

# Introduction IP Cores

## Introduction

Today's design engineer has powerful tools at his or her fingertips, which can simplify the task of hardware design. Among these tools are IP Cores, pre-packaged blocks of VHDL code that can reduce the time an engineer has to spend on a design. In this part of the tutorial, you will learn how to use Xilinx's CORE Generator System to create a VHDL IP core and incorporate it into a VHDL project.

## Goals

After completing this lab, you will be able to:

- Understand the two's complement number system.
- Use IP Cores to create projects.
- Use structural level of abstraction to describe a digital system.

## Start Project Navigator and Create the Project

Step 1

As you have done previously, create a new directory for your project with the following format: **C:\ece238\Fall2005\student\_name\lab4**. Download the file **complement.vhd** to your new directory.

The steps for creating this project will be a little different than the steps for the previous labs.

1. Open Xilinx ISE 7.1 and create a new project; as you did in previous labs open ISE 7.1 by going to **Start Menu** → **Programs** → **Xilinx ISE 7.1i** → **Project Navigator**. Create a new project named **lab4** and choose the settings shown in Figure 1. These settings will ensure that the device has enough space to store your project.

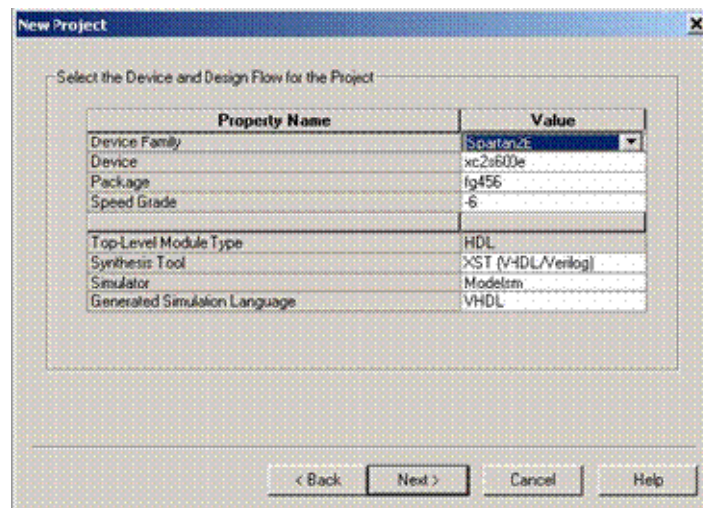


Figure 1: Device Settings

## Create a Core using Coregen

## Step 2

1. Add new source; Go to **Project** → **New Source**. Name your file **twos\_comp**. On the left hand side, choose **IP (CoreGen)**. Click **Next**. Refer to Figure 2.
2. Expand **Math Functions, Complementers**. Click **Twos Complementer**. Click **Next**. Click **Finish**. Refer to figure 3.

