EELE 367 – Logic Design
Lab #1 – Introduction

Overview

This lab will introduce you to the Altera Quartus II *Integrated Design Environment* (IDE), which we will use to synthesize, and implement our VHDL designs. This lab will also introduce you to the *Terasic DE0-nano* FPGA board, which contains an Altera *Cyclone IV* FPGA that we use to test our digital designs. Finally, this lab will introduce you to the DE0-nano *I/O Shield*, which plugs into the DE0-nano FPGA board to provide additional basic I/O (4x 7-segment displays, 16 LEDs, and 16 input switches). This lab provides a tutorial that walks you through creating a new Quartus II project, implementing a design to drive all of the basic I/O, downloading the programming file to the DE0-nano, and running your design.

Outcomes

After completing this lab you should be able to:

- Create a new Quartus II project and synthesize a VHDL design for implementation on the DE0-nano FPGA Board.
- Make pin assignments which map the pins on the Cyclone IV FPGA to the signal names defined in your VHDL entity.
- Download the FPGA bitstream to the DE0-nano and run your design.

Deliverables

The deliverable(s) for this lab are as follows:

- Demonstration of the successful operation of your design (90% of your grade).
- Uploading your top.vhd files for this lab to the course DropBox (10% of your grade)

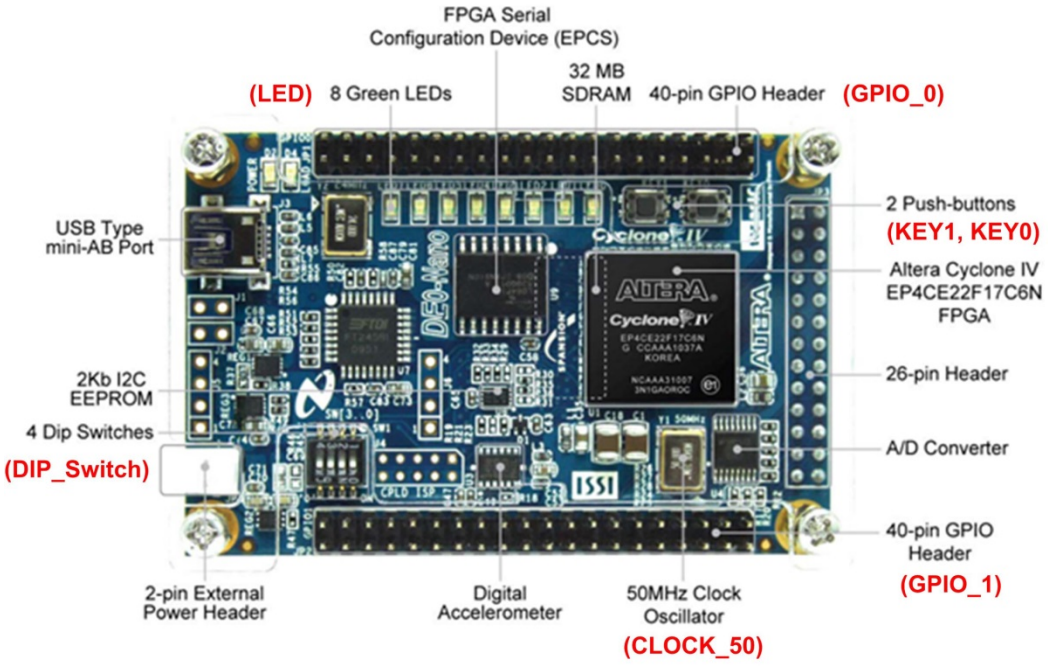
Pre-Lab

n/a

Lab Work & Demonstration

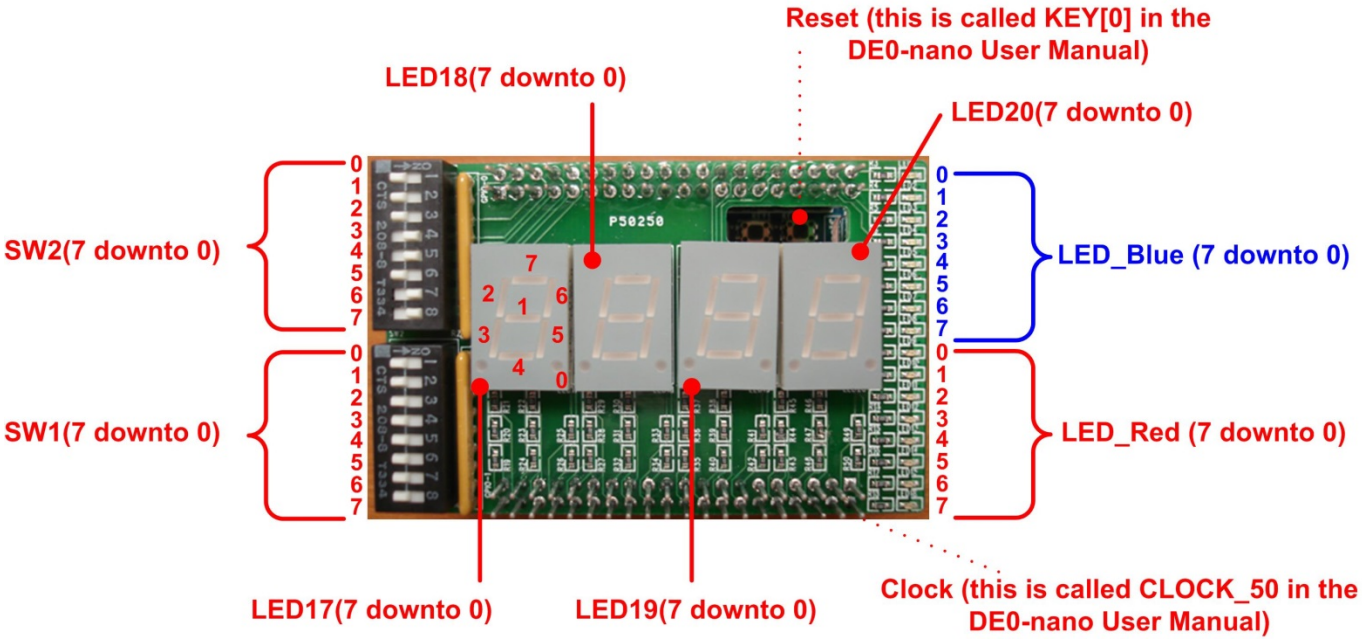
Part 1 – Hardware Overview (no demo, just reading)

The *DE0-nano board* from Terasic contains the Altera Cyclone FPGA that we will be implementing our VHDL designs on. On this board are a variety of basic I/O including LEDs, switches, and buttons. This board also contains a 50MHz oscillator that we will use for our clock. We will power the