Simulation Lab 4
Register and Buffers

Use Xilinx ISE Project Navigator and FPGA Editor to investigate Look-up tables (LUTs) and Registers

National
Science
Foundation

Funded in part, by a grant from the
National Science Foundation
DUE 1003736 and 1068182

Acknowledgements

Developed by Craig Kief, Alonzo Vera, Alexandria Haddad, and Quinlan Cao, at the Configurable Space Microsystems Innovations & Applications Center (COSMIAC). Based on original tutorial developed by Bassam Matar, Engineering Faculty at Chandler-Gilbert Community College, Chandler, Arizona. *Funded by the National Science Foundation (NSF).*

Lab Summary

This lab will present design entry and simulation of registers and buffers. It will show the difference between a clocked register and an unclocked buffer. It will also investigate how the logic is created in the Look-Up Tables of the fabric of the Spartan Device.

Lab Goal

The goal of this lab is to continue learning how to use the Xilinx design tools. The goal of the lab is to introduce students to the inner workings of an FPGA using the FPGA Editor.

Learning Objectives

1. Create a simple project in Xilinx® ISE.
2. Add existing file sources to the new project.
3. Compile and simulate the design.
4. Review RTL Schematic.
5. Investigate the implementation using the FPGA Editor.
6. Download the design to the FPGA board.

Grading Criteria

Your grade is 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

Students should work in teams of two or three. Each team of students will need the following supplies:

Supplies	Quantity
1. ISE® Design Suite (or WebPACK™) software from the Xilinx website, www.xilinx.com, if you don't already have it installed. Your classroom should have a full working version of Xilinx ISE® Design Suite. 2. FPGA kit including download and power cable(s). 3. Free Digilent Adept software (instructions for download and installation are included at the beginning of this lab): http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,828&Prod=ADEPT2	1

Additional References

The FPGA reference manual, data sheet, and any other supporting documents that may be of use.

ISE Design Suite User Guide :
http://www.xilinx.com/support/index.htm#nav=sd-nav-link-106173&tab=tab-dt

Digilent Adept User Guide:
http://www.digilentinc.com/Data/Software/Adept/Adept%20Users%20Manual.pdf

Introduction to Registers and Buffers

Registers and Buffers serve a unique role in the digital world and as such, they are developed differently in VHDL. This tutorial will explain these devices; their differences and similarities. The tutorial will then show how each of them is created in VHDL. The tutorial will show how to display the Resistor Transistor Logic (RTL) schematic associated with each.

Although the RTL schematic provides a convenient method for investigating the flow of designated systems, it really isn't completely accurate in how the circuitry is actually developed. The actual circuitry is created using items called Configurable Logic Blocks or CLBs. These internal sections of the FPGA will also be investigated in an attempt to remove the "black box" magic of FPGAs and to actually show the student how the hardware is dynamically configured using the supplied VHDL.

A register is a one-bit storage element. Since it stores a single bit, it is the building block for things such as memories. It also changes values based on clock edges. Comparatively, a buffer is nothing more than a repeater. It helps improve fan-out. Fan-out is the ability to drive many outputs off of a single input without excessively loading the circuit. What you will notice is that the buffer's output changes as soon as the input changes.

Inside the FPGA – CLBs, Slices and LUTs

There are two main parts to the FPGA. These parts are the Configurable Logic Blocks (CLBs) and the Input/output Blocks (IOBs). CLBs are the board's main building block of logic resources and are used to implement sequential and combinatorial circuits. The main working components of the FPGA are very simple. They consist of a register (D-flip flop), some control logic and a look-up table (LUT). The LUT can be thought of as a file cabinet of basic function. The input control lines to the LUT determine which function will be accomplished.

Each CLB in the Spartan 6 contains two slices, with each slice containing four Look-Up Tables (LUTs) to implement logic and eight D Flip-Flops. These functions consist of basic digital logic equations (such as AND, OR, XOR, etc.). The average FPGA consists of tens of thousands of these LUTs. There are three types of slices used in the Spartan 6 FPGA. Table 1 lists the logic function(s) that are contained in each slice type.