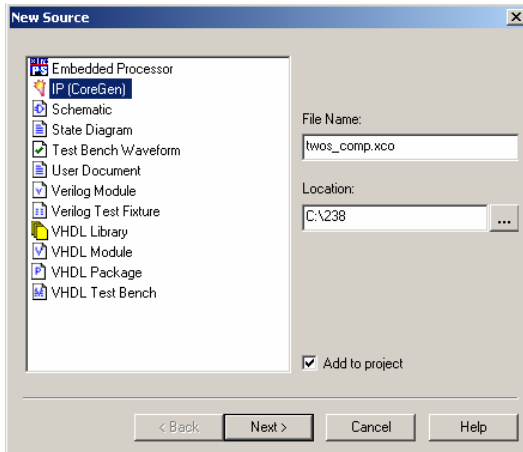


Figure 2: New Source Settings

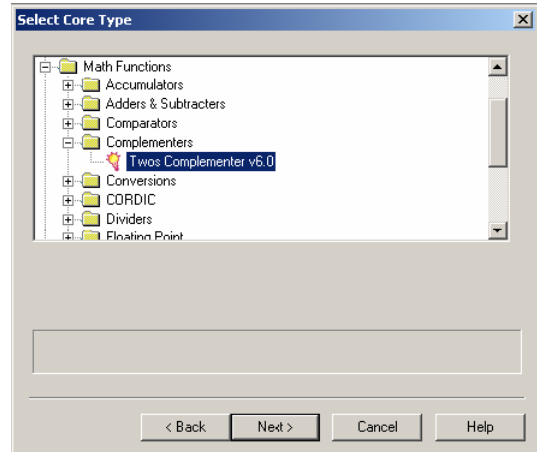


Figure 3: Select Core Type

3. Click **Next, Next, Finish**. The Core Generator will be opened. Use the following input:

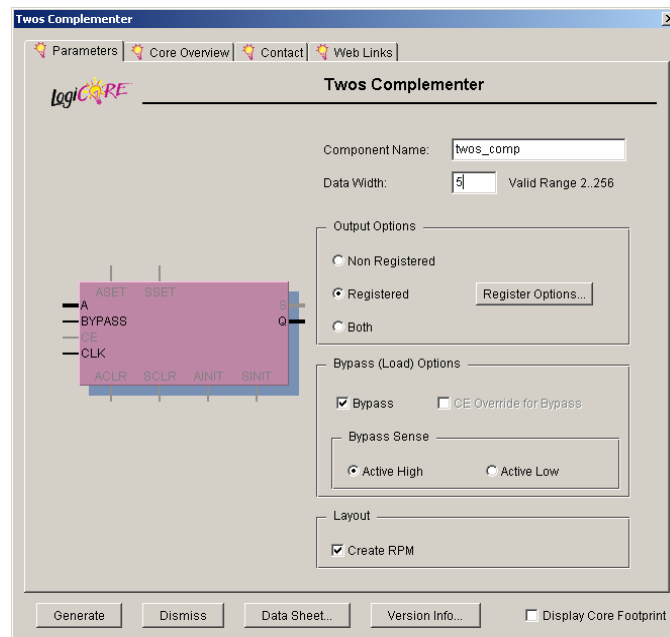


Figure 4: IP Core Generator Settings

We are setting up a 2's completer with a data width of 5 bits and registered (which means that flip-flops will be setup at the inputs and outputs). A **bypass** signal is also setup. **Active high** means that the completer will be bypassed whenever the signal "**bypass**" is setup to high. Later it will become clear why we need this signal. Click **Generate**.

4. After some time, a message in the console will tell you that the core was generated successfully.

## Incorporate the Core

## Step 3

Maximize the **Project Navigator** window. If you look to the left, you will notice that the core now appears in your design, nor is your design ready to be simulated. In order to proceed we will use a "wrapper" (or "top-level") file called **complement.vhd** which was provided (step 1). The only role of this file is to take the core we just generated and place it into a design, connecting its inputs and outputs to the inputs and outputs of the design as shown in figure 5.

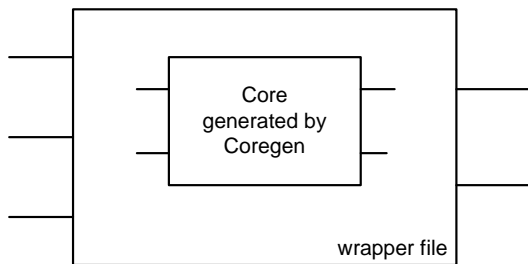


Figure 5. Wrapper file and core

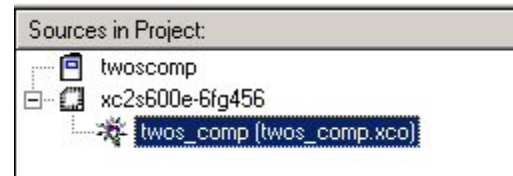


Figure 6: Sources in Project

1. Highlight the generated file in the **Sources in Project** window, but **DO NOT** double-click it. Refer to Figure 6.
2. Click the **Open** button on the task bar. Type in the filename **twos\_comp.vho** and click **Open**. This will open one of the files created by the CORE Generator System. This does not actually do anything for you. However, we will be copying some information out of the file, so it is nice to have it handy.
3. Add the **complement.vhd** file to your project; Go to **Project, Add Source**. Select the **complement.vhd** file, click **Open**. Choose **VHDL Design File** as the **Source Type**, click **Ok**. Now we will have to copy sections of code from the **twos\_comp.vho** file to complete the code.
4. Click on the tab for the **twos\_comp.vho** file you were asked to open. Copy the **component declaration** (shown in Figure 7a) and Paste it into the **complement.vhd** file where the comments indicate.
5. Copy the **port map** (shown in Figure 7b) from the **twos\_comp.vho** file. Paste it into the **complement.vhd** file where the comments indicate.

```

component twos_comp
  port (
    A: IN std_logic_VECTOR(4 downto 0)
    BYPASS: IN std_logic;
    CLK: IN std_logic;
    Q: OUT std_logic_VECTOR(5 downto 0)
  end component;