DE0-nano board through the USB-mini connector, which also enables the programming of the Cyclone VI. This board also contains 2x, 40-pin GPIO pin headers which allow additional I/O to be connected to the board. In our lab, we will use these GPIO pins to interface with the I/O shield to provide even more basic I/O. The FPGA pins that are connected to each of the I/O on the DE0-nano board are given in the *DE0-Nano User Manual*. The following is an annotated picture of the DE0-nano board.



***The RED annotations are the names given to the I/O in the DE0-nano User Manual**

The *DE0-nano I/O Shield* is a custom board that has been designed to plug into the DE0-nano in order to provide even more basic I/O. The following is an annotated picture of the I/O Shield. The names shown here will be used in our top level entity definition in our VHDL design.



The I/O shield can be plugged onto the top of the DE0-nano board in order to make connections with the GPIO header pins. In this configuration, we can define a top level entity for our system as follows:

```
entity top is
    port (Clock    : in  bit;
          Reset    : in  bit;
          SW1      : in  bit_vector (7 downto 0);
          SW2      : in  bit_vector (7 downto 0);
          LED_Red  : out bit_vector (7 downto 0);
          LED_Blue : out bit_vector (7 downto 0);
          LED17    : out bit_vector (7 downto 0);  --Seven Segment LEDs + Decimal
          LED18    : out bit_vector (7 downto 0);  --Seven Segment LEDs + Decimal
          LED19    : out bit_vector (7 downto 0);  --Seven Segment LEDs + Decimal
          LED20    : out bit_vector (7 downto 0)); --Seven Segment LEDs + Decimal
end entity;
```

Based on the pinouts from the DE0-nano User Manual and the schematics from the I/O shield, this entity definition has a pin mapping as follows:

