

# Introduction to Sequence Detectors and CPLDs

---

## Introduction

---

In this lab we will review the steps to create and compile a project on programmable logic using ISE and we will introduce the required steps to download the project into a chip. The programmable device you are you going to use is the Coolrunner CPLD.



In front of you is a black plastic board with two circuit boards on it. There should be labels. One should say FPGA and the other CPLD. We are going to use the CPLD board today.

We are using a Coolrunner XPLA CPLD mounted on a Digilent demo board like the one in the picture above. The CPLD is the square 44-pin chip in the middle of the board. Locate the power strip for the two boards and switch the power on. If the CPLD was previously programmed, you may see some lights begin to flash. These power strips should be left off when the boards are not in use to avoid unnecessary wear and tear. Next; locate the A/B switch box on top of the computer and ensure it is set to CPLD.

---

## Goals

---

After completing this lab, you will be able to:

- Perform the basic operations to create, synthesize, simulate and download a project into a CPLD.
- Recognize basic characteristics of a CPLD
- Identify capabilities of XCR development board
- To use behavioral level of abstraction to describe a sequence detector.

---

## Design Description : Sequence Detectors

---

Sequence detection is the act of recognizing a predefined series of inputs. Note that this is a sequential circuit; therefore we'll need a clock. Since the XCR board we are working with doesn't include a clock, we will use the slowest clock we have: your finger!

Create a sequence detector using VHDL that has the following characteristics:

- Serial input X
- The sequence to be detected is 10010.
- Five bits wide output Z with value 00000 when no sequence is detected, 00001 when the first bit of the sequence is detected, 00010 for the second bit, 00100 for the third bit, 01000 for the fourth and 10000 when the whole sequence is detected.
- If input Y is one, the system is reset to output 00000.

## Design Analysis

The sequence detector created here will detect the sequence 10010. As seen in classes the design must start with a state diagram as shown in figure 1.

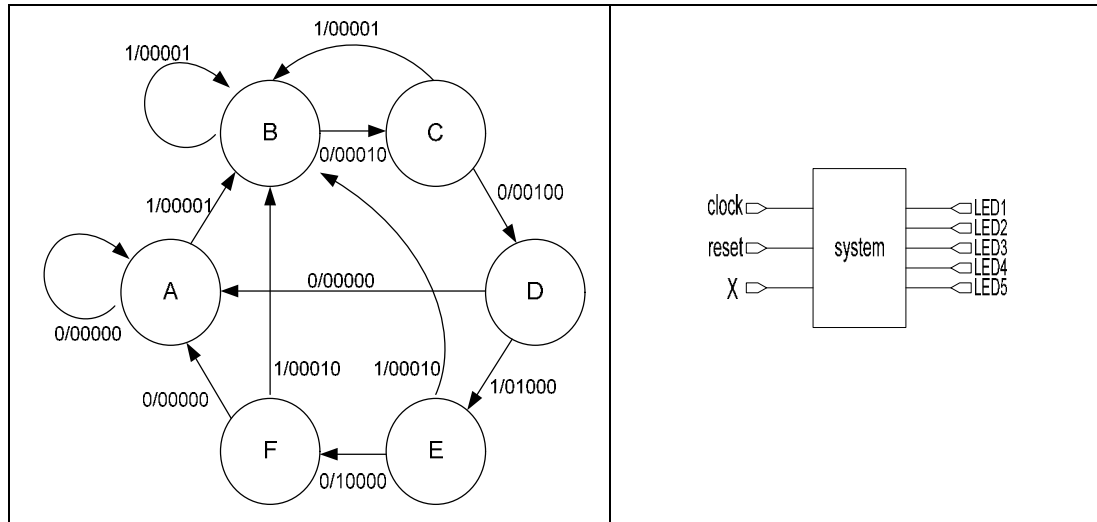


Figure 1; State diagram

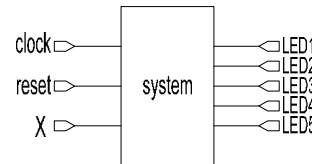


Figure 2; Block diagram

A basic block diagram of the sequence detector is shown in figure 2. A basic introduction to CPLDs can be found in the lecture notes of this lab. Make sure you go through it since you will need that information to answer some of the questions in this lab.

