

# Introduction to Counters and CPLDs

---

## Introduction

---

This lab will introduce concepts on digital counters, modular design and explore more capabilities of the XCR board. The systems we will implement are divided into subsystems or modules described using independent VHDL files. We will use the concepts learned in previous labs to create, synthesize and download a project to the CPLD.

---

## Goals

---

After completing this lab, you will be able to:

- Create a project that utilizes multiple VHDL files.
  - Route signals in and out of the device to facilitate I/O.
  - Design and implement counters.
  - Understand clocks and 7-segment display circuits on the XCR board.
- 

## Design Description : Counter

---

The counter we will design must:

- Have a pause, reset and increasing count state.
  - Count from 0 to a maximum of 15.
  - Display its count using the 7-segment display provided on the XCR board.
- 

## Design Analysis

---

### Counter

The counter we were asked to design is more complex than the one you saw in the lecture notes. It has three states and is represented by the state diagram in figure 1. A proposed block diagram is shown in figure 2.

### 7-Segment display *(from XCR datasheet, Digilent Inc.)*

From the lecture notes it is understood that we need a display controller to implement the translation table (figure 2.d from lecture notes) and the timing diagram (figure 3 from lecture notes) for the dual-digit seven segment display on the XCR board. A proposed block diagram of such a system is given in figure 3.

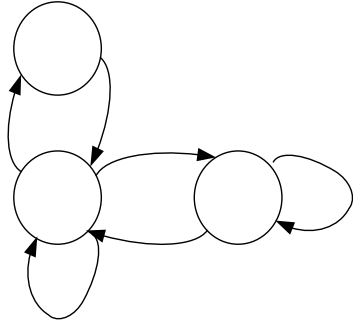


Figure 1; three state counter

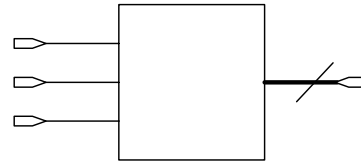


Figure 2; counter block diagram

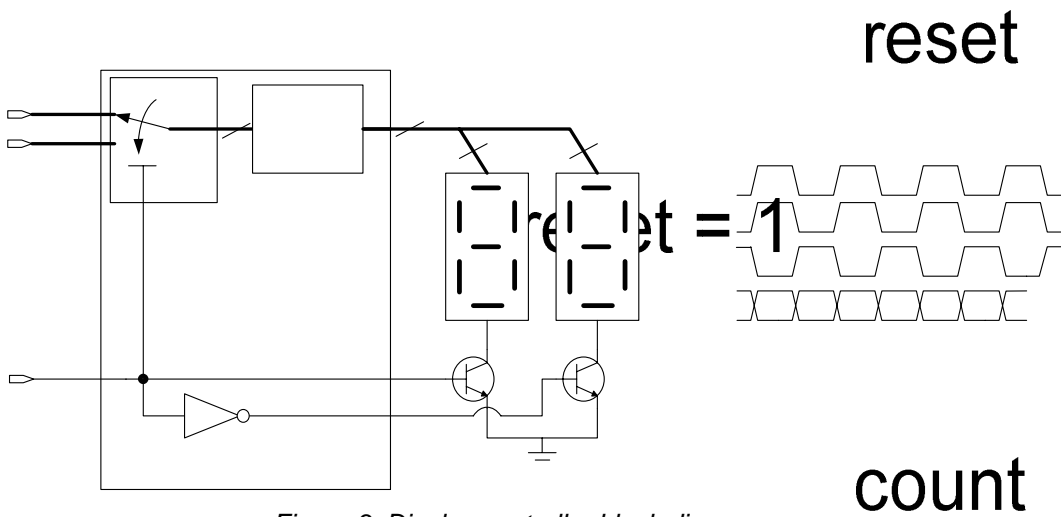


Figure 3; Display controller block diagram

The design of the whole system will follow a modular scheme. We will first describe and simulate a counter, then a display controller, and finally the whole system. A block diagram of the whole system is shown in figure 4.

