



Configurable Space Microsystems Innovations & Applications Center

Lab 8 – Tutorial

Introduction to IP Cores with Xilinx ISE 10.1 and Digilent Spartan 3E Starter Kit Board

Introduction

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. 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 multiplier. 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 and program the Spartan 3E Starter Kit board [2].

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 double click on **Create New Source** in the Processes window. Choose **IP (Coregen & Architecture Wizard)** in the left menu, and name the file '**multi**'. In the next screen open the tree to Math Functions => Multipliers => Multiplier v10.1 as shown in figure 1. Click **Next** and **Finish**.

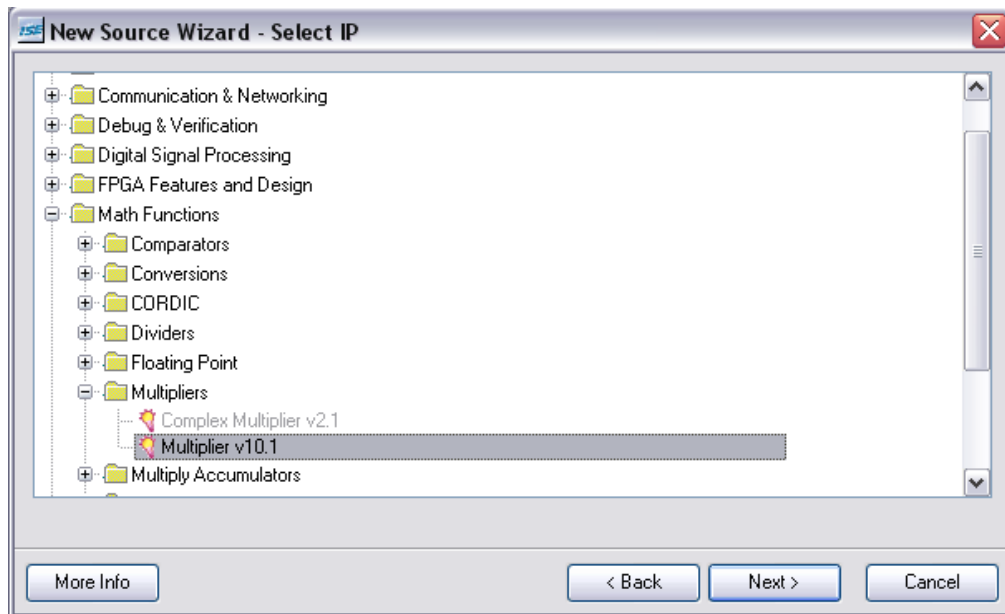


Fig. 1. New Source Wizard window

3. After a few seconds a new window (*LogiCore*) should pop up. Select the options to match those in figure. 2. Select **Finish** on figure. 2. The window will disappear and will have created a Multiplier .