## Start Project Navigator and Create the Project

### Step 1

1. Acquire files for the project; As you have done previously, create a new directory for your project with the following format: **C:\ece238\spring2005\student\_name\lab5**. Download the files sequence.vhd, sequence\_tb.vhd and sequence.ucf provided at the lab website and save them in your new directory. Make sure you are saving them without a ".txt" extension but a ".vhd" or ".ucf" extension.
2. 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)
3. Create the **lab5** project; In the Project Navigator, select **File** → **New Project** and setup options as shown in figure 3. Click **Ok**.
4. Setup CPLD options; The next window you'll get is shown in figure 4. Note that it is essential to choose the correct CPLD part number for the project to work. There are ways to change this part number after creating the project, but it is just easier to choose the correct one at a first time.
5. Add VHDL code to the project; Click **Next** on the following window (figure 5). A new window as shown in figure 6 will open. Click on the **Add Source** button and browse to the files sequence.vhd (VHDL Design file) and sequence\_tb.vhd (VHDL Test Bench File). If for either reason you missed the windows we just described, you can still add the source files as you did in previous labs: Go to **Project** → **Add source**, select the sequence.vhd (vhdl module) and sequence\_tb.vhd (test bench associated to sequence.vhd).

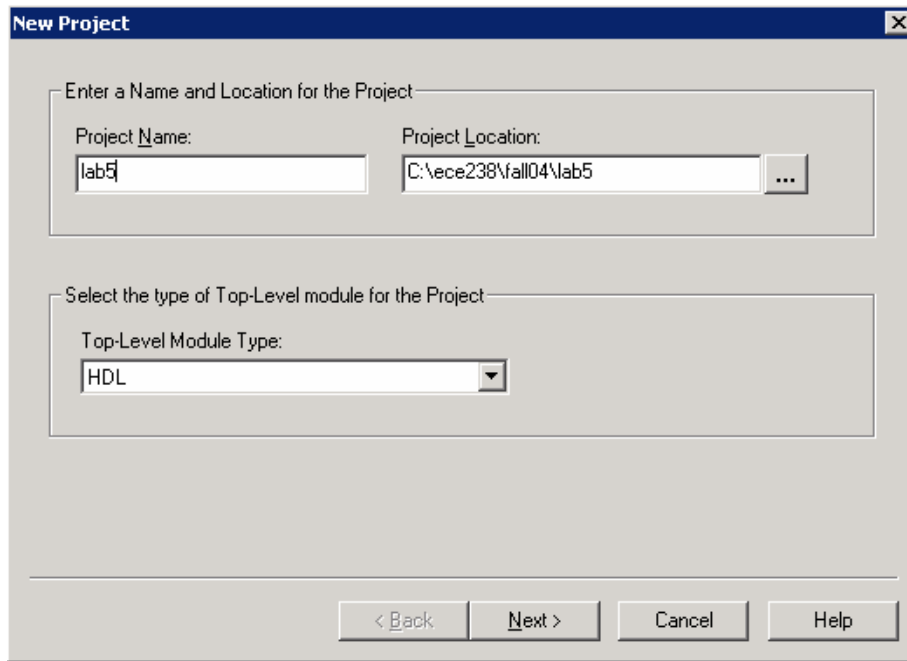


Figure 3. Creation of a new project.