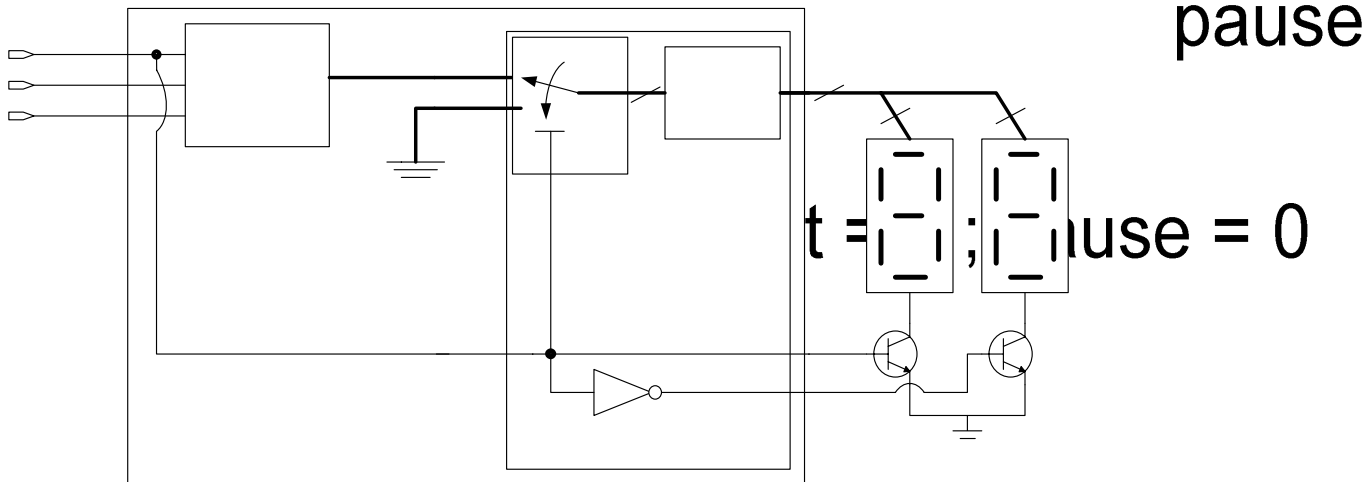


Figure 4; Complete system to be implemented in this lab

## Start Project Navigator and Create the Counter Project

### Step 1

As you have done in previous labs, create a new directory for your project with the following format: **C:\ece238\Fall2005\student\_name\lab6**. Download the files **counter.vhd**, **counter\_tb.vhd**, **disp\_controller.vhd**, **disp\_controller\_tb.vhd**, **disp\_controller.ucf**, **converter.vhd**, **inv\_v.vhd**, **selector.vhd**, **top\_counter.vhd** and **top\_counter.ucf**, all of them provided at the lab website. Save them in the directory you just created. Make sure you are saving the files with the right extensions.

1. Open ISE software; Go to **Start Menu** → **Programs** → **Xilinx ISE 7.1i** → **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 follow the same procedure as in previous labs. Setup project options as in figure 5. Click **OK**.
3. Add VHDL code to the project; Go to **Project** → **Add source**, select the counter.vhd (vhdl module) and counter\_tb.vhd (test bench associated to counter.vhd).

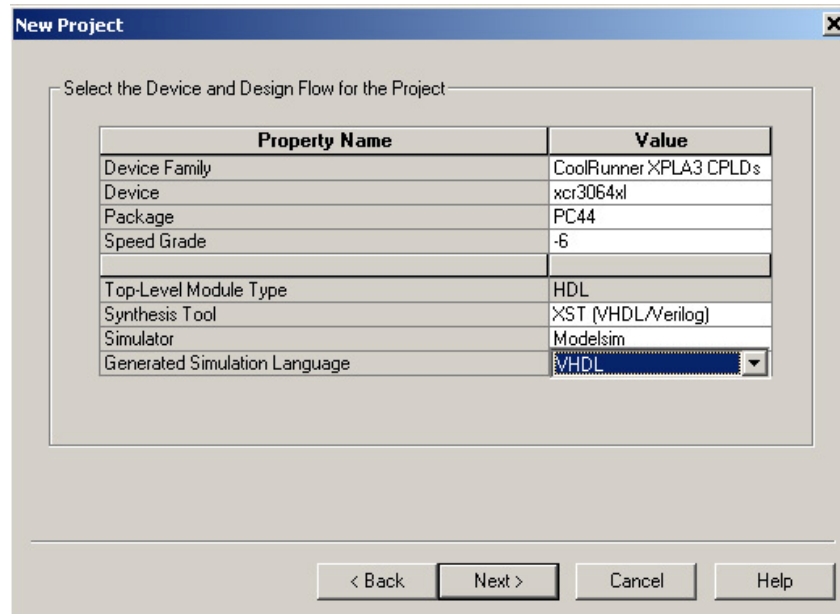


Figure 5; Creating the project.

