Multiplier Core VHDL Lab

Multiplier Core VHDL Lab

Introduction

The purpose of this lab is to demonstrate the use of cores in chip-level VHDL designs. Cores are customized, parameterized macros that are used to develop higher complexity designs, with maximum performance, minimum CLB usage, and most importantly, much lower design time.

Objective

After completing this lab, students will be able to:

- Use Xilinx CORE Generator to create a custom module.
- Instantiate a custom macro based on a core into a VHDL-based design.
- Understand importance of linking and hierarchy in VHDL designs.

Procedure

Create a new FPGA Express Project

1) Start Foundation Express by clicking on **Start → Programs → Xilinx Foundation Series → Accessories →Foundation Express**

2) Once inside Foundation Express, click on **File → New…**

3) Browse to the **`C:\F15_labs\cores\vhdver`** directory, and type "**`corv`**" in the **Name:** edit box. Click <**Create**>.

4) In the **Identify Sources** window, click <**Cancel**>. Notice there is now a project created, but no sources have been identified.

5) If you were really creating a new design, you would probably create your own VHDL code. To save time, let's assume that instead you will copy the source VHDL code. This has been done, so that there are already some *.VHD files copied into **`C:\F15_labs\vhdver`** directory.

6) Highlight **WORK** under the Design Sources window. Select **Synthesis → Identify Sources…**, or just click on the + symbol to add sources. 

7) Select **xivhd.vhd** under the default directory **C:\F15_labs\cores\vhdver**, and click <**Open**>. FPGA Express does syntax and other preliminary checks.

8) Double-click **xivhd.vhd** to expand the two entities within **xivhd**. With **xivhd.vhd** highlighted, right-click and select **Edit File** to view the VHDL code.
You can compare this design to the schematic **C:\F15_labs\cores\schver.pdf** and note the similarities. One significant difference between the two designs is that the VHDL code does not explicitly define **IBUF**s for all of the switch and parallel port inputs. That is because the FPGA Express tools automatically generate proper buffers according to the design in an automatic process called **IO synthesis**.

Instantiate a VHDL Core using Xilinx CORE Generator

9) Open **CORE Generator**: **Start → Programs → Xilinx CORE Generator → CORE Generator**.

10) Under the **Core Generator Options** window, check **EDIF Implementation Netlist** and **VHDL Instantiation Template** as the output products. Click <**OK**>.

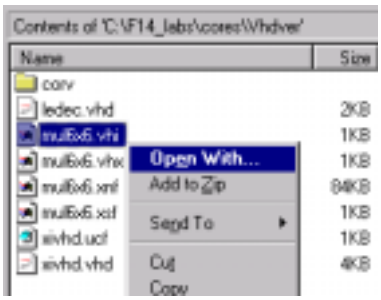
11) Under the **CORE Generator v1.5.0** window, select **Options → System options…** and type in the **Project Path** "**C:\F15_labs\cores\vhdver**". Click <**OK**>.

12) Double-click to expand the directories in this order: **LogiCORE → Math → Multipliers**. Double-click **Parallel Multiplier – Area Optimized**.

13) Type "**mul6x6**" under **Component Name**. Set **A** and **B** input bus widths to **6** and **6**. (Even though we are going to build a 4x4 multiplier, the smallest component available is 6x6.) Select **Unsigned**.

14) Click <**Generate**>. This creates several files. **Mul6x6.vhi** is the VHDL file instantiation file. **Mul6x6.vhx** is the simulation template. **Mul6x6.xnf** is the underlying Xilinx Netlist File which will be included in the flattened project netlist for implementation, later.

15) Open **Windows Explorer** (or NT Navigator) and find **C:\F15_labs\cores\vhdver\mul6x6.vhi**. Highlight this file, right-click it, and select **Open With…** Find **WORDPAD** in the list of applications, highlight it, and click <**OK**>.
Notice that there is a component **initialization** followed by a component **instantiation** for **mul6x6**. Highlight the following text from within **mul6x6.vhi :**



```
component mul6x6 port (
      a: IN std_logic_VECTOR(5 downto 0);
      b: IN std_logic_VECTOR(5 downto 0);
      c: IN std_logic;
      ce: IN std_logic;
      prod: OUT std_logic_VECTOR(11 downto 0));
end component;
```

16) Once it is highlighted, copy this text by holding <**Ctrl**> + <**C**> on your keyboard (or **Edit → Copy**).

17) Go to the **HDL Editor** that shows the **`xivhd.vhd`** VHDL code. You should paste (<**Ctrl**> + <**V**>) the instantiation text below this line,

```
architecture str_xivhd of xivhd is
```

and above this line,

```
component OSC4 port (
```

Make sure not to accidentally paste any non-ASCII characters (they look like '▮' or '□') into the VHDL text file.

