**Lab 10**

**How to Use Xilinx ISE 12 Project Navigator with IP Cores**

**Acknowledgements:**

Developed by Craig Kief, Engineering Faculty at University of New Mexico, Albuquerque, New Mexico. *Funded by the ATE Program at the National Science Foundation (NSF). This effort was assisted by Qunlan Cao and Andrew Dodd.*

Lab Summary:

This lab will present design entry, simulation, and prototyping with tools that are provided by Xilinx® ISE 12 for this purpose. The use of cores (or prepackaged code) for creating portions of (or full) projects is important. This lab will demonstrate how to make a simple project using Coregen, how to simulate it and how to download it to hardware.

**Lab Goal:**

The goal of this lab is to learn the use of using Xilinx® ISE 12 software with Coregen by implementing inputs and outputs for a simple subtractor.

**Learning Objectives**

1. Create a 2 inputs subtractor project in Xilinx® ISE 12 using the free software **ISE** WebPACK.
2. Use the Xilinx® ISE 12 VHDL Editor to enter a VHDL code project in Xilinx® ISE 12
3. Compile and simulate the design.
4. Download the project to hardware.

**Method of Evaluation:** Your grade will be determined by your instructor.

**Time Required**: 1 hour

**Lab Preparation**

* Read this document completely before you start on this experiment.
* Print out the laboratory experiment procedure that follows.

**Equipment and Materials**

Access to Xilinx software and the Digilent NEXYS-2 prototyping board.

|  |  |
| --- | --- |
| **Software needed** | **Quantity** |
| The following items from the Xilinx: www.xilinx.com   * free software **ISE** WebPACK | 1 |

Additional References:

* Full documentation for the NEXYS-2 board as well as examples can be found at www.digilentinc.com

**Introduction**

**Lab Procedure**

In the world of digital design, engineers use Hardware Description Languages to describe complex logic functions. These are included in design suites such as Xilinx's ISE [1] and similar tools. However, if a digital engineer were to code an adder or create a cosine lookup table each time they were doing a project, it would be reinventing the wheel and a waste of their time. Similarly, if the design engineer had to continually re-code commonly used complex digital circuits in large projects; they would end up wasting more time and money. Because of this, a digital design engineer may just use an IP core [1]. An IP (*Intellectual Property*) core is a block of HDL code that other engineers have already written to perform a specific function. It is a specific piece of code designed to do a specific job. IP cores can be used in a complex design where an engineer wants to save time. As with any engineering tool, IP cores have their advantages and disadvantages. Although they may simplify a given design, the engineer has to design the interfaces to send and receive data from this “black box”. Also, while an IP core may reduce design time, the engineer frequently has to pay for the right to use the core. Many are designed for particular parts but some come free such as on [www.opencores.org](http://www.opencores.org/). Open cores contain many open source projects where the designer has full access to codes and design documents that are built to specific standards. Other cores may cost you thousands of dollars. In the lab, you will be given the chance to incorporate a Xilinx IP core into a simple project.

**Objective**

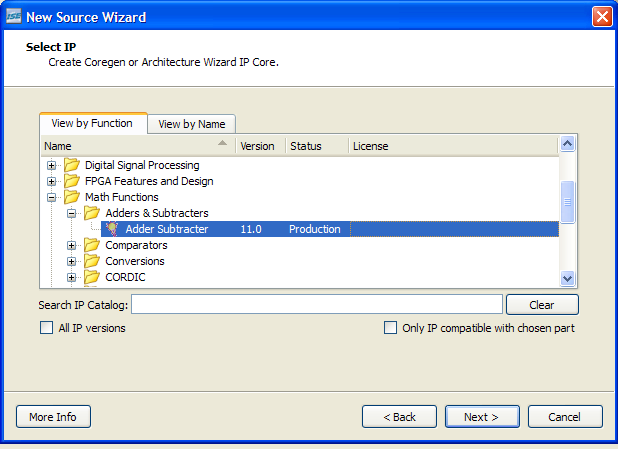
In this tutorial, the designer will learn how to use Xilinx's CORE Generator System to incorporate an IP core into a VHDL project thus creating a subtracter. Xilinx cores are often beneficial to use as they are written by engineers with knowledge of the inner components of the FPGA. This allows them to be optimized for speed and space.