Entity Name	FPGA Pin	DE0 User Manual Name
Clock	PIN_R8	# CLOCK_50
Reset	PIN_J15	# KEY0
SW1[0]	PIN_A5	# GPIO_08
SW1[1]	PIN_D5	# GPIO_09
SW1[2]	PIN_B6	# GPIO_010
SW1[3]	PIN_A6	# GPIO_011
SW1[4]	PIN_B7	# GPIO_012
SW1[5]	PIN_D6	# GPIO_013
SW1[6]	PIN_A7	# GPIO_014
SW1[7]	PIN_C6	# GPIO_015
SW2[0]	PIN_D3	# GPIO_00
SW2[1]	PIN_C3	# GPIO_01
SW2[2]	PIN_A2	# GPIO_02
SW2[3]	PIN_A3	# GPIO_03
SW2[4]	PIN_B3	# GPIO_04
SW2[5]	PIN_B4	# GPIO_05
SW2[6]	PIN_A4	# GPIO_06
SW2[7]	PIN_B5	# GPIO_07
LED_Red[0]	PIN_D9	# GPIO_025
LED_Red[1]	PIN_C9	# GPIO_024
LED_Red[2]	PIN_E9	# GPIO_023
LED_Red[3]	PIN_F9	# GPIO_022
LED_Red[4]	PIN_F8	# GPIO_021
LED_Red[5]	PIN_E8	# GPIO_020
LED_Red[6]	PIN_D8	# GPIO_019
LED_Red[7]	PIN_E7	# GPIO_018
LED_Blue[0]	PIN_B12	# GPIO_033
LED_Blue[1]	PIN_D12	# GPIO_032
LED_Blue[2]	PIN_D11	# GPIO_031
LED_Blue[3]	PIN_A12	# GPIO_030
LED_Blue[4]	PIN_B11	# GPIO_029
LED_Blue[5]	PIN_C11	# GPIO_028
LED_Blue[6]	PIN_E10	# GPIO_027
LED_Blue[7]	PIN_E11	# GPIO_026
LED20[0]	PIN_T13	# GPIO_13

```
LED20[1]        PIN_T15         # GPIO_11
LED20[2]        PIN_R12         # GPIO_16
LED20[3]        PIN_T12         # GPIO_15
LED20[4]        PIN_T14         # GPIO_12
LED20[5]        PIN_R13         # GPIO_14
LED20[6]        PIN_F13         # GPIO_10
LED20[7]        PIN_T11         # GPIO_17

LED19[0]        PIN_R10         # GPIO_111
LED19[1]        PIN_R11         # GPIO_19
LED19[2]        PIN_N9          # GPIO_114
LED19[3]        PIN_P9          # GPIO_113
LED19[4]        PIN_P11         # GPIO_110
LED19[5]        PIN_N12         # GPIO_112
LED19[6]        PIN_T10         # GPIO_18
LED19[7]        PIN_N11         # GPIO_115

LED18[0]        PIN_P16         # GPIO_121
LED18[1]        PIN_L15         # GPIO_119
LED18[2]        PIN_N15         # GPIO_124
LED18[3]        PIN_N16         # GPIO_123
LED18[4]        PIN_P15         # GPIO_120
LED18[5]        PIN_R14         # GPIO_122
LED18[6]        PIN_R16         # GPIO_118
LED18[7]        PIN_P14         # GPIO_125

LED17[0]        PIN_L13         # GPIO_129
LED17[1]        PIN_N14         # GPIO_127
LED17[2]        PIN_J13         # GPIO_132
LED17[3]        PIN_K15         # GPIO_131
LED17[4]        PIN_M10         # GPIO_128
LED17[5]        PIN_J16         # GPIO_130
LED17[6]        PIN_L14         # GPIO_126
LED17[7]        PIN_J14         # GPIO_133
```

Part 2 – Creating a Simple Quartus II Project (Demo #1)

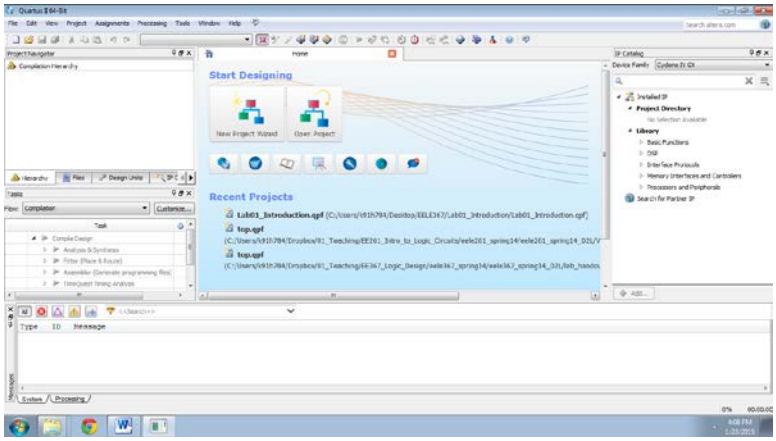
We are now going to create a new Quartus II project that will implement a simple VHDL design that will drive the switch inputs onto the LEDs of the I/O shield.

A. Create a Folder for Today's Lab

Decide on where you are going to keep all of your lab files. In this location, create a folder called "Lab01_Introduction". This must be created prior to making a new project in Quartus II.

