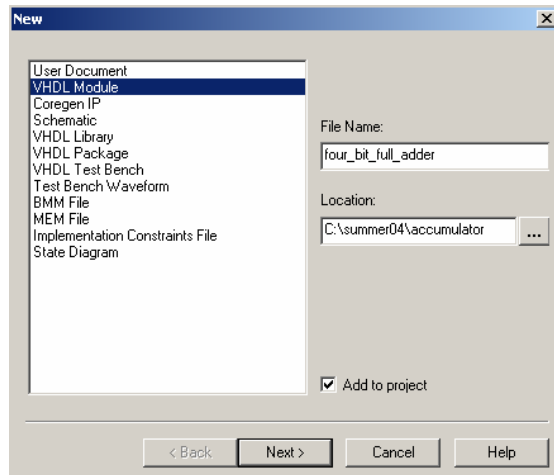


Figure 13

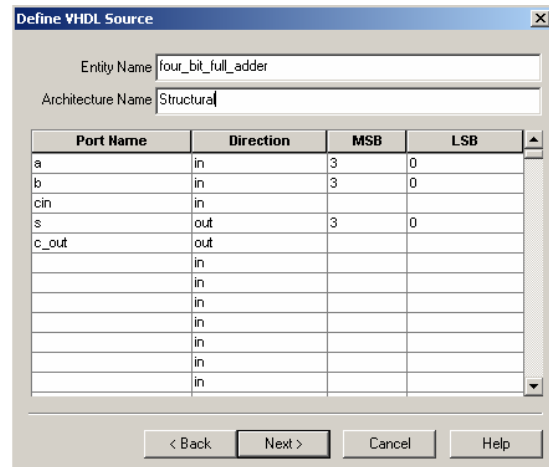


Figure 14

2. Declaring components; Following logic diagram in figure 3, you will need only one kind of

component; a 1-bit full adder. Remember that a component declaration goes between “*architecture structural of four\_bit\_full\_adder is*” line and the “*begin*” line.

3. Connecting components; Component instantiation (section where you describe the connections between components) goes between the lines “*begin*” and “*end structural*” in the VHDL description of the system. Follow the logic diagram in figure 3 to describe the connections between the components and inputs/outputs of the 4-bit full adder.

Note that in the new VHDL source file created there are 4 lines commented. In our case, we ARE NOT using Xilinx primitive components. As a result we do not need to uncomment the last two lines of the comment.

4. Synthesize the project; Highlight the file `four_bit_full_adder.vhd` in the **Sources in Project** window, expand **Implement Design** and double click on **Synthesize** in the **Processes for Source** window. Make sure there are no errors. Note that you will get a couple warnings stating the gates will be treated as black boxes, which is ok.
5. Creating a testbench; Similar as in steps 2 and 3 create a new source for the test bench. Go to **Project** → **New Source**, select **VHDL Test Bench** and enter the name “`four_bit_full_adder_tb`”. Associate the test bench with the `four_bit_full_adder` source file. Click **Next** → **Finish**. The result will be a new source file where the skeleton is ready for you to fill it out with the processes per each signal. Describe the input signals for simulation following the waveform graph in figure 15.

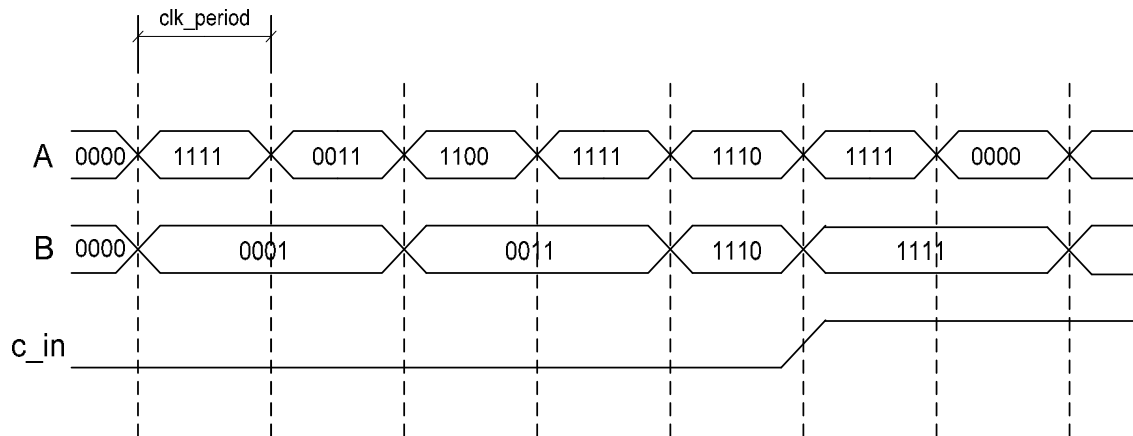


Figure 15; Desired waveform for 4-bit full adder simulation

6. Run behavioral simulation; Highlight the file `four_bit_full_adder_tb.vhd` in the **Sources in Project** window, expand **ModelSim Simulator** and double click on **Simulate Behavioral Model** in the **Processes for Source** window. Check it and make sure it complies with the behavior of a full adder. Note that you can change the radix representation of the numbers in the output waveforms of the simulation.