**Process**

1. Use the Xilinx CORE Generator System to create an IP core.
2. Connect the IP Core to the VHDL source as a component.
3. Synthesize, simulate, and program the Spartan 3E NEXSYS-2 board.

**Implementation**

1. Start by creating a project. Launch the Xilinx ISE software. Once the Project Navigator window opens, create a new project by clicking on the File drop-down menu, and selecting ***New Project***. Remember the file location of your project. Change the device properties to match those in the previous labs.
2. Finish creating your project, noting the folder it was created in, and proceed to right clickin the Sources window and select ***New Source***. Choose **IP (Coregen & Architecture Wizard)** in the left menu, name the file ‘**subtracter**’and click **Next**.In the next screen open the tree to Math Functions => Adders & Subtracters => Adder Subtracter 11.0 as shown in Figure 1. Click **Next** and **Finish.**



3. After a few seconds a new window (*LogiCore*) should pop up. Select the options to match those in Figure 2. Take a minute to click on the *Datasheet* button on the bottom of the screen. The provided data sheets are essential in larger designs to understand many timing issues associated with a specific core as well as explaining all the different design choices. Select **Generate** in Figure 2. The window will disappear and will have created a subtracter.

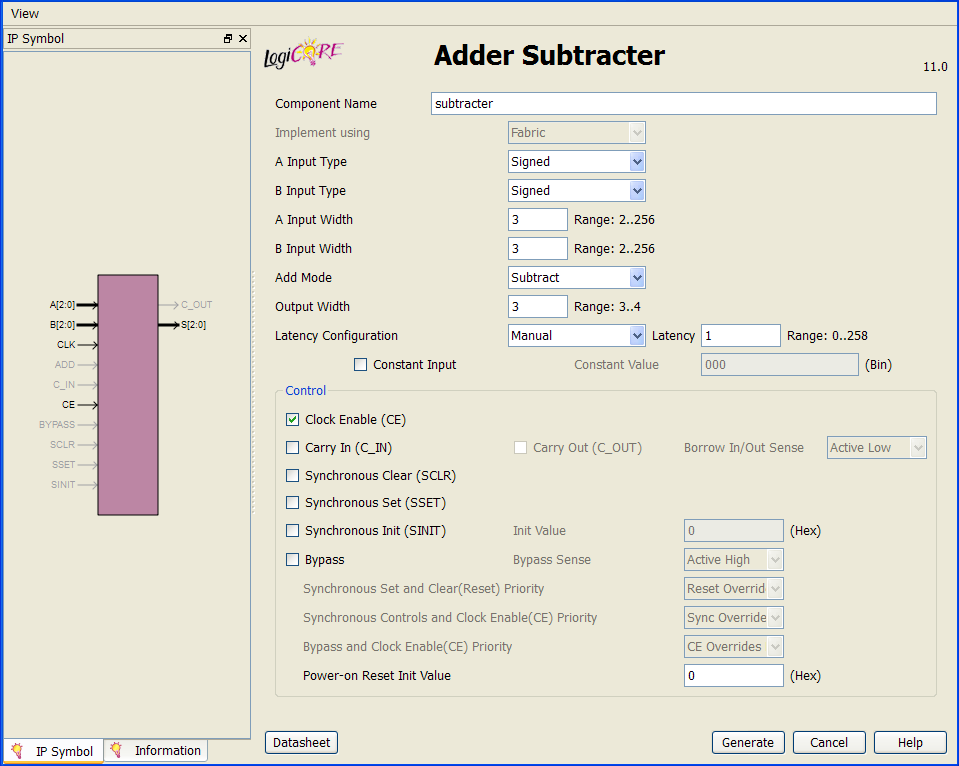


Fig. 2 - First page of the LogiCore Window

1. It will take a few moments to generate all of the files. When it is finished, ‘**subtracter**’will be shownin the sources window inside the Project Navigator. At this point, the easiest part in the tutorial has been completed. The hardest part, integrating the core into the project and simulating is yet to come.
2. Now, right click on the sources window and click **New Source**. Select **VHDL Module** and name the file ‘**main**’. Click **Next**.
3. Now fill in the New Source Wizardas shown in figure 3. Click **Next** and **Finish**.

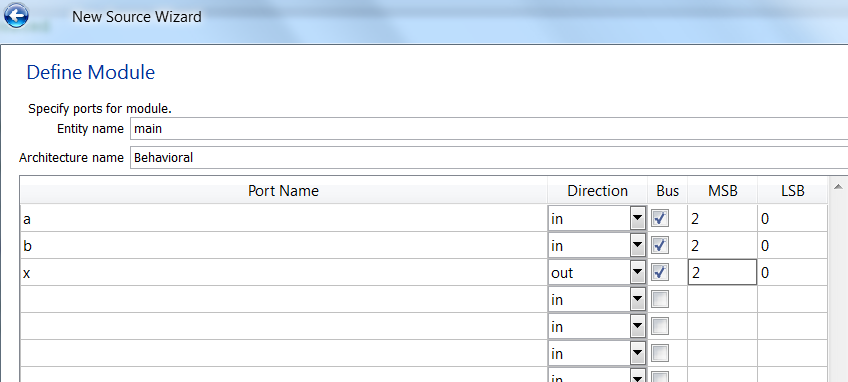


Fig. 3 - Define Module window