Function	SliceM	SliceL	SliceX
LUTs (Logic Function Generator or Look-Up Table)	4	4	4
D Flip-Flops	8	8	8
Wide Multiplexers	✓	✓	
Carry Logic	✓	✓	
Distributed RAM	✓		
Shift Registers	✓		

Table 1 Spartan 6 Slice Features

Figure 1shows the inner workings of ¼ of SliceX of the Spartan 6 FPGA. For a more detailed diagram of each type of slice in the Spartan 6 refer to the Spartan-6 FPGA, Configurable Logic Block User Guide (http://www.xilinx.com/support/documentation/user_guides/ug384.pdf).

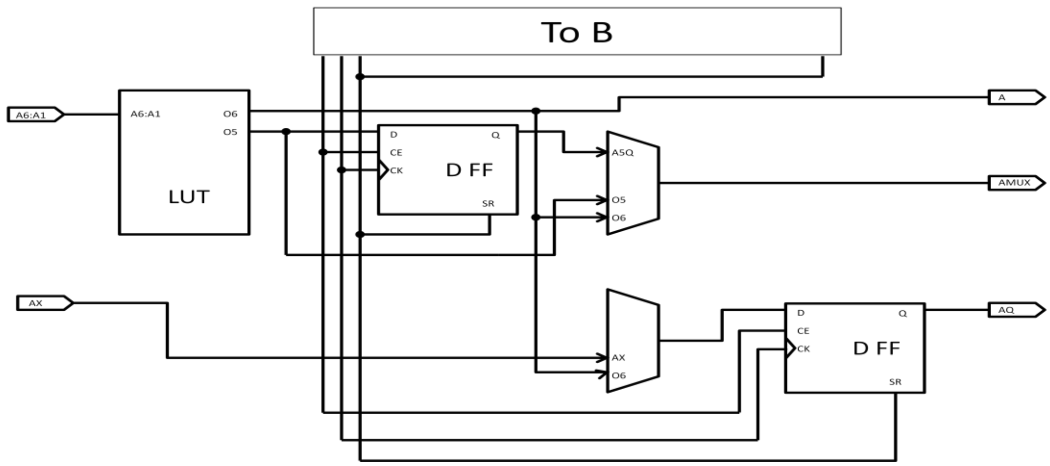


Figure 1 Simplified Section A, SliceX of Spartan 6

Figure 2 shows a top down view of the Spartan device. It is important to have knowledge of the FPGA you are working with. If you are doing an extensive FPGA project, it is highly recommended that you invest the time required to read the User's Manual(s) of your FPGA. Learn as much as you can before you attempt to write the VHDL for your project. The number of slices contained in an FPGA's CLBs varies, depending on the family of FPGA chosen. However, the general concept is same.