---

## Creating, synthesizing and simulating an Accumulator

Step 4

1. Add VHDL code to the project; Go to **Project** → **Add source**, select the `four_bit_reg.vhd` (vhdl module).
2. Analyzing `four_bit_reg.vhd` description; Open the file `four_bit_reg.vhd` by double clicking on it under **Sources in Project** window. This file describes the “*register*” block shown in figure 16. When the input “*reset*” is high, the output of the block is “0000”. If a raising edge is detected in the input signal “*clk*”, the block will read its input “*reg\_in*” and write it

down on its output signal “sum\_out”.

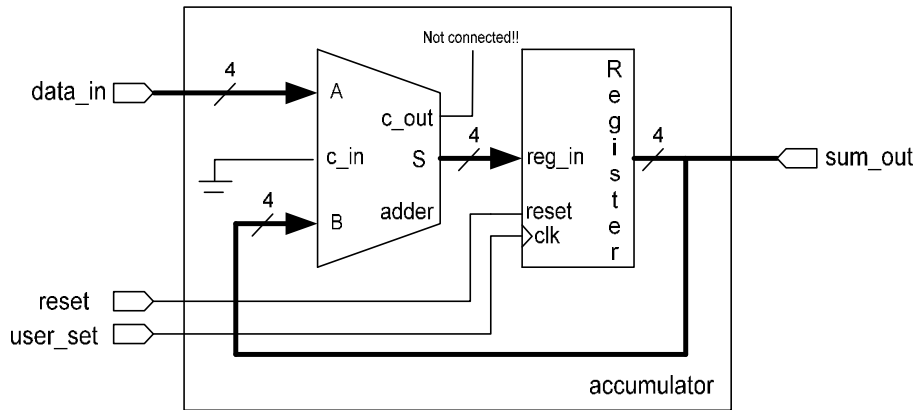


Figure 16; Accumulator block diagram.

3. Creating a new source file; Following the same procedure as in step 1,2 and 3, go to **Project** → **New Source**, select **VHDL Module** and enter the name “accumulator”. Define the VHDL source as a structural architecture and declare the inputs and outputs of the accumulator following the block diagram in figure 16. Click **Next** → **Finish**. The result will be a new source file where the skeleton is ready for you to fill it out with the description of the circuit.
4. Declaring components; Following block diagram in figure 16, you will need two kind of components; a 4-bit full adder and a 4-bit register (the block that you added to your project through the file *four\_bit\_register.vhd* . Remember that a component declaration goes between “*architecture structural of accumulator is*” line and the “*begin*” line.
5. Connecting components; Component instantiation (section where you describe the connections between components) goes between the lines “*begin*” and “*end structural*” in the VHDL description of the system. Follow the logic diagram in figure 16 to describe the connections between the components and inputs/outputs of the accumulator.
6. Synthesize the project; Highlight the file accumulator.vhd in the **Sources in Project** window, expand **Implement Design** and double click on **Synthesize** in the **Processes for Source** window. Make sure there are no errors. Note that you will get warnings due to the “no connection” of the signal “c\_out” in figure 16.
7. Add VHDL code to the project; Go to **Project** → **Add source**, select the accumulator\_tb.vhd (test bench associated to accumulator.vhd).
8. Run behavioral simulation; Highlight the file accumulator\_tb.vhd in the **Sources in Project** window, expand **ModelSim Simulator** and double click on **Simulate Behavioral Model** in the **Processes for Source** window. You should get a waveform similar to figure 17. Make sure you answer the questions corresponding to this step in your final report in order for you to understand how the accumulator works.

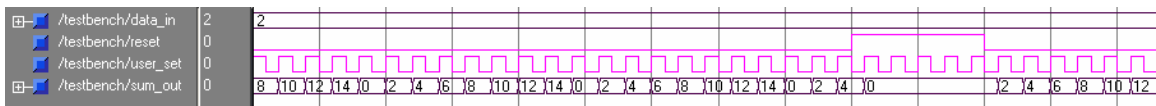


Figure 17; Simulation of the accumulator.