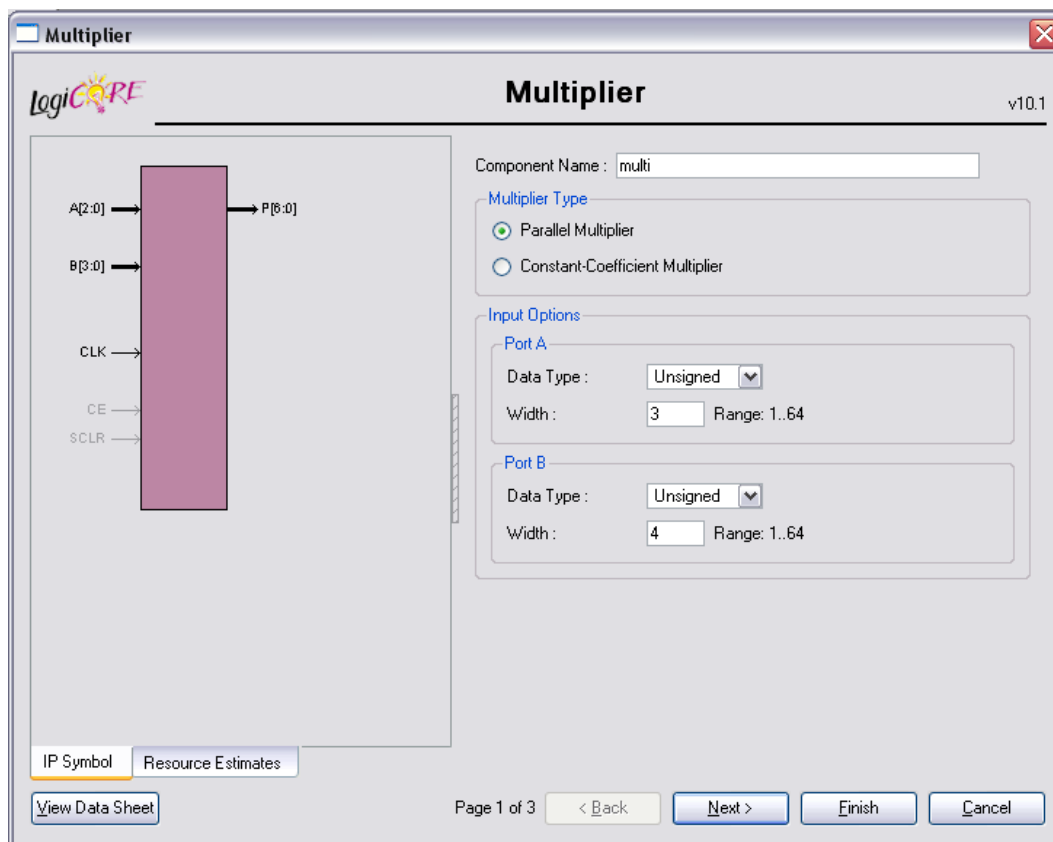


Fig. 2. First page of the LogiCore Window

Take a minute to click on the *Data Sheet...* 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.

4. It will take a few moments to generate all of the files. Progress made can be seen by looking at the messages in the *Transcript* window. When it is finished, **multi** will be shown in 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.

5. Go to **File**, click on **Open**, and browse to “**multi.vhd**”. This will bring up one of the files created by the CORE Generator System. This 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.

6. Now, click on the **New** icon to create a new VHDL file. Enter the code in the file as you see it below: (comments being optional)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
Library XilinxCoreLib;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity main is
  Port ( clk : in  STD_LOGIC;
        LED : out STD_LOGIC_VECTOR (6 downto 0);
        input : in  STD_LOGIC_VECTOR (3 downto 0));
end main;

architecture Behavioral of main is

signal A: std_logic_vector(2 downto 0);

```

Click on the *multi.vhd* file that was opened earlier in the project. Find the text which looks like this:

```

-- synthesis translate_off
component wrapped_multi
  port (
    clk: IN std_logic;
    a: IN std_logic_VECTOR(2 downto 0);
    b: IN std_logic_VECTOR(3 downto 0);
    p: OUT std_logic_VECTOR(6 downto 0));
end component;

```

Highlight and copy it. Return to the original Untitled VHDL source file and paste it in below the previous code. Below that, type a ‘**begin**’ statement. Again, return to the multi.vhd file. Find the following text:

```

U0 : multi
  port map (
    clk => clk,
    a => A,
    b => input,
    p => LED);

```

Copy it, and paste it into the VHDL source. Change “**your_instance_name**” to something shorter, such as UUT. Add an ‘**end behavioral;**’ to the end of the VHDL source file. And now, the entire VHDL source file should look like the following:

MULTIPLIER CODE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
Library XilinxCoreLib;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity main is
    Port ( clk : in  STD_LOGIC;
          LED : out STD_LOGIC_VECTOR (6 downto 0);
          input : in  STD_LOGIC_VECTOR (3 downto 0));
end main;

architecture Behavioral of main is

signal A: std_logic_vector(2 downto 0);

----- Added from Core Gen -----
component multi
    port (
        clk: IN std_logic;
        a: IN std_logic_VECTOR(2 downto 0);
        b: IN std_logic_VECTOR(3 downto 0);
        p: OUT std_logic_VECTOR(6 downto 0));
end component;
-----

begin

    A <= "101";

----- Added from Core Gen -----
U0 : multi
    port map (
        clk => clk,
        a => A,
        b => input,
        p => LED);
-----

end Behavioral;
```

7. Save the file as “**main.vhd**”. Add it to your project as an existing source. Highlight the file, and double-click **Synthesize** in the Processes window. If everything was done correctly, it should synthesize without errors. If there are any warnings, ignore them for now, and in the case that there are errors review the code and steps as pointed out below:

1. Create a new project.
2. Start the CORE Generator System
3. Create the desired IP core.
4. Create the top-level VHDL source.
5. Copy and paste in the component declaration in the generated .vhd file, and the instance declaration.
6. Save the new source file and incorporate it into your project.
7. Synthesize the source file and check for errors.
8. Create a new source **Implementation Constraints File**, and assign the pins as follows.

```
NET "CLK" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "input<3>" LOC = "N17";
NET "input<2>" LOC = "H18";
NET "input<1>" LOC = "L14";
NET "input<0>" LOC = "L13";
```

In the figure below, a behavioral simulation is displayed of the multiplier using Xilinx ISE Simulator [1]. The circuit multiplies the input by the constant 5, and displays the output on the LED's contained on the board.