7. Go to **File*,*** click on**Open**,and browse to **subtracter.vho**(not .vhd). The .vho file is located within **ipcore\_dir** in your project folder. The .vho file is one of the files created by the CORE Generator System. It does not actually do anything. However, there is some information that needs to be copied from the file, so it is nice to have it handy. If the .vho file is not there then there is a problem in your project, design properties selection. If you go into these settings then you will most likely find that your preferred language is Verilog and not VHDL. Change it to VHDL and regenerate your core. What had happened is that instead of creating the required source files for VHDL (ie .vho), you created them for Verilog.

8. Locate the following text in the .vho file. Copy and paste it into the .vhd file, beneath the text ‘**architecture Behavioral of main is**’.

component subtracter

port (

a: IN std\_logic\_VECTOR(2 downto 0);

b: IN std\_logic\_VECTOR(2 downto 0);

clk: IN std\_logic;

ce: IN std\_logic;

s: OUT std\_logic\_VECTOR(2 downto 0));

end component;

9. Go back to the .vho file and locate the following text. Copy and paste it into the .vhd file, below the **begin** statement.

your\_instance\_name : subtracter

port map (

a => a,

b => b,

clk => clk,

ce => ce,

s => s);

Change your bottom line to this: s => x to allow the mapping to be completed.

10. Change **your\_instance\_name** to something shorter, such as ‘**main**’.

11. Locate the text ‘**entity main** is’. Add the following lines to the entity, after port b and before port x. It would have been easier to just put these in when you were using the wizard to create source file but this is done now to show you that you can add I/O later in the process.

clk : in STD\_LOGIC ;

ce: in STD\_LOGIC ;

**Clk** refers to ‘clock’. It produces a clock signal that transitions back and forth between low and high. Clkalso defines the amount of time that occurs between these transitions. When the clock signal goes from low to high, a ‘rising edge’ is triggered. When the clock signal goes from high to low, a ‘falling edge’ is triggered. Rising and falling edges will be illustrated later in the tutorial, when the behavioral model is simulated through the test bench file.