## Implementing the Accumulator

## Step 5

The accumulator will be implemented in the D2XL board following the schematic shown in figure 18.

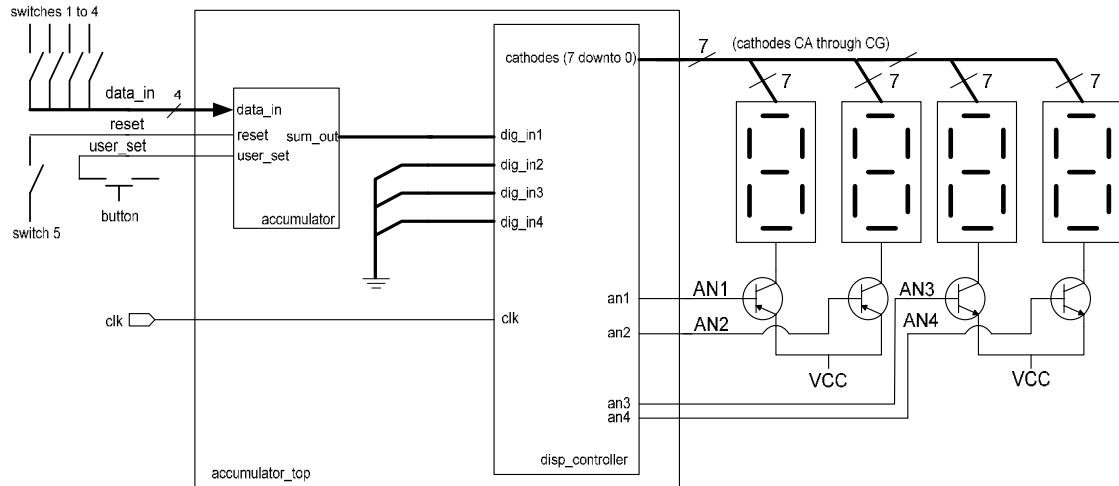


Figure 18; Accumulator implementation schematic.

1. Add VHDL code to the project; Go to **Project** → **Add source**, select the disp\_controller.vhd (VHDL module).
2. Analyzing disp\_controller.vhd description; Open the file disp\_controller.vhd by double clicking on it under **Sources in Project** window. This file describes the “disp\_controller” block shown in figure 18. It is a variation of the display controller we saw in previous labs. Each one of the inputs “dig\_in1” to “dig\_in4” will correspond to the digit displayed at each of the seven segment displays.
3. Creating a new source file; Following the same procedure as in step 1,2 and 3, go to **Project** → **New Source**, select **VHDL Module** and enter the name “top\_accumulator”. Define the VHDL source as a structural architecture and declare the inputs and outputs of the accumulator following the block diagram in figure 18. Click **Next** → **Finish**. The result will be a new source file where the skeleton is ready for you to fill it out with the description of the circuit.
4. Declaring components; Following logic diagram in figure 18, you will need two kind of components; an accumulator and a display controller. Remember that a component declaration goes between “architecture structural of top\_accumulator is” line and the “begin” line.
5. Connecting components; Component instantiation (section where you describe the connections between components) goes between the lines “begin” and “end structural” in the VHDL description of the system. Follow the logic diagram in figure 18 to describe the connections between the components and inputs/outputs of the system.
6. Synthesize the project; Highlight the file top\_accumulator.vhd in the **Sources in Project** window, expand **Implement Design** and double click on **Synthesize** in the **Processes for current sources** window. Make sure there are no errors. Note that you will get a couple warnings stating the gates will be treated as black boxes, which is ok.
7. Add VHDL code to the project; Go to **Project** → **Add source**, select the top\_accumulator.ucf (User constraint file associated to top\_accumulator.vhd).
8. Completing User Constraint File; Highlight top\_accumulator.ucf file under Sources in Project window. Expand **User Constraints**, double click on **Edit Constraints (text)** under **Processes for Source** window. Using table “Pin assignment for D2XL board” provided in the lab website complete the information missing on the top\_accumulator.ucf

file.

9. Configuring generation of programming file process; Highlight top\_accumulator.vhd, right click on **Generate Programming File** in the **Processes for Source** window. Select the **Readback Options** tab. Click the check box to the right of **Create ReadBack Data Files** (see figure 19). Then click the check box to the right of **Create Mask File**. Click Ok.
10. Create programming file; Highlight top\_accumulator.vhd file in the **Sources in Project** window and double click in **Generate Programming File** in the **Processes for Source** window.