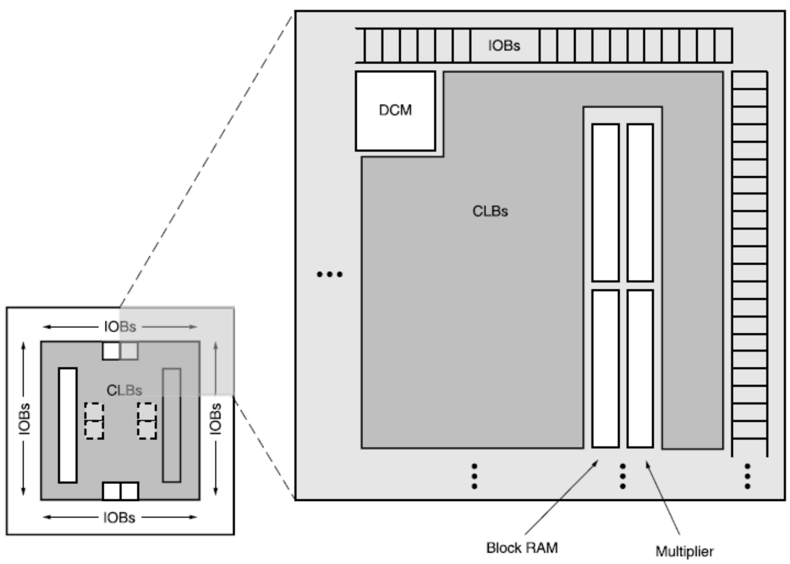
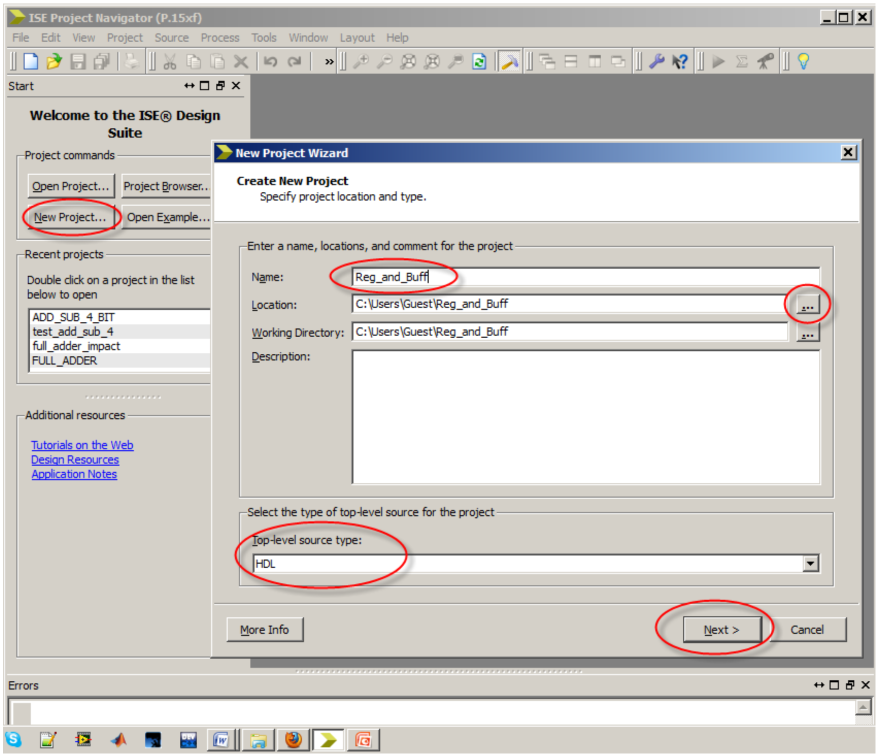


Figure 2 Logic Placement in Spartan FPGA

Lab Procedure 1: Registers and Buffers

Step 1: Create Ne Project Reg_and_Buff and Add Source Files

1. Open **Xilinx ISE Design Tool Project Navigator**.
2. Click **OK** to close the Tip of the Day.
3. Start a new project by selecting **File → New Project** from the menu or click the **New Project** button. The New Project Wizard starts.
   a. Type **Reg_and_Buff** in the Name text box.
   b. Select a location on your computer to save your project files by clicking the ellipsis (**...**) button to the right of the **Location** text box.
   c. Under **Top-level source type**, select **HDL**.
   d. Click **Next**.
4. The Project Settings dialog box opens.