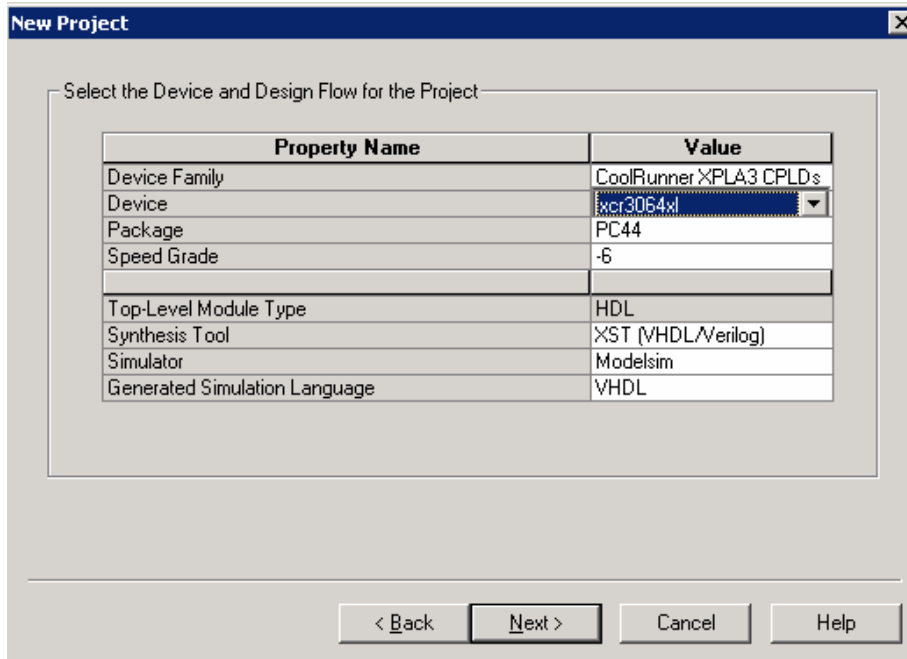


Figure 4: Device Settings

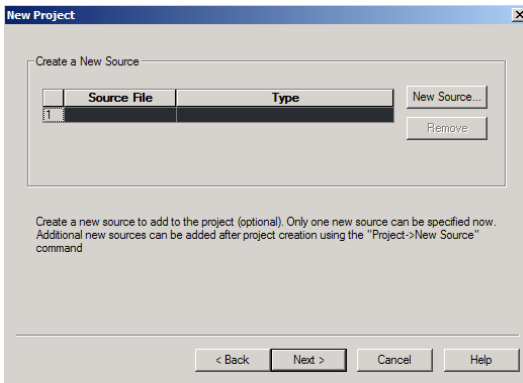


Figure 5. Creating new sources

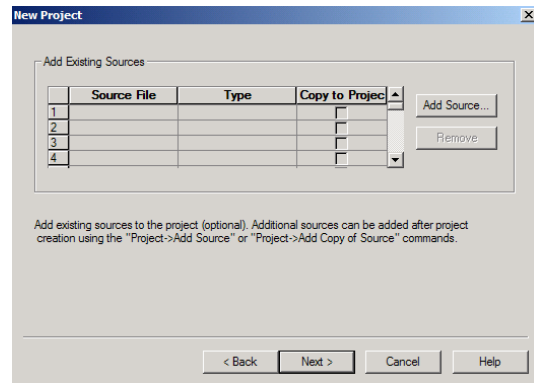


Figure 6. Adding existing sources.

## Simulate the system

## Step 2

1. Synthesize the project; Highlight the file **sequence.vhd**. In the **Processes for Source** pane expand **Implement Design** and double click on **Synthesize - XST**. Make sure there are no warnings or errors.
2. Run behavioral simulation; Highlight the file **sequence\_tb.vhd** and double click on **Simulate Behavioral Model** under **ModelSim simulator** in the **Processes for Source:sequence\_tb.vhd** pane.
3. Verify the operation of the sequence detector by looking at the input and output signals in the **Wave** window (figure 7) from the simulator.

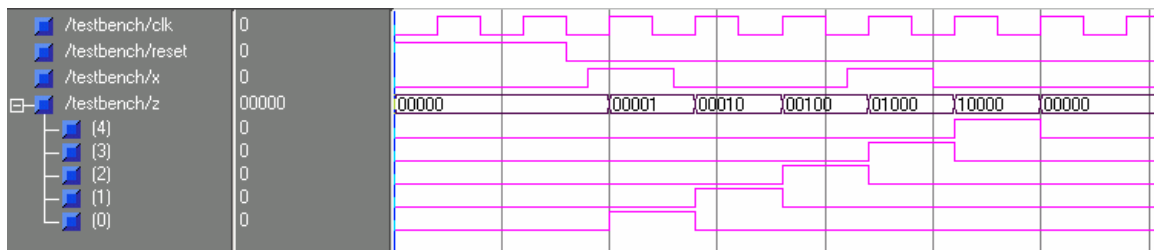


Figure 7: Simulation Waveform

**Note:** This current simulation says that if the required sequence is accompanied with a low reset and rising clock edges, the result will be a high output on Z. It says nothing about what will happen given any other set of inputs. Only one bit of the output Z is high at any time. Each bit in of Z represents a state. When only one state is allowed to be high at once, the design scheme is called *one hot*.

**Note:** Z is a bus; to expand the bus into its individual signals click on the “+” symbol by the side of the signal’s name. Also; sometimes it is useful to change the radix of the signal. Right click on the bus Z and change the radix of the signal to decimal or hexadecimal.