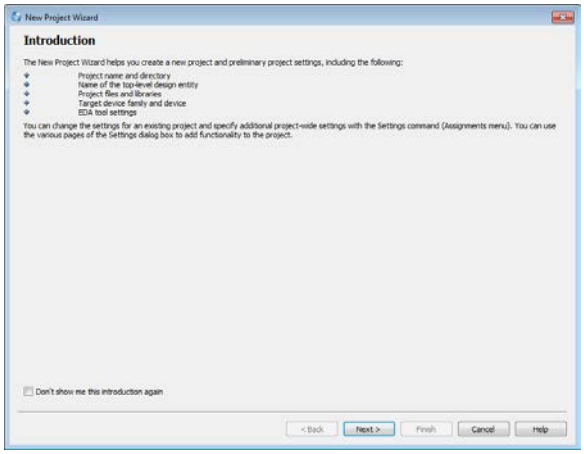
B. Launch Quartus II

Start – All Programs – Altera 14.xx Web Edition - Quartus II Web Edition 14.xx – Quartus II 1.4.0 (64-bit). You should see the following screen.



C. Create a New Project using the “Project Wizard”

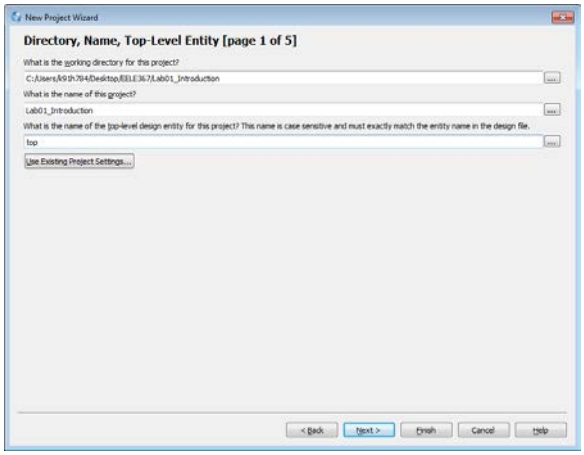
File – New Project Wizard.



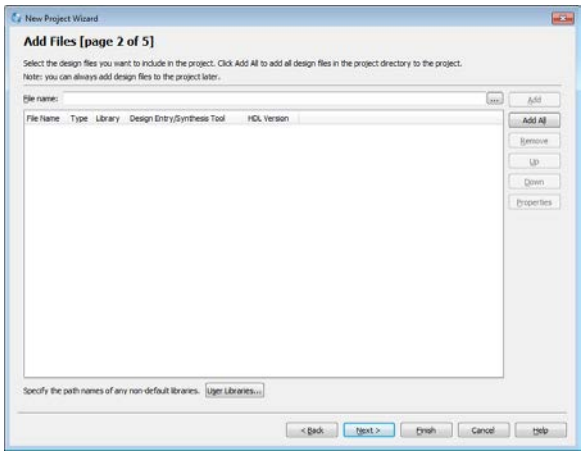
On the Introduction page, click “Next”.

In the next page fill out information about the project name and location:

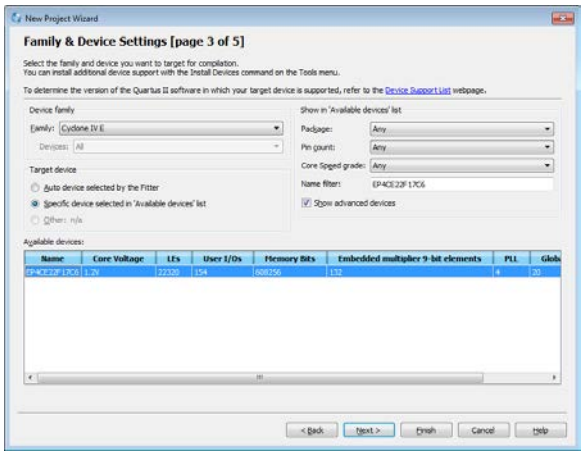
Working directory	= browse to the project directory you created before
Name of the project	= Lab01_Introduction
Name of top level	= top (Note that this is what you need to name your entity.



On the Directory/Name page, click “Next”.

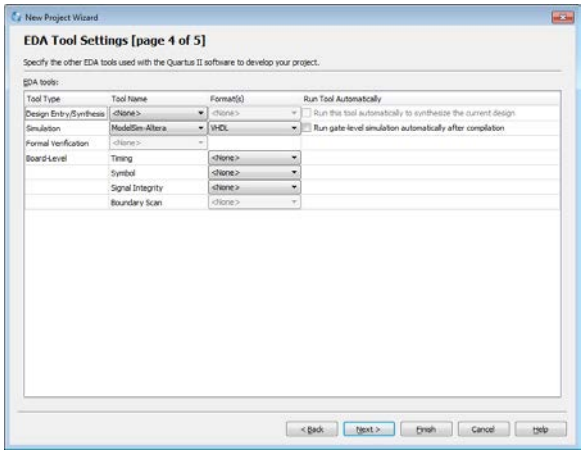


The Add Files screen is where you would add an existing VHDL files if you had them. For this lab we will create our VHDL file directly in Quartus II. Click “Next”.