```

Figure 7a: Component Declaration

```

your_instance_name : comps
  port map (
    A => A,
    BYPASS => BYPASS,
    CLK => CLK,
    Q => Q);

```

Figure 7b: Port Map

6. Highlight the **complement.vhd** file. Double-click **Synthesize** in the **Processes for Source** pane. You will get some warnings (yellow marks) that are part of the process, disregard them. Make sure you don't get any errors (red marks).

## Simulate the system

## Step 4

1. Add the **complement\_tb.vhd** code to your project; Download the file **complement\_tb.vhd** provided in the lab website. Go to **Project** → **Add source**, look for the file **complement\_tb.vhd** and highlight it. Choose **VHDL Testbench File** as the **Source Type**, click **Ok**.
2. Highlight the test bench and double-click on **Simulate Behavioral Model** in the **Processes for Current Source** window.
3. Your waveform should look like Figure 8.

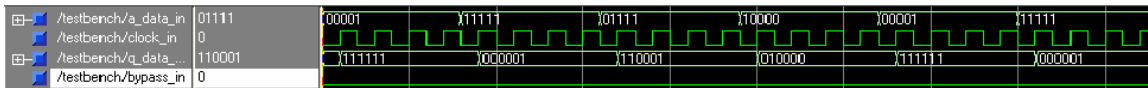


Figure 8: ModelSim Waveform

4. If you right click on the signals for the input and output, you can change the display radix. Go to **Radix**, click on **Decimal**. After changing the radix, it is much easier to see that the twos-complementer is working.

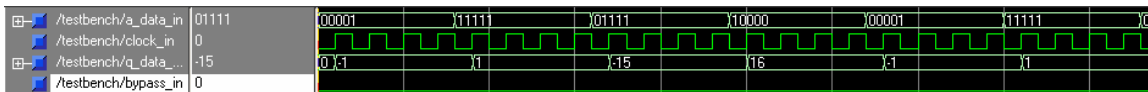


Figure 9; ModelSim Waveform with radix changed

In the past you had individual signals where a high was a single line and conversely, a low flat line was a zero. The wave that you see above is a box with numbers in it. When you see 10000, this means that it is a bus (collection of wires) instead of an individual input or output. You can see the individual wires for each bus if you click the **plus sign** next to each signal.

## Final Task: Design of an Adder/Subtractor

## Step 5

In this section we'll use what we just learn to implement the adder/subtractor shown in figure 10. The adder block and the 2's completer block will be created using Coregen. You will need to create a wrapper file and structural description to connect these elements together and build the system.

From the lecture notes it is known that in order to subtract ( $A - B$ ) two numbers in 2's complement representation the operand ( $B$ ) must be complemented and the result added to the second operand ( $A$ ). Schematic shown in figure 10 has a 2's complement block with a signal named **bypass**, which will enable or disable the 2's complement block according to the **add\_sub** input signal of the system. Let say we want to add  $A$  and  $B$ . **add\_sub** signal will then be high (1) and the 2's complement block will be bypassed. As a result  $A$  and  $B$  get added by the Adder block. Let say that now we want to subtract  $A$  and  $B$  ( $A - B$ ). **add\_sub** signal will now be low (0), the 2's complement block will take  $B$  and calculate its complement to send it to the Adder block. The adder block will add  $A$  and the 2's complement's of  $B$  which is equivalent to subtract  $B$  from  $A$  ( $A - B$ ).