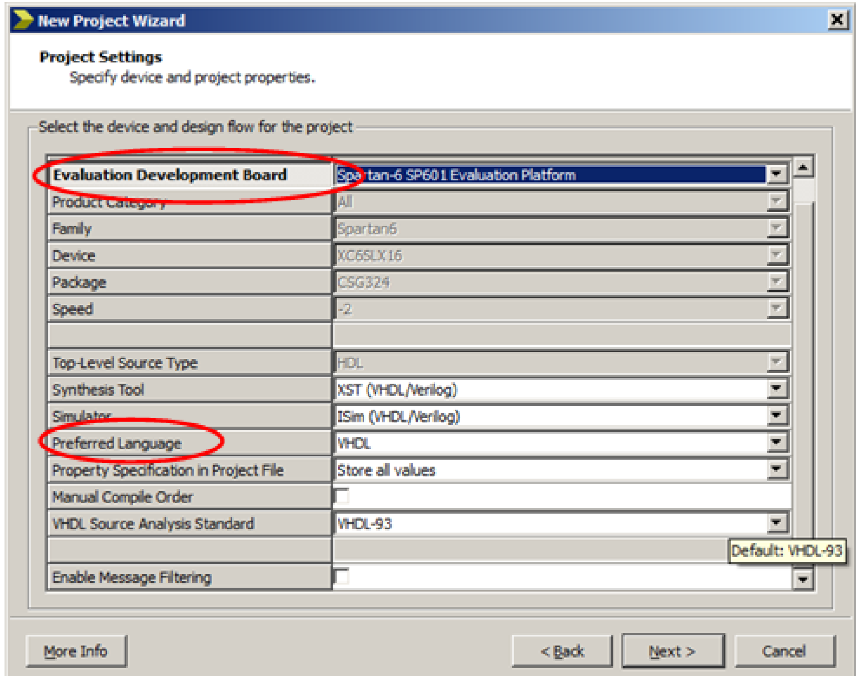
NOTE: File names must start with a letter. Use underscores (_) for readability. Do not use hyphens (-); although the file name will work, the entity name will not. We recommend using directories that **do not** have spaces in the directory names, i.e. avoid “My Documents.”

5. Select **Spartan-6 SP601 Evaluation Platform** from the **Evaluation Development Board** drop down menu.

   The **Product Category**, **Family**, **Device**, **Package**, and **Speed** should all automatically populate (top half of the screen). You will need to set some options in the lower half of the dialog box.

   Select **VHDL** from the **Preferred Language** drop down menu.

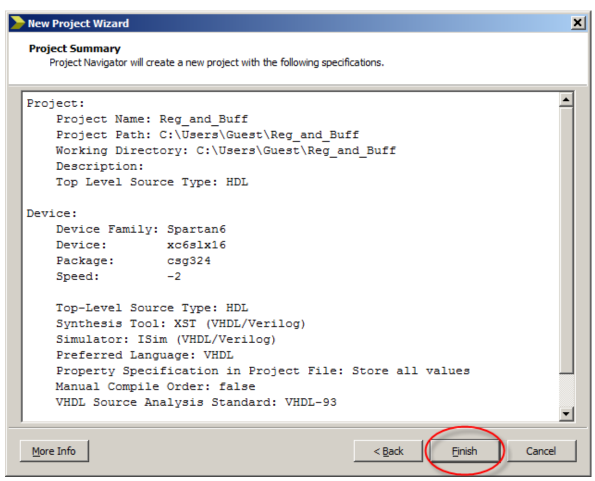
   The Project Settings should resemble the figure to the right.



6. Click **Next**.

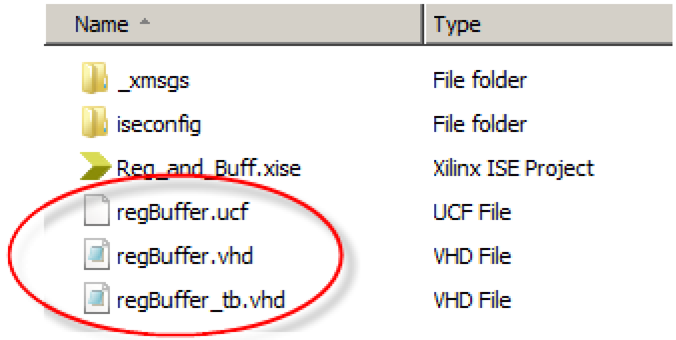
NOTE: The options specified are for the **Spartan 6 LX FPGA**. Your board might be different than the board used for these instructions. Board specifications are printed on the FPGA chip in the middle of the board. The board information is also listed on the box it came in.