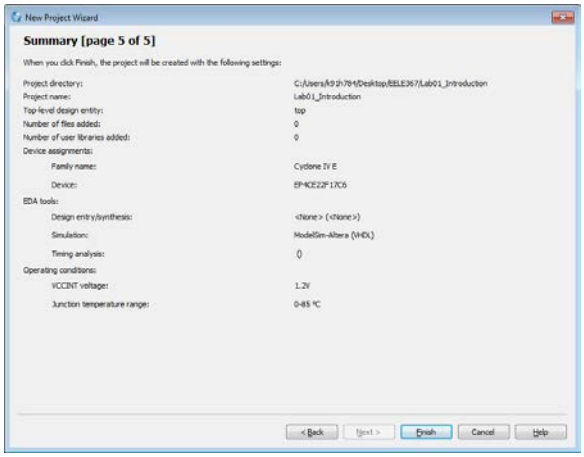


On the Family & Device page, select the EP4CE22F17C6 FPGA as the device that our design will be implemented on. First, select “Cyclone IV E” from the Device family drop down. Then you can filter the devices by starting to type the name into the Name Filter field. Once you find the device, select it.

Click “Next”.



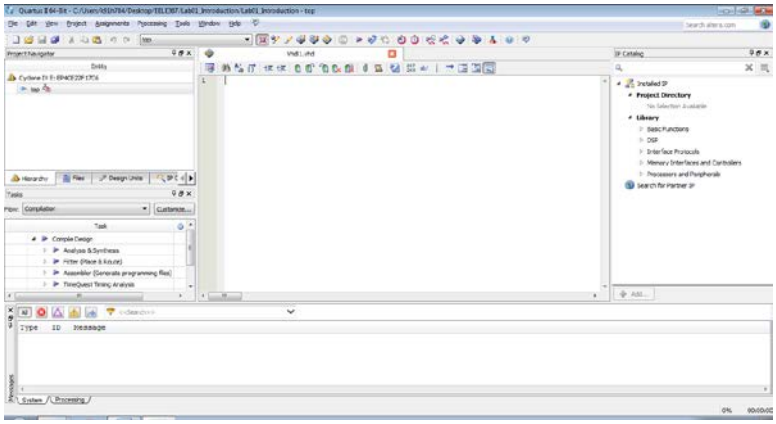
You don't need to configure any settings on the EDA Tool page. Click "Next"



On the Summary page, click "Finish"

D. Create the VHDL Source File for the Design

File – New – VHDL File



At this point the file is not named. We need to save it as **top.vhd**. It is very important to name this file the same name as what you called the top level entity in the project wizard. In general, we will always call our top level entity **top**.

File – Save As, enter top.vhd.

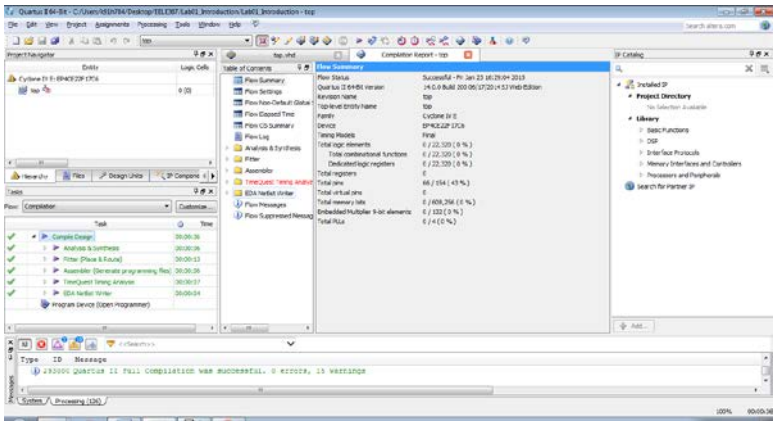
Now let's design a system that simply drives the values on the switches (SW1 and SW2) to all of the LEDs on the I/O shield. In your top.vhd file, enter the following VHDL:

```
1  entity top is
2    port (Clock    : in  bit;
3          Reset    : in  bit;
4          SW1      : in  bit_vector (7 downto 0);
5          SW2      : in  bit_vector (7 downto 0);
6          LED_Red  : out bit_vector (7 downto 0);
7          LED_Blue : out bit_vector (7 downto 0);
8          LED17    : out bit_vector (7 downto 0);   --Seven Segment LEDs + Decimal
9          LED18    : out bit_vector (7 downto 0);   --Seven Segment LEDs + Decimal
10         LED19    : out bit_vector (7 downto 0);   --Seven Segment LEDs + Decimal
11         LED20    : out bit_vector (7 downto 0));  --Seven Segment LEDs + Decimal
12 end entity;
13
14 architecture top_arch of top is
15
16  begin
17
18    LED_Blue <= SW2;
19    LED_Red  <= SW1;
20
21    LED17 <= SW1;
22    LED18 <= SW1;
23    LED19 <= SW2;
24    LED20 <= SW2;
25
26 end architecture;
27
```