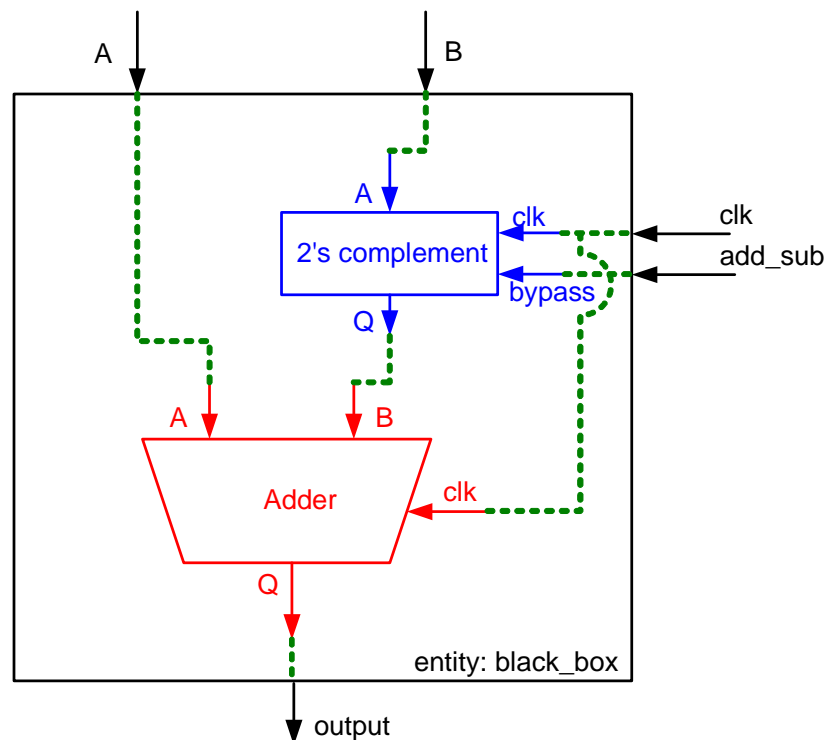


Figure 10. Adder – Subtractor schematic.

1. Create a new project; As you did before, create a new project, this time called **lab4\_2**.
2. Add source file; Add the file **top.vhd** provided in the website by downloading it to your working directory and going to **Project** → **Add source** in the ISE main interface window. Look for the file **top.vhd**, highlight it and click **Open**. Choose **VHDL Design File** as the **Source Type**, click **Ok**.

3. Create and incorporate 2's complement core; Duplicate steps 2 and 3 of this lab to create and incorporate the 2's complement core (use the same parameters as before). Note that in this case you need to copy the component declaration and component instantiation in the **top.vhd** file that you added previously.
4. Create an Adder core; Duplicate step 2 of this lab to create the core for an adder. Use the **Adder Subtractor** core that is found under **Math Functions / Adders & Subtractors** in the Coregen first window. The core needs the parameters shown in figure 11. Once you finished setting up these parameters, click on the **Next** button and fill out the parameters shown in figure 12. The important thing about the parameters that you just set up is that the inputs and output ports of the core will be 5 bits wide.