## Synthesize and Simulate the Counter

### Step 2

1. Synthesize the project; Highlight the file counter.vhd in the **Sources in Project** window and double click on **Synthesize** in the **Processes for Source** window. Make sure there are no warnings or errors.
2. Create a testbench; Open the file counter\_tb.vhd by double clicking on it under the **Sources in Project** window. The skeleton for the testbench is already there. There are empty processes for each of the signals. Complete the lines missing such that **reset** and **pause** capabilities are tested. Hint: draw the waveforms you think are necessary before writing down the code.

Note: An example of wave forms is shown in figure 6. Only the pause capability is tested. The wave form shows that when signal “*pause*” is high, the count doesn’t change.

3. Run behavioral simulation; Highlight the file counter\_tb.vhd in the **Sources in Project** window and double click on **Simulate Behavioral VHDL Model** in the **Processes for Sources** window.

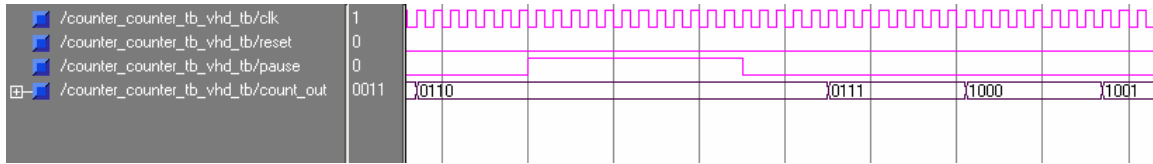


Figure 6; Simulation output.

## Adding the Display Controller to the Project

## Step 3

1. Add VHDL code to the project; Go to **Project** → **Add source**, select the disp\_controller.vhd, converter.vhd, inv\_v.vhd, selector.vhd (vhdl modules) and disp\_controller\_tb.vhd (test bench associated to disp\_controller.vhd).
2. Analyzing controller description; open the file disp\_controller.vhd by double clicking on it under **Sources in Project** window. Note that the description is structural and is not complete.
3. Declaring components or subsystems; every component or subsystem used in the description of a system using structural level of abstraction, must be declared. The display controller we are implementing has three components: a selector, a converter and an inverter. The declaration area for components is under the comment – *Components declarations* – in the disp\_controller.vhd file. Use the inverter declaration as an example, to complete the declarations of the selector and the converter.
4. Components instantiations; once the components are declared, they must be instantiated. This means the connections between components and/or signals inside the system must be declared. The declaration area for components instantiations is under the comment – *Components Instantiations* – in the disp\_controller.vhd file. Use the inverter instantiation as an example. Use the block diagram in figure 5 as a map to complete the connections.
5. Synthesize the project; Highlight the file disp\_controller.vhd under **Sources in Project**, and double click **Synthesize** under **Processes for Source** window. Make sure the process doesn’t generate any error. If it does, read the comment under the **Console** window. It will specify the line where the error is and the reason for it.
6. Simulate the project; Highlight the file disp\_controller\_tb.vhd under **Sources in Project**, expand **ModelSim Simulator** and double click **Simulate Behavioral model** under **Processes for Source** window. Make sure the simulation produces a waveform similar to figure 7.