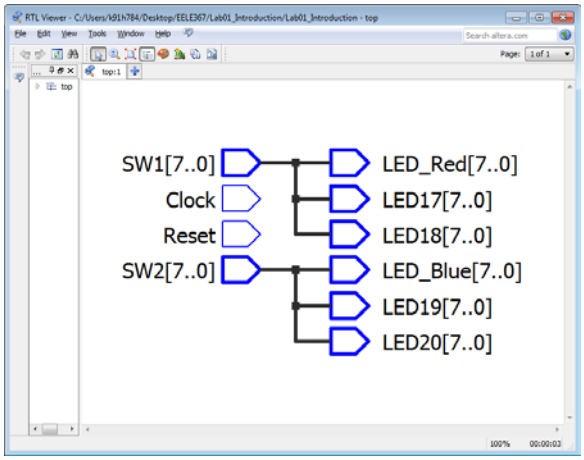
Save your top.vhd file using a File – Save.

E. Compile your Design

In the Task field of Quartus II (lower left), you will see all of the commands that can be executed on your project. You run a command by double clicking on the command. If you select a higher level command such as Compile, it will execute all of its lower level tasks (i.e., Analysis & Synthesis, Fitter, etc.). If the compile is successful, it will display a green check next to every step. If it is unsuccessful, it will display a red X next to the step that failed. Double click on “Compile”. If you get errors, fix them and recompile until it passes.



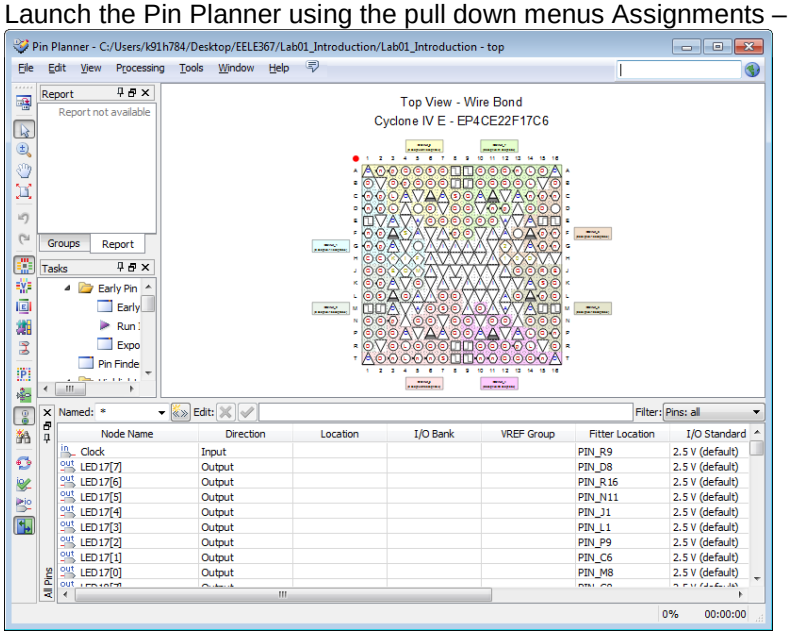
At this point, your design has compiled successfully. More has happened during this step than a simple syntax check. Quartus has actually synthesized your design and mapped it into the hardware on the FPGA. You can see the schematic of what was synthesized by executing “Analysis & Synthesis – Netlist Viewer – RTL Viewer”. For this design we have created simple wires, but this view becomes useful for more complex designs.



While your design has been mapped into the gates of the FPGA, we have not provided any information about how we want our entity ports to be routed to the pins of the FPGA. We need to make these assignments and then recompile in order to complete the design.

F. Assign the Pins for the Design

Launch the Pin Planner using the pull down menus Assignments – Pin Planner.