## Explore a Different Input Case Through Simulation

Step 3

1. Open the test bench; Double click on the file `sequence_tb.vhd` in the **Sources in Project** window to open the file.
2. Modify input sequence; Modify lines below the comment – *Following lines specify the input sequence X* – so that the input sequence is now 001110010.

## Implement the Design on the XCR Board

Step 4

In this section we will map the three inputs of the project to three switches on the board and the five outputs to five LEDs. The chip on the XCR board is a Coolrunner XCR3064xl with 44 pins of which we must choose 8 by looking at the data sheet. The following is a flow diagram of what happens when you implement the design.

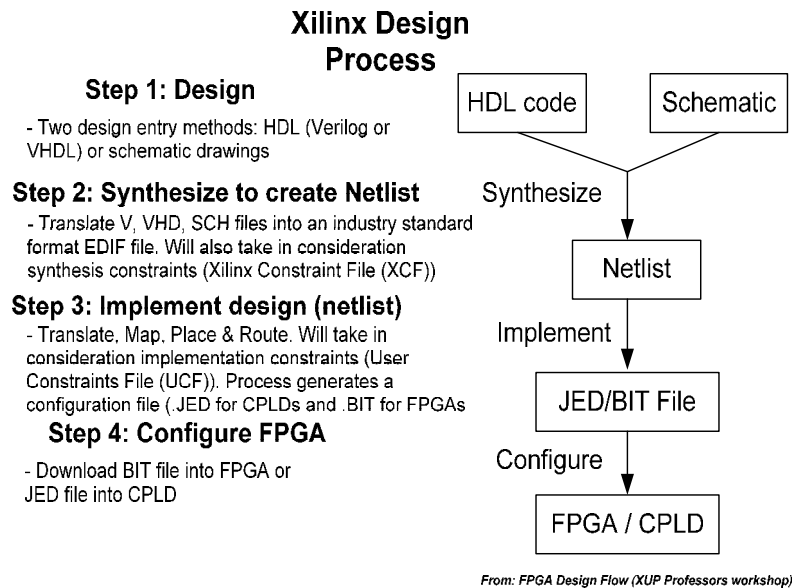


Figure 8: Design Process Flow Diagram

1. Finding CPLD pins available; Look at the board's datasheet (it can be download from the lab website) and find table 3. Fill out the table in the Laboratory section of the worksheet with the appropriate pin numbers.
2. Creating a User Constraints file; Download the file **sequence.ucf** from the lab website and complete it using the information from table 3. Make sure you download the file with extension ".ucf" and NOT ".txt". You can edit the file using any text editor. The extension for this file is **UCF**, this stands for User Constraints file. This file is used to specify which pins on the CPLD chip will be used for the inputs and outputs of the design
3. Adding the UCF file to your project; Go to **Project** → **Add Source** to add the **sequence.ucf** file to your project. Choose to associate the UCF file with the **sequence.vhd** source and click **Ok**. Refer to Figure 9.

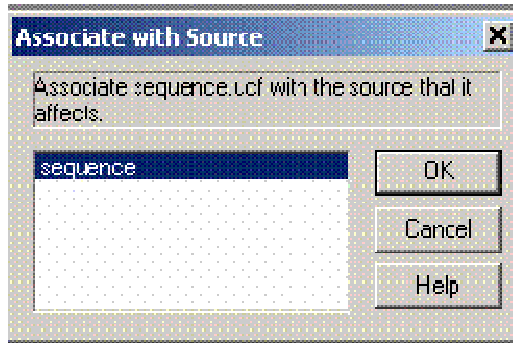


Figure 9: Associate with Source

4. Create programming file; Highlight **sequence.vhd**, expand **Implement Design** and double click **Generate Programming File** in the **Processes for Source** pane.
5. Reading implementation results; Expand **Fit** under **Implement Design** and double click on **Fitter Report**. Answer question in the Laboratory section based on the information in this report.

**Note:** By double click on the sequence.ucf file in the **Sources in Project** window a graphical interface for entering pins information can be seen and modified if necessary.

**Note:** A graphical view of the chip and its pin assignment can be obtained by: highlight sequence.vhd in **Sources in Project** window; expand **Implement Design** under **Processes in Project** window; expand **Fit** under **Implement Design** process and double click on **View Fitted Design (Chip Viewer)**.