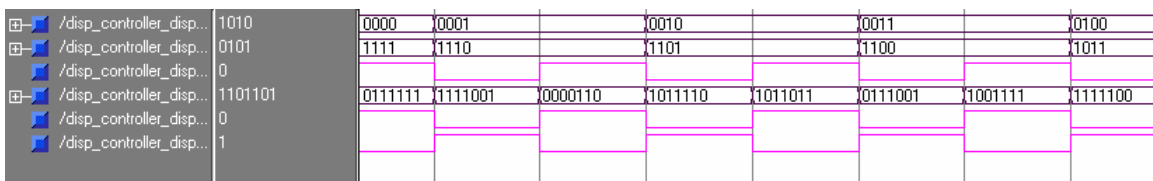


Figure 7; Display controller simulation

## Implementing and testing the Display Controller

## Step 4

1. Add VHDL code to the project; Go to **Project** → **Add source**, select the disp\_controller.ucf (User constraint file associated to disp\_controller.vhd).
2. Completing User Constraint File; highlight disp\_controller.ucf file under Sources in Project window. Double click on **Edit Constraints (text)** under **Processes for current sources** window. Using table 3 of the XCR datasheet complete the information missing on the disp\_controller.ucf file.
3. Create programming file; Highlight disp\_controller.vhd file in the **Sources in Project** window expand **Implement Design** and double click in **Generate Programming File** in the **Processes for Source** window.
4. Launching iMPACT; Highlight disp\_controller.vhd in **Sources in Project** window; Expand **Implement Design** and then **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.
5. Programming the device; Browse for the file disp\_controller.jed 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**. Figure 8 shows the result of this operation and the options that must be selected. Click **Ok**.

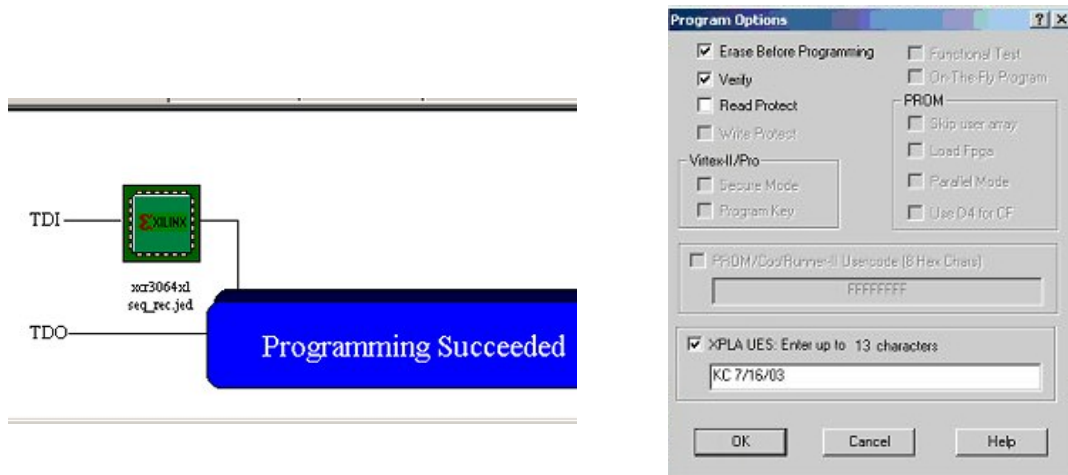


Figure 8; iMPACT window and options for CPLD configuration.

6. Test the system; Switches 1 to 4 are the input MSB while the switches 5 to 8 are the input LSB. Both inputs are a binary representation of the number we want to display in the dual-digit 7-segment display (figure 9). Change the switches 1 to 8 and observe the changes in the displays.
7. Test the clock; The XCR board provides a user-adjustable oscillator that can produce a clock signal in the 0.5 to 4 KHz range. The frequency is fixed by a 15-turn precision potentiometer (variable resistor) that can be adjusted from 0 to 500K ohms. Vary the value of the potentiometer as shown in figure 10 (slowly please!!!). Explain to your T.A what is happening (Hint: remember from lecture notes, the dual-digit seven segment display is multiplexed in time).