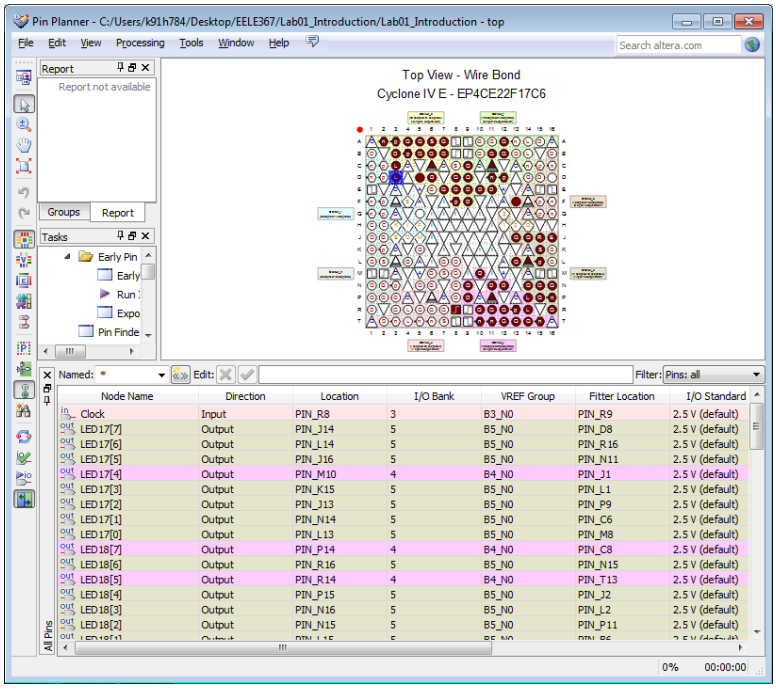


You'll see a graphic of the FPGA package and a list of signal names that were read in from the entity. For each signal, you can assign its pin in the "Location" field. To do this you first double click in the field, then use the drop-down menu to select the desired pin.

Enter the pin assignments given in part 1. This will take some time. If you want to avoid this manual step, you can create a csv file that can be imported into Quartus using the pull down menu Assignments – Import Assignments. The format of the csv file is as follows:

	A	B
1	To	Location
2	Clock	PIN_R8
3	Reset	PIN_J15
4	SW1[0]	PIN_A5
5	SW1[1]	PIN_D5
6	SW1[2]	PIN_B6
7	SW1[3]	PIN_A6
8	SW1[4]	PIN_B7
9	SW1[5]	PIN_D6
10	SW1[6]	PIN_A7
11	SW1[7]	PIN_C6
12	SW2[0]	PIN_D3

After you have entered (or imported) the pin assignments, you should see the following in Pin Planner.

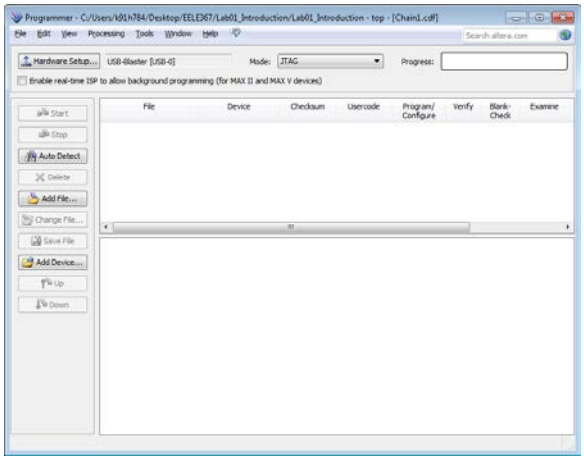


Once all of the signals have a pin assigned, save the assignments and **recompile your entire design**.

G. Program the FPGA

Now you are going to download your design file (aka a bitstream) to the FPGA. The file that will be downloaded in located in the “output_files” folder in your project directory and is called “top.sof”.

Launch the programmer by double clicking on “Program Device” in the task window of Quartus.



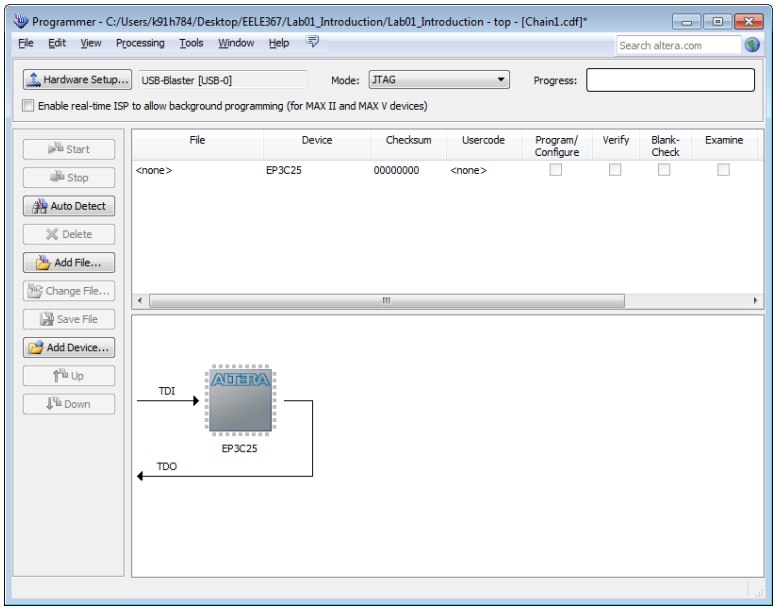
The first step in programming is to detect the device. At this point, you should plug in the DE0-nano board to your computer using a USB connector.