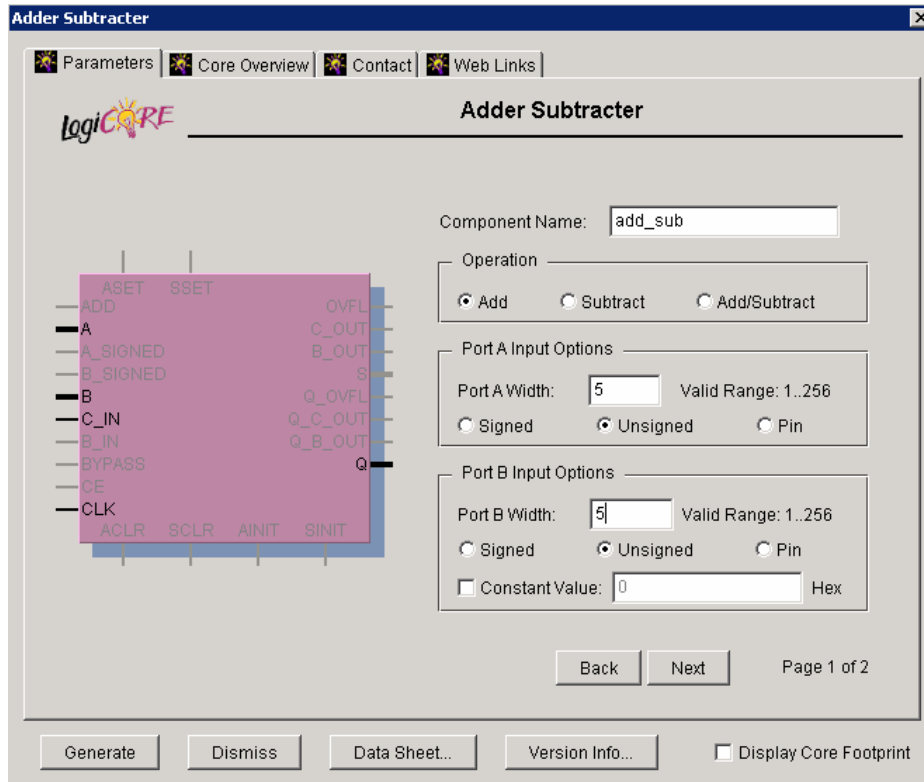


Figure 11; First window parameters for Adder Subtractor Core.

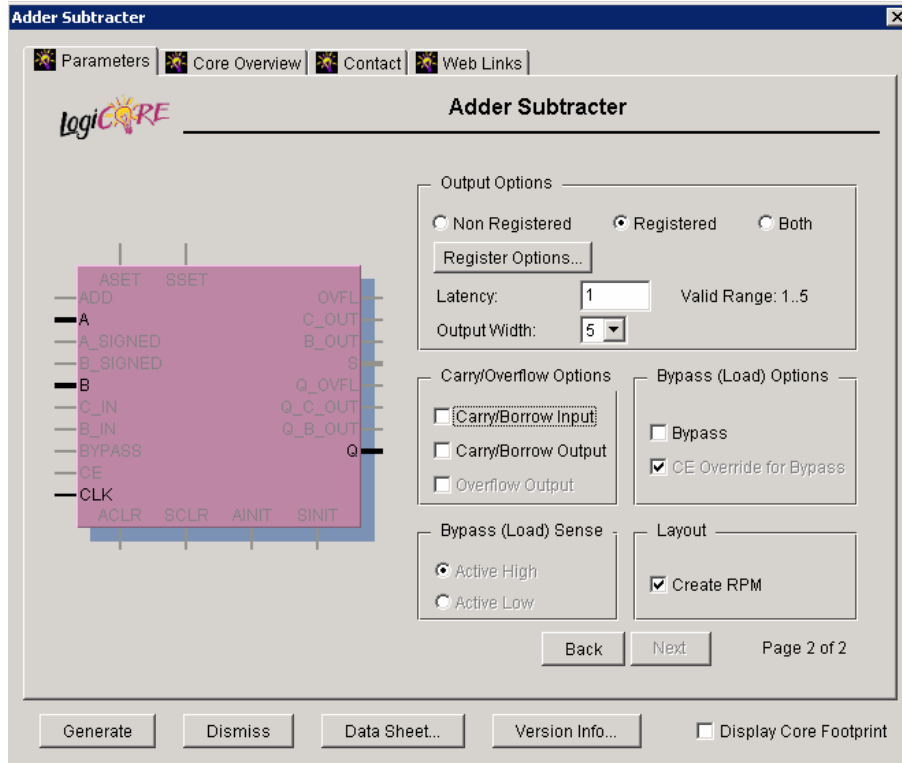


Figure 12; Second window parameters for Adder Subtractor Core.

5. Incorporate the adder core to your design; Similar to step 3 of this lab, copy the component declaration and the component instantiation templates into the **top.vhd** file.
6. Complete the system description, using structural description. Complete the description of the system filling out the connections in the component instantiation section of the code, following the schematic shown in figure 10. The connections you have to complete are drawn in green. Note that the output “Q” of the 2’s complement block is 6 bits wide and the input B of the adder-subtractor is only 5 bits wide. Here you will have to leave the most significant bit of “Q” unconnected. (Hint: instead of connecting “Q”, connect only “Q(4 down to 0)”. This will leave “Q(5)” unconnected).
7. Simulate the system; Similar to step 4 of this lab, download the file **top\_tb.vhd** and add it to your design. In this case you must complete the testbench file. Follow the comments and hints on the file itself. You should at least have the following cases in your simulation:
  - Add two small positive numbers together (say 00110 and 00001).
  - Add a positive number (00010) and a negative number (11110).
  - Subtract two small positive numbers together (say 00110 and 00001).
  - Subtract a positive number (00010) and a negative number (11110).