7. The Project Summary displays. Verify the file type and name are correct then click **Finish** to complete the New Project creation process.



NOTE: There are three main files associated with this tutorial. The project files are in the **regbuf.zip** file. If you don't have the zip file you can download it from the COSMIAC website (http://cosmiac.org/Projects_FPGA.html).

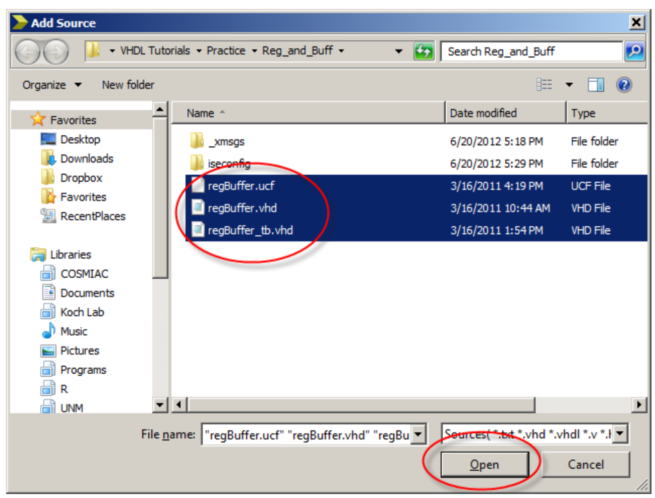
We will be adding the files from the regbuf.zip file to our new project.



8. Using Windows Explorer, **extract** or **move** these three files to your project folder.

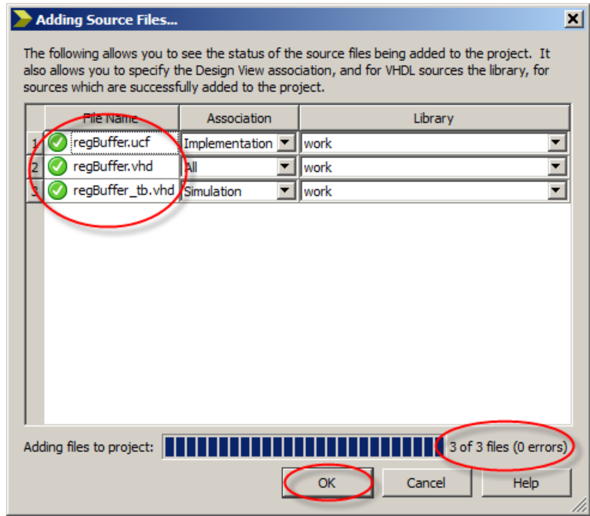
9. Select **Project → Add Source** from the menu or **Right-click** in the Hierarchy pane and select **Add Source** from the shortcut menu. The **Add Source** dialog box opens.

10. **Select** the three files that were extracted to the project directory earlier. **Click** the **Open** button. The **Adding Source Files…** dialog box opens.



11. Click the OK button to complete the add files process.

NOTE: All three project files display with a green checkmark to the left of the file name. This lets us know that ISE understands the design association of each of the files.