18) We also need to add an instance of the multiplier core to the VHDL code. Add the bolded lines, as shown below, between the **osc** and **togmux** lines. Note that the specific port names differ slightly from the instantiation code generated from the CORE Generator. This is done in order to maintain consistency with the net names in the corresponding schematic version of the design.

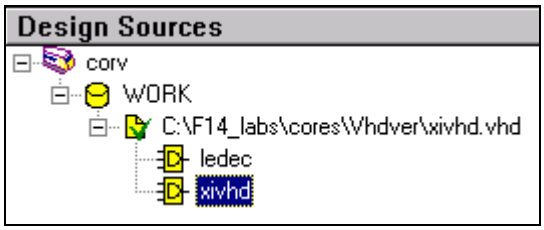
```
osc : OSC4 port map(F15 => clk);

mul : mul6x6
        port map (Ain, Bin, clk, ce, prod(7 downto 0) => PROD);

togmux : process(LNE, PRAA, PRBB3, PRBB2, PRBB1, PRBB0, SWAA,
SWBB)
```

19) If you have any trouble with steps 17 or 18, you can just copy the correct, edited file from **`C:\F15_labs\shortcut\xivhd.vhd`** to **`C:\F15_labs\cores\vhdver\xivhd.vhd`**.

20) Select **File → Save** to save the modified VHDL file.

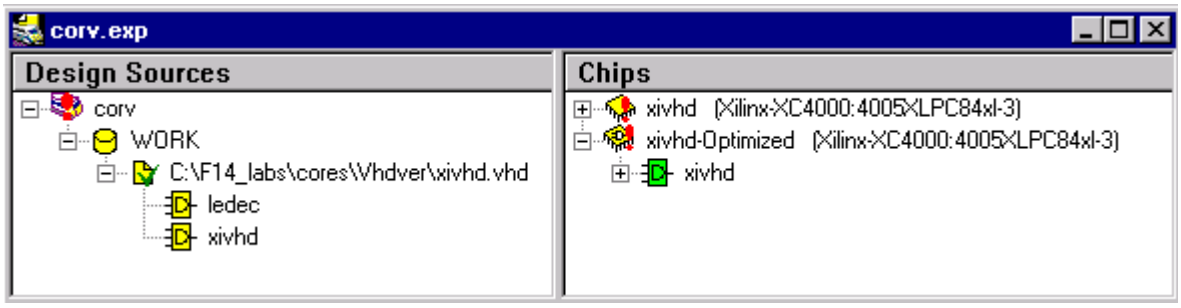
21) Back in FPGA Express, highlight **`xivhd.vhd`**, right-click, and select **Force Update File** to re-analyze the changes.

22) Since **`xivhd`** is the top-level design, highlight this component, as shown below.



23) Then select **Synthesis → Create Implementation**. Since this is a complete VHDL design, eg. Top-level VHDL design, leave "**Do not insert I/O pads**" unchecked. Make sure the Target device is XC4005XL-3PC84. Click <**OK**>.
You will notice some warnings, especially indicating that some of the mul6x6 module's nets are not linked. This is OK, since we only are connecting inputs for a 4x4 multiplier, rather than a 6x6 multiplier.

24) Under the **Chips** window, highlight **`xivhd`**, right-click, and select **Optimize.Chip**. At this point, FPGA Express should look something like this:



25) Right click on **`xivhd-Optimized`** and select **Export Netlist**. Make sure you go up one to the top directory, **`C:\F15_labs\cores\vhdver`** . Click <**Save**>.

Creating and Implementing the new project in Xilinx Design Manager

Now that the complete VHDL design has been designed and synthesized, we must implement the netlist into the Xilinx device. Although pin location constraints could have been included in the FPGA Express constraints editor, we have set up pin location constraints in a ***.UCF** file in advance. So, let's open a new project and implement our design.

26) **Start → Programs → Xilinx Foundation Series → Accessories → Design Manager.**

27) Select **File → New Project…**

28) Under **Input Design**, browse to find **`C:\F15_labs\cores\vhdver\xivhd.xnf`** . Select this and choose <**Open**>, then <**OK**>.

29) The rest is simple. Highlight **`xivhd`** in Design Manager, select **Design Implement**.

30) We want to make sure that the proper UCF file is being used, so click <**Options…**>. The User Constraints window should show
`C:\F15_labs\cores\vhdver\xivhd.ucf`.

31) Click <**OK**>. Check that the right part is targetted (XC4005XL-3-PC84). Click <**Run**>.

32) When finished compiling, download the
`C:\F15_labs\cores\vhdver\xivhd.bit` bitstream and test the multiplier to make sure it works in both the **XSPORT mode**, in which A and B 4-bit inputs come from the PC's parallel port, as well as the **Switch mode**, where A and B inputs come from the 8-bit switch.

Conclusion

In this project,we created a core using CORE Generator, and instantiated it into the VHDL code. The design functions identically to the schematic version of the multiplier design.