**Ce** stands for ‘chip enable’. When chip enable is high, the device will function correctly. When chip enable is low, the chip is essentially turned off, and the device will not function correctly.

12. Finally, beneath the text ‘**port map**’, change the second ‘s’ (the one between the arrow and the parenthesis) to an ‘x’. This is because you defined your output as ‘x’ in the Define Module window.

When you are finished, the entire main.vhd file should look like this.

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date:

-- Design Name:

-- Module Name: main - Behavioral

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx primitives in this code.

-- library UNISIM;

-- use UNISIM.VComponents.all;

entity main is

Port ( a : in STD\_LOGIC\_VECTOR (2 downto 0);

b : in STD\_LOGIC\_VECTOR (2 downto 0);

clk : in STD\_LOGIC ;

ce: in STD\_LOGIC ;

x : out STD\_LOGIC\_VECTOR (2 downto 0));

end main;

architecture Behavioral of main is

component subtracter

port (

a: IN std\_logic\_VECTOR(2 downto 0);

b: IN std\_logic\_VECTOR(2 downto 0);

clk: IN std\_logic;

ce: IN std\_logic;

s: OUT std\_logic\_VECTOR(2 downto 0));

end component;

begin

main : subtracter

port map (

a => a,

b => b,

clk => clk,

ce => ce,

s => x);

end Behavioral;

13. Next, create the test bench file for the project. Make sure you save all your source files first. Right click in the Sources window and select **New Source**. Select **VHDL Test Bench** in the left window and name the file ‘**subtracter\_tb**’. Adding ‘tb’ to the end of ‘subtracter’, separated by an underscore, makes the test bench file easier to locate among a large list of files. In the next window, highlight ‘**main**’ to show that the test bench file will be associated with the .vhd file. Click **Finish**.

14. Once the test bench file is created, scroll down to where it says “COMPONENT main” and lists the components. Below where it says:

COMPONENT main

PORT(

a : IN std\_logic\_vector(2 downto 0);

b : IN std\_logic\_vector(2 downto 0);

Add the following lines if they are not there (since you didn’t save your source files as directed in step 13):

clk : IN std\_logic;

ce : IN std\_logic;

Next, scroll down the test bench file and find where it says:

--Inputs

signal a : std\_logic\_vector(2 downto 0) := (others => '0');

signal b : std\_logic\_vector(2 downto 0) := (others => '0');

Underneath, add the following lines if they are not there (since you didn’t save your source files as directed in step 13):

signal clk : std\_logic := '0';

signal ce : std\_logic := '0';

Next, find where it says,

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: main PORT MAP (

a => a,

b => b,

Underneath, add the lines if they are not there (since you didn’t save your source files as directed in step 13):

clk => clk,

ce => ce,

Next scroll through the test bench file and replace every instance of “<clock>” with “clk” (if necessary). This will allow the program to link the “clk” from the main.vhd file to the clock from the testbench.

15. When the test bench file is created, scroll down to the bottom and locate the ‘**Stimulus process**’ text in green. Beneath ‘**Stimulus process**’, delete everything between the ‘**begin**’ statement and the ‘**end process**’ statement. Enter the following text in that space.

ce <= '1' ;

a <= "100" ;

b <= "010" ;

wait for 50 ns ;

**Ce** is set to 1 in order to activate the chip. ‘**wait for 50 ns**’ is a statement that prevents a possible infinite loop. The values next to ‘a’ and ‘b’ are binary. In this example, “010” (2 in decimal form) is being subtracted from “100” (4 in decimal form). You can think of it as ‘a – b = output’. You do not have to use 100 and 010 as your input values. You can change these by replacing them with different binary input values. For example:

ce <= ‘1’ ;

a <= “101” ;

b <= “011” ;

wait for 50 ns ;

In this case, “011” (3 in decimal form) is being subtracted from “101” (5 in decimal form).