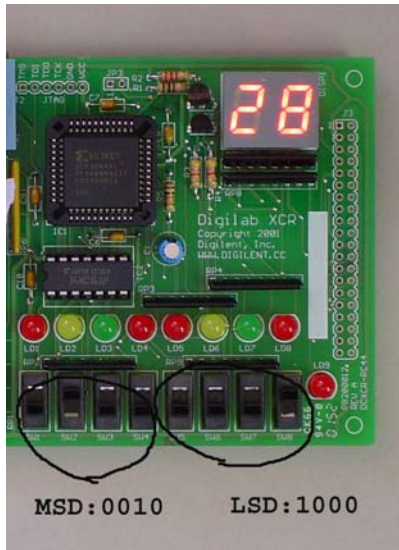


Figure 9; Display controller test

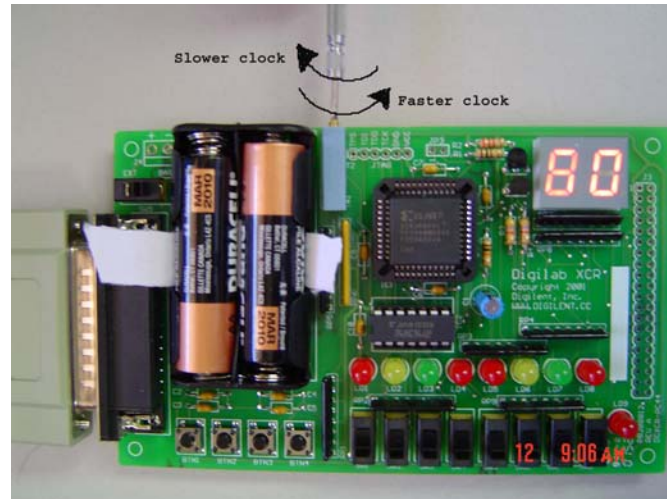


Figure 10; Clock setup by moving potentiometer

## Integrating the System

## Step 5

Now we will create a VHDL file to integrate the counter and the display controller in a single system (figure 4).

1. Add top\_counter.vhd code to the project; Go to **Project** → **Add source**, select the top\_counter.vhd (vhd module).
2. Analyzing system description; open the file top\_counter.vhd by double clicking on it under **Sources in Project** window. Note that the description is structural and is not complete.
3. Declaring components or subsystems; the system we are implementing has two components: a display controller and a counter. The declaration area for components is under the comment – *Components declarations* – in the top\_counter.vhd file. Complete the component declarations as you did for the controller display system.

Note that the display controller subsystem has several subsystems within it, which is reflected in the hierarchical way the files appear in the Sources in Project window. Since the display controller subsystem was already designed and synthesized we don't need to care about what is inside of it. We can work with it as a black box.

4. Components instantiations; once the components are declared, they must be instantiated. This means the connections between components and/or signals inside the system must be declared. The declaration area for components instantiations is under the comment – *Components Instantiations* – in the top\_counter.vhd file. Use the inverter instantiation as an example. Use the block diagram in figure 4 as a map to complete the connections.
5. Synthesize the project; Highlight the file top\_counter.vhd under **Sources in Project**, and double click **Synthesize** under **Processes for Source** window. Make sure the process doesn't generate any error. If it does, read the comment under the **Console** window. It will specify the line where the error is and the reason for it.

---

## Programming the system into the CPLD

## Step 6

---

1. Add VHDL code to the project; Go to **Project** → **Add source**, select the top\_counter.ucf (User constraint file associated to top\_counter.vhd).
2. Completing User Constraint File; highlight top\_counter.ucf file under Sources in Project window. Double click on **Edit Constraints (text)** under **Processes for Source** window. Using table 3 of the XCR datasheet complete the information missing on the top\_counter.ucf file.
3. Create programming file and Launching iMPACT; Highlight top\_counter.vhd in **Sources in Project** window; Expand **Implement Design** and then **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.
4. Programming the device; browse for the file top\_counter.jed 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**. Figure 8 shows the result of this operation and the options that must be selected. Click **Ok**.

---

## Project Testing

## Step 7

---

1. Wait until the display shows a count different than 0 and apply the reset signal (set reset signal to 1). Make sure the system goes to reset state (Displays must show “00”). The system should reinitiate the count as soon as you set the reset signal to 0 again.
2. Wait until the display shows a count different than 0 and apply the pause signal (set pause signal to 1). Make sure the system stays in the same count until you set pause signal to zero again. Demonstrate your system to the T.A.
3. Testing CPLD memory capacity; Switch the power off and on to the power strip. The program should remain in the CPLD and the count will restart as soon as you power up the board.

---

## Final Task

## Step 8

---

1. Design a system: Design a system able to count from zero to 99. The display of the count will use both seven segment displays (note that previous system was only using one display while the second one was set up to zero).

---

## Bonus Activity

## Step 9

---

1. Modify your counter, such that it can be told to have an increasing and a decreasing count.