Note: The USB-Blaster drivers need to have already been installed. If you haven’t installed the drivers, the steps are as follows:

- Plug in the USB cable between the DE0-nano and your computer. The USB driver will attempt to install but will fail. This step will allow the DE0-nano to show up as a device called “Altera USB Blaster”, but does not have the proper driver.
- Select the USB Blaster device from Device Manager and then manually update its driver. The driver is found in the Altera install directory at: C:\altera\14.0\quartus\drivers\usb-blaster. Note that when you browse to this path, DO NOT select 32-bit or 64-bit, the install will do that for you. Also DO NOT select usb-blasterII.

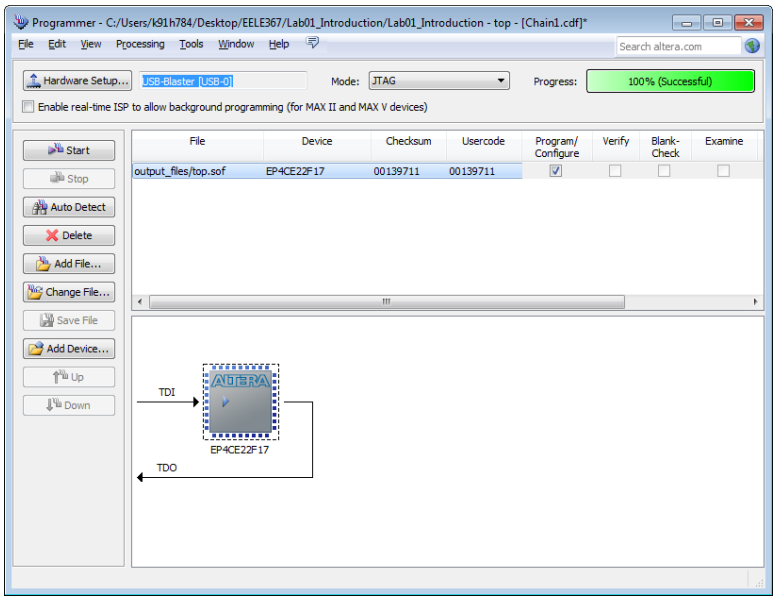
In the programmer window, click “Auto Detect”. The programmer will go out and find the DE0-nano board and ask which device you wish to program. Select the EP4CS22 device.

Note: If the auto detect fails, you have not selected USB-Blaster as the hardware to use to talk to the DE0-nano board. Click on the Hardware Setup button and choose “USB-Blaster”. Then try Auto-Detect again.



At this point, the programmer has found the FPGA, however, it does not have the proper file to download. Click on the “Add File” button and browse in your project directory to “output_files/top.sof”. This will add a valid file in the upper field of the programmer window. You will need to delete the file line that starts with “<none>”. Select this line and click the “Delete” button.

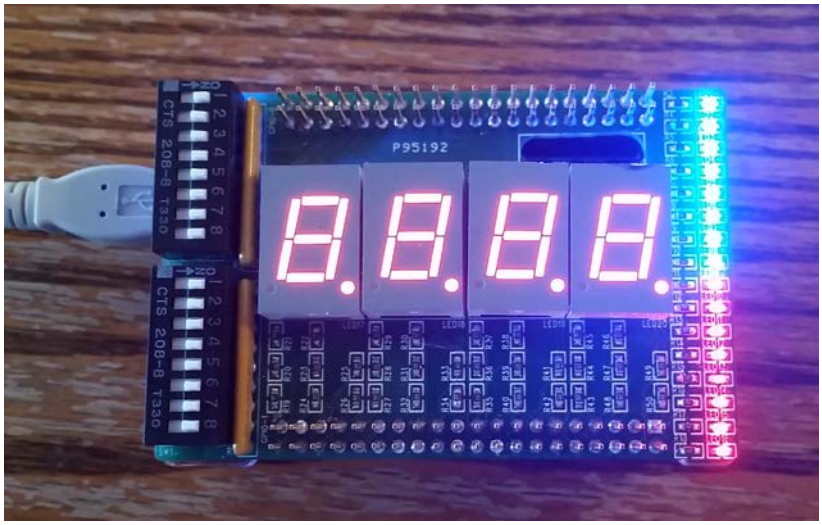
Now highlight the top.sof line and press “Start”. Once the device is programmed, the window will indicate that the programming is successful.



Save your programming configuration so that you don’t have to repeat the above steps the next time you program your FPGA.

H. Test your Design

Your program has now been downloaded and is active. You can toggle the switches on the I/O shield and a variety of LEDs will light up.



DEMO

I. Demonstrate your Design

demonstrate the proper operation of your design. This is worth 90% of your grade.

Also, upload your top.vhd file to the Lab #1 DropBox. This is worth 10% of your grade.