You can also do multiple subtraction problems. An example is shown below.

ce <= '1' ;

a <= "100" ;

b <= "010" ;

wait for 50 ns ;

a <= “101” ;

b <= “011” ;

wait for 50 ns ;

(Note: if you subtract numbers to get a negative number, the program will not function correctly).

When you are finished, save the test bench file. It should look like this.

--------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 11:25:41 07/19/2011

-- Design Name:

-- Module Name:

-- Project Name: subtracter

-- Target Device:

-- Tool versions:

-- Description:

--

-- VHDL Test Bench Created by ISE for module: main

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

-- Notes:

-- This testbench has been automatically generated using types std\_logic and

-- std\_logic\_vector for the ports of the unit under test. Xilinx recommends

-- that these types always be used for the top-level I/O of a design in order

-- to guarantee that the testbench will bind correctly to the post-implementation

-- simulation model.

--------------------------------------------------------------------------------

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--USE ieee.numeric\_std.ALL;

ENTITY subtracter\_tb IS

END subtracter\_tb;

ARCHITECTURE behavior OF subtracter\_tb IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT main

PORT(

a : IN std\_logic\_vector(2 downto 0);

b : IN std\_logic\_vector(2 downto 0);

clk : IN std\_logic;

ce : IN std\_logic;

x : OUT std\_logic\_vector(2 downto 0)

);

END COMPONENT;

--Inputs

signal a : std\_logic\_vector(2 downto 0) := (others => '0');

signal b : std\_logic\_vector(2 downto 0) := (others => '0');

signal clk : std\_logic := '0';

signal ce : std\_logic := '0';

--Outputs

signal x : std\_logic\_vector(2 downto 0);

-- Clock period definitions

constant clk\_period : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: main PORT MAP (

a => a,

b => b,

clk => clk,

ce => ce,

x => x

);

-- Clock process definitions

clk\_process :process

begin

clk <= '0';

wait for clk\_period/2;

clk <= '1';

wait for clk\_period/2;

end process;

-- Stimulus process

stim\_proc: process

begin

ce <= '1';

a <= "100";

b <= "010";

wait for 50 ns;

a <= “101” ;

b <= “011” ;

wait for 50 ns ;

end process;

END;

15. With the test bench file still open, double click on **Simulate Behavioral Model** in the Processes window. This will open up ISim. Zooming out will allow you to see the waveform. By scrolling back and forth, you can see what combinations of inputs result in different outputs. Next to the waveform are two columns, ‘**Name**’ and ‘**Value**’. Your inputs and outpus (‘a’ and ‘b’) are displayed under the Name column, and their values are displayed under the Value column as you scroll back and forth. In this case, however, the waveform should show that four minus two always equals two, as seen in the Value column next to x[2:0] (this is your output). Once you finish looking at the simulator, close the simulator.

16. Now, return to the ISE Project Navigator and right click in the sources window. Click on **New Source**. Click on **Implementation Constraints File**and title it ‘**main**’*.*When the .ucf file is generated, type in the following:

######### Pin assignments for top\_sequence ###########

## The Spartan 3E’s 50 MHz clock is used in the clock buffer.

NET “clk” LOC = “B8” ; # CLK\_50MHZ

NET “a<0>” LOC = “G18” ;

NET “a<1>” LOC = “H18” ;

NET “a<2>” LOC = “K18” ;

NET “b<0>” LOC = “K17” ;

NET “b<1>” LOC = “L14” ;

NET “b<2>” LOC = “L13” ;

NET “ce” LOC = “N17” ;

NET “x<2>” LOC = “K15” ; # LED<2>

NET “x<1>” LOC = “J15” ; # LED<1>

NET “x<0>” LOC = “J14” ; # LED<0>

This .ucf file will tie the program to the switches and LEDs on the FPGA board later on. The .ucf file shows that ‘clk’ is located at B8, and ‘chip enable’ is located at N17. Notice that N17 is a switch on the FPGA board. When N17 goes high, so does chip enable, thus allowing the device to function.

Notice that ‘a’ and ‘b’ each have three switches. This is because ‘a’ and ‘b’ have three digits (in binary). On the FPGA board, use these switches to represent the binary value of ‘a’ and ‘b’. A high switch represents a 1, and a low switch represents a 0. The LEDs are used to display the output. A lit LED represents a 1.

17. Next, you will create the programming file, known as the .bit file. Click on the implementation section, click on your main.vhd file, and then click on **Generate Programming File**.Once this completes, it will have created the .bit file.

18. Open up Digilent Adept and make sure your FPGA is connected and turned on. Then click on **Browse**next to the FPGA. Find your .bit file, it should be called ‘**main.bit**’. Once you have located the .bit file, open it. (A warning may appear regarding the startup clock, just click ‘yes’ if you get this). Now click **Program** to the right of where it says **Browse**. (If another warning regarding the clock appears just click ‘yes’ and continue.)

19. Once the FPGA is programmed, you can experiment with it, using the switches to subtract the values you defined in the test bench file.

**APPENDIX A** – main.vhd file

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date:

-- Design Name:

-- Module Name: main - Behavioral

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx primitives in this code.

-- library UNISIM;

-- use UNISIM.VComponents.all;

entity main is

Port ( a : in STD\_LOGIC\_VECTOR (2 downto 0);

b : in STD\_LOGIC\_VECTOR (2 downto 0);

clk : in STD\_LOGIC ;

ce: in STD\_LOGIC ;

x : out STD\_LOGIC\_VECTOR (2 downto 0));

end main;

architecture Behavioral of main is

component subtracter

port (

a: IN std\_logic\_VECTOR(2 downto 0);

b: IN std\_logic\_VECTOR(2 downto 0);

clk: IN std\_logic;

ce: IN std\_logic;

s: OUT std\_logic\_VECTOR(2 downto 0));

end component;

begin

main : subtracter

port map (

a => a,

b => b,

clk => clk,

ce => ce,

s => x);

end Behavioral;

**APPENDIX B** – subtracter\_tb.vhd file

--------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 11:25:41 07/19/2011

-- Design Name:

-- Module Name:

-- Project Name: subtracter

-- Target Device:

-- Tool versions:

-- Description:

--

-- VHDL Test Bench Created by ISE for module: main

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

-- Notes:

-- This testbench has been automatically generated using types std\_logic and

-- std\_logic\_vector for the ports of the unit under test. Xilinx recommends

-- that these types always be used for the top-level I/O of a design in order

-- to guarantee that the testbench will bind correctly to the post-implementation

-- simulation model.

--------------------------------------------------------------------------------

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--USE ieee.numeric\_std.ALL;

ENTITY subtracter\_tb IS

END subtracter\_tb;

ARCHITECTURE behavior OF subtracter\_tb IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT main

PORT(

a : IN std\_logic\_vector(2 downto 0);

b : IN std\_logic\_vector(2 downto 0);

clk : IN std\_logic;

ce : IN std\_logic;

x : OUT std\_logic\_vector(2 downto 0)

);

END COMPONENT;

--Inputs

signal a : std\_logic\_vector(2 downto 0) := (others => '0');

signal b : std\_logic\_vector(2 downto 0) := (others => '0');

signal clk : std\_logic := '0';

signal ce : std\_logic := '0';

--Outputs

signal x : std\_logic\_vector(2 downto 0);

-- Clock period definitions

constant clk\_period : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: main PORT MAP (

a => a,

b => b,

clk => clk,

ce => ce,

x => x

);

-- Clock process definitions

clk\_process :process

begin

clk <= '0';

wait for clk\_period/2;

clk <= '1';

wait for clk\_period/2;

end process;

-- Stimulus process

stim\_proc: process

begin

ce <= '1';

a <= "100";

b <= "010";

wait for 50 ns;

end process;

END;