Note: At this point make sure you have the power to the FPGA board switched on (turn the power strip and make sure the A/B switchbox on the top of the computer is set to FPGA).

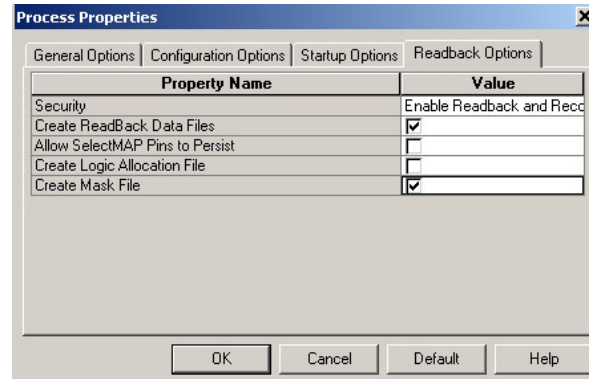


Figure 19. Readback options

11. Launching iMPACT; Highlight top\_accumulator.vhd in **Sources in Project** window. Expand **Generate Programming File** tasks on the **Processes for Source** window and double click on **Configure Device (iMPACT)**. Choose **Configure Devices** → **Next** → **Boundary-Scan Mode** → **Next** → **Automatically connect to cable and identify Boundary-Scan chain** → **Finish** → **Ok**. An **Assign New Configuration File** window must be the result of this operation.
12. Programming the device; Browse for the file top\_accumulator.bit in your project directory using the resulting window from previous step. Click **Ok**. Click on the Xilinx chip to highlight it and go to **Operations** → **Program**. Click **Ok**.
13. Demonstrate your system to the T.A.

---

## Resources Analysis

## Step 6

---

1. Opening Floorplanner; Highlight top\_accumulator.vhd in **Sources in Project** window. Expand **Implement Design** task on the **Processes for Source** window. Expand **Place & Route** under **Implement Design** task and double click on **View/Edit Placed Design (Floorplanner)**. You will get a window similar to figure 20. This is a graphical representation of how the project was developed onto the chip. What you are looking at is an overhead view of the device. The outer ring is the input and output blocks (IOBs). Just as with CPLDs, this is how the device communicates with the outside world. All of the inner boxes are configurable logic blocks (CLBs). This is where all the data processing actually occurs.
2. Make sure that the Toggle Labels, Toggle Rubberbands and Toggle Resource Graphics buttons are selected as in figure 21.

3. Zoom in on the chip and click one of the CLBs. You can see how this CLB is connected in your project. By double clicking, you can see how the whole project is interconnected.

---

## Final Task

## Step 7

---

1. Design a system: Design an accumulator with an 8-bit wide output. Display the output using two seven segment displays instead of only one.

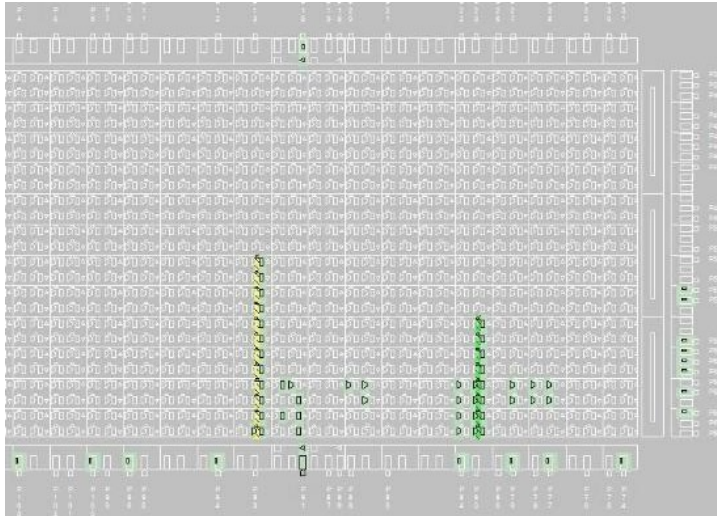


Figure 20. Floorplanner view

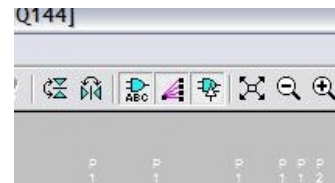


Figure 21

---

## Bonus Activity

## Step 8

---

1. Describe the accumulator using behavioral description. It should be possible to do so in a single file. Implement it in the FPGA board.

This document was created with Win2PDF available at <http://www.daneprairie.com>.  
The unregistered version of Win2PDF is for evaluation or non-commercial use only.