**Note:** At this point make sure you have the power to the CPLD board switched on (turn the power strip and make sure black slide switch next to the parallel cable connector is set to EXT and the A/B switchbox on the top of the computer is set to CPLD).

6. Launching iMPACT; Expand **Generate Programming File** in the **Processes for Current Source** pane and double click on **Configure Device (iMPACT)**. Choose **Boundary-Scan Mode** and click **Next**. Choose **Automatically connect to cable and identify Boundary-Scan chain** and click **Finish**.

Note: iMPACT stands for Intelligent Multi-purpose Programming And Configuration Tool. Note that you can only have one instance of iMPACT opened. If you have more than one you will an error indicating problems in the communication with the chip.

7. Assigning a configuration file: A dialog box will pop up letting you know that computer has successfully connected to the board, click **Ok**. Browse to the file **sequence.jed** in your project directory and click **Open**.
8. Programming the CPLD; Click on the Xilinx chip to highlight it and go to **Operations, Program**. Choose **Erase Before Programming** and **Verify**. Click **Ok**. Refer to Figure 10.

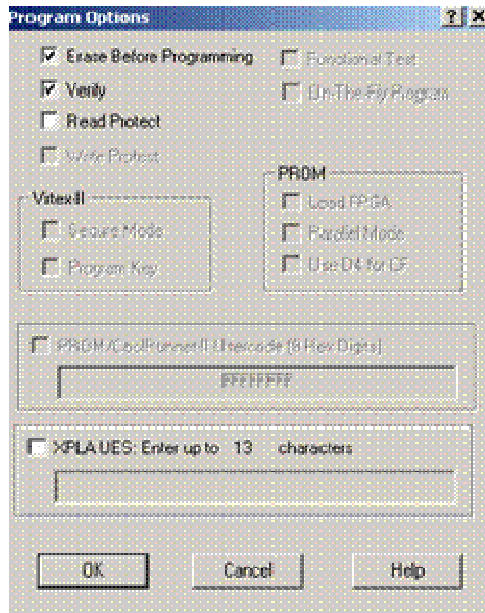


Figure 10: Program Options

**Note:** By checking the option Read Protect on Program Options window, we disable the CPLD design from being read back. This is particularly useful for intellectual property protection. At the bottom of Program Option window there is a check for XPLAUES. UES stands for user electronic signature. A message (i.e. name and date) can be recorded on the chip for future reference.

9. Once the chip has been programmed you will receive a confirmation. Close iMPACT and do not save. Refer to Figure 11.

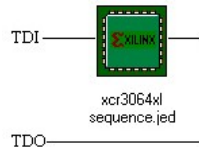


Figure 11: Programming Succeeded

---

## Test the Project

## Step 5

---

1. Reset the board; Set the switch designed for the reset signal to high (push it away from you).
2. Test reset state; With the reset switch set up to high, change input X (by manipulating switch assigned to signal X) and generate several clock cycles (by pushing the button assigned to the clock signal). Another way to test the reset signal is to input half the

sequence X into the system and then activate the reset signal. See what happens.

3. Looking for a sequence; Set the reset signal to low. Give a value to the input signal X by manipulating the corresponding switch and generate a clock cycle (by pushing the corresponding button) to make the system read the input X.
4. Demonstrating your system to your T.A.; Introduce a wrong sequence to the system and show the corresponding state changes according to your state diagram. Introduce a right sequence and show the corresponding state changes according to your state diagram. Ask your T.A. to sign your final report.
5. Testing CPLD memory capacity; Switch the power off and on to the power strip. Notice what happens.

---

## Final Task

## Step 6

---

1. Design a system: Design a system able to recognize the sequence 00101.
2. Implementing your design: Follow steps 1 to 5 and implement your design on the XCR board. Demonstrate your system to your T.A and ask him/her to sign your final report.

---

## Bonus Activity

## Step 7

---

The VHDL code given for the sequence detector (sequence.vhd) contains several processes. From what we have seen in classes we know that all of those processes are executed in parallel and within the processes, each instruction is executed in a sequential way. The first of the process describes the reset signal behavior. It is an asynchronous reset.

1. Change your design of Step 6 so that reset becomes synchronous (for the system to acknowledge the reset signal, a rising clock edge must occur)

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.