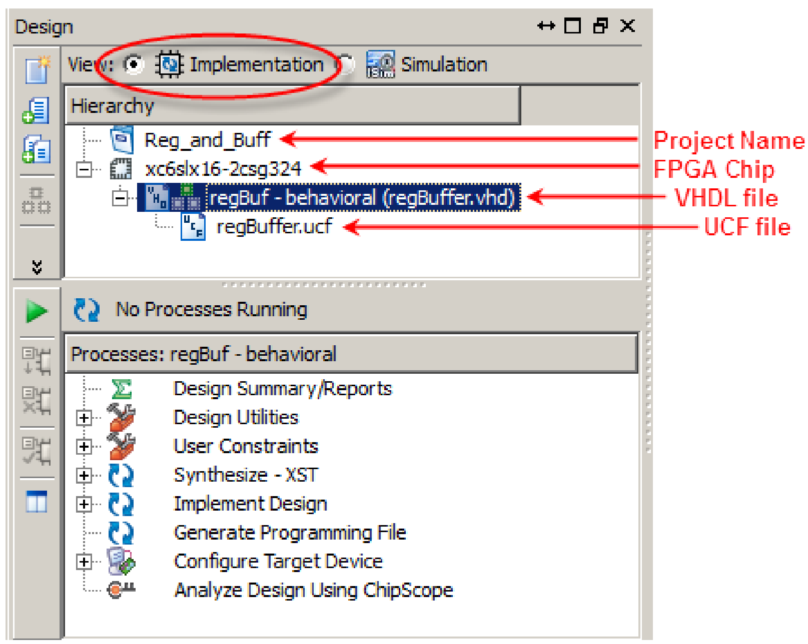


12. The files are added to the **Reg_and_Buf** project. Clicking the Implementation or Simulation options displays the files that were just added.

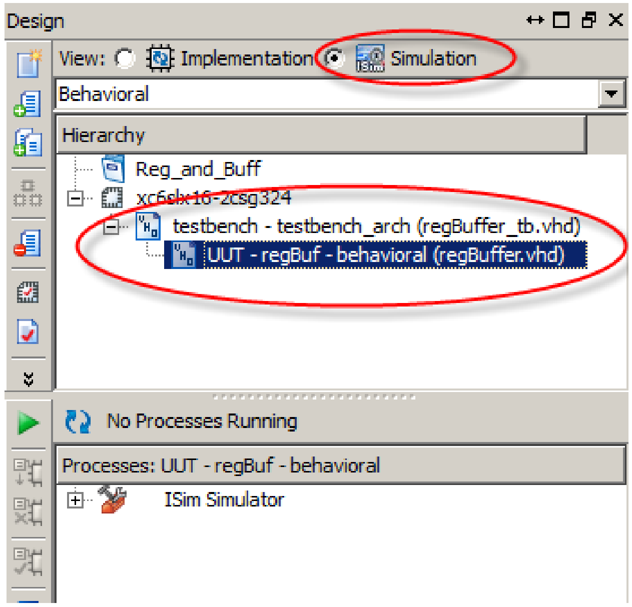
NOTE: It is important to understand the file Hierarchy pane. The files display in a tree structure similar to the file and directory structure on a computer.

When Implementation is selected the UCF file should be below the design file it is associated with.

This image shows that at the top level is the project, then the chip type. Under the chip is the VHDL file and the associated UCF file.



Similarly, when Simulation is selected, the UUT should be under the associated testbench file.



Step 2: VHDL Overview

One key to successful VHDL design is file naming. In the beginning, most of your projects will only consist of one design file. That quickly changes. It is very common to see large VHDL FPGA projects with tens or hundreds of design and test files. To organize this well, use intelligent naming conventions. For example, if you have an **adder** module it makes sense to name your VHDL file something similar, such as **adder.vhd**.

Let's review the VHDL code. This will provide some understanding of syntax and flow of the code. Some things to keep in mind:

a. VHDL is designed for parallel operations. As such, multiple processes can be contained in a single design file. Operations within a process are acted upon sequentially (similar to a programming language). Processes within a design file are operated upon concurrently. This provides the ability for parallel operations.

b. VHDL is portable.

c. VHDL allows modeling and simulation of a system before expensive hardware is created.

d. VHDL is NOT case sensitive, i.e. CLK is the same as clk, which is the same as Clk.