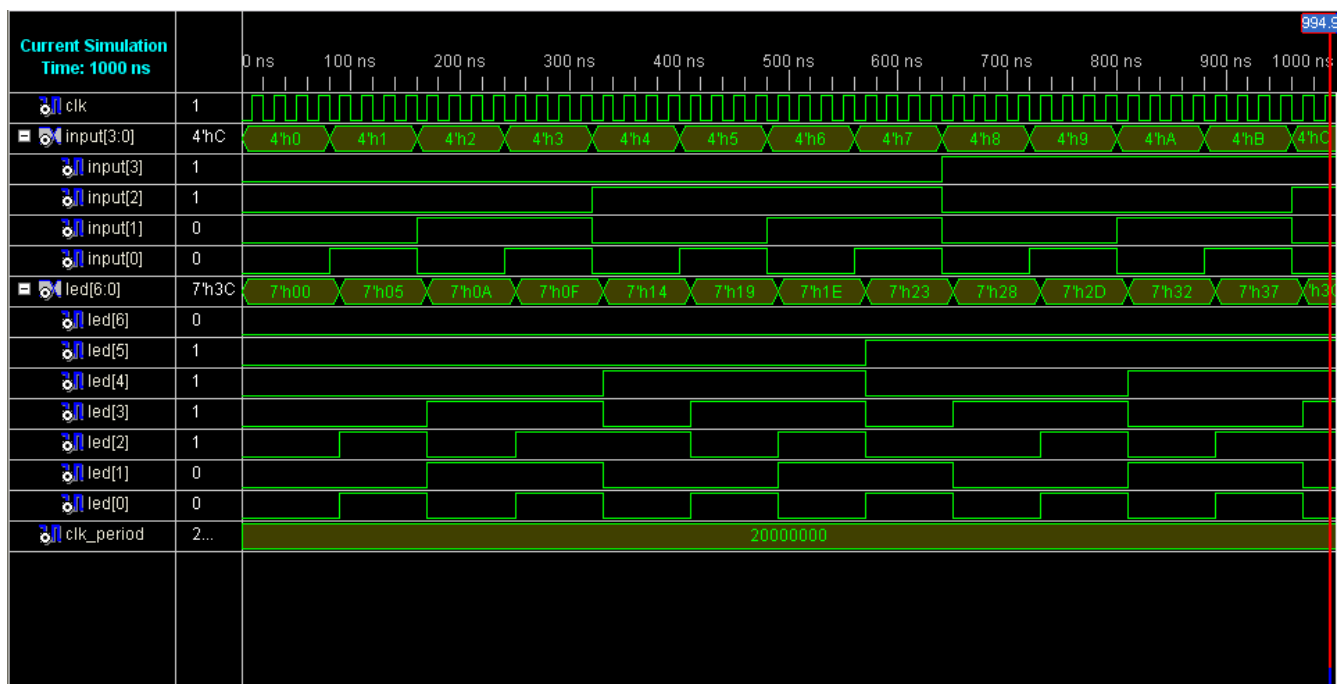


Fig. 3. Behavioral simulation of the Multiplier project.

9. Double click **Generate Programming File**. Once a green check mark appears, double click over Configure Device (iMPACT) and follow the instructions in the wizard to program the board. Test the logic by moving switches. The switches supply the circuits input while LED0 to LED6 display the output. The binary value entered in the switches (input) will be multiplied by 5 (0xb 101) and displayed on the LED's in binary. The MSB is found on LED6. See Figure 4.

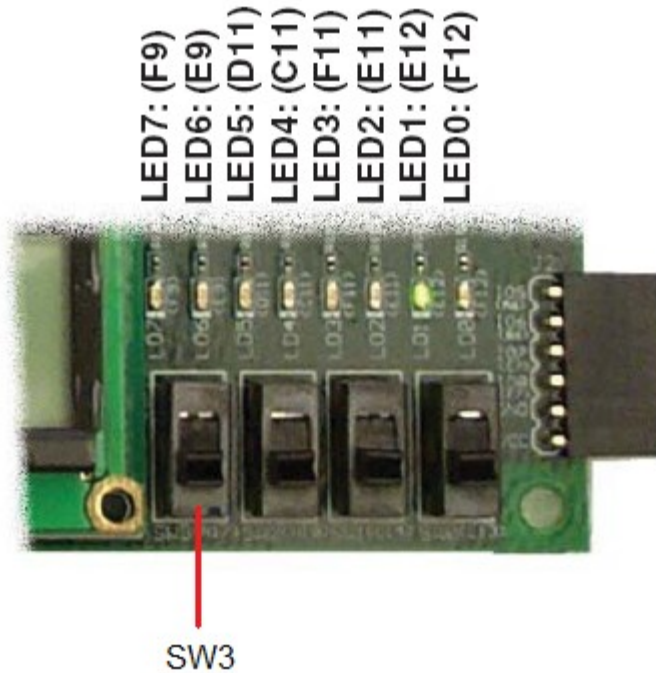


Fig. 4. Leds associated to the output of the multiplier and the operation control switches included in the Digilent Spartan 3E Starter board.

References

- [1] *ISE In-Depth Tutorial*. Xilinx Inc. Copyright 1995-2009. Available: <http://www.xilinx.com/itp/xilinx10/books/manuals.pdf>
- [2] *Spartan-3E Starter Kit Board User Guide*. Xilinx Inc. Revision UG230 March 9, 2006. pp. 121-122. Available: http://www.digilentinc.com/Data/Products/S3EBOARD/S3EStarter_ug230.pdf

About the author:

Jose Marcio Luna Castaneda
M.S. in Electrical Engineering Student
The University of New Mexico
Office: ECE Building Room 223, Phone: 277-1372
Areas of Interest: Multi-agents, robotics, hybrid systems, computational intelligence.
e-mail: jmarcio@ece.unm.edu

Brian Zufelt
Senior at The University of New Mexico