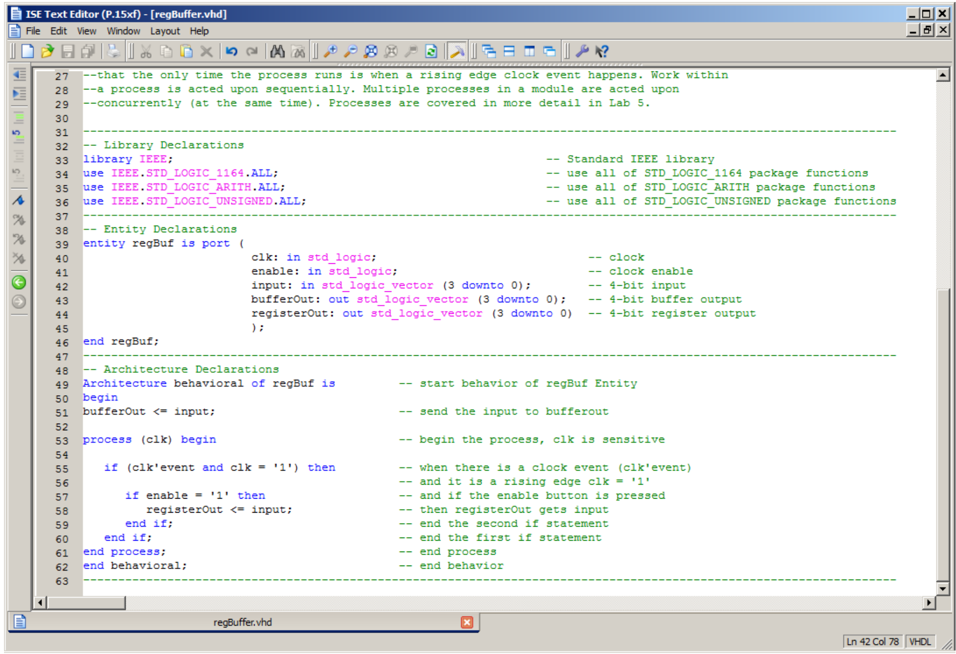
e. Semicolons are used to end VHDL statements; very much like in C or C++.

f. Comments, proceeded by two hyphens (--), are good, should be used often, and will be helpful to your replacement in case you die.

g. Assignment operators are different in VHDL than in other languages (Pedroni 47):

<=	is used to assign a value to a signal
:=	assigns a value to a variable
=>	assigns values to individual vector elements or OTHERS

1. Double click **the regBuf – behavioral** (**regBuffer.vhd**) file name. The **regBuffer.vhd** file opens in the Workspace.



2. Double click the **regBuffer.ucf** file to review the UCF code.

```
# The pin assignments in this file are for the
# Nexys 3, Spartan 6 XC6LX16-CS324
# if you are using a different board, change the
# LOC = pin assignment as appropriate

NET "clk" LOC = "V10";                # clk location, as referenced in the Data Sheet
NET "enable" LOC = "C4";              # BTNL
NET "input(0)" LOC = "T10";           # SW0
NET "input(1)" LOC = "T9";            # SW1
NET "input(2)" LOC = "V9";            # SW2
NET "input(3)" LOC = "M8";            # SW3
NET "bufferOut(0)" LOC = "U16";       # LD0
NET "bufferOut(1)" LOC = "V16";       # LD1
NET "bufferOut(2)" LOC = "U15";       # LD2
NET "bufferOut(3)" LOC = "V15";       # LD3
NET "registerOut(0)" LOC = "M11";     # LD4
NET "registerOut(1)" LOC = "N11";     # LD5
NET "registerOut(2)" LOC = "R11";     # LD6
NET "registerOut(3)" LOC = "T11";     # LD7
```

NOTE: Notice that the **clk** is not assigned to a button or a switch. We first assigned the **clk** to the left button (BTNL) and received the error message shown below. The board's data sheet (http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf) provided the correct pin location for the **clk** assignment.

```
Errors
ERROR:Place:1108 - A clock IOB / BUFGMUX clock component pair have been found
   that are not placed at an optimal clock IOB / BUFGMUX site pair. The clock
   IOB component <clk> is placed at site <C4>. The corresponding BUFG component
   <clk_BUFGP/BUFG> is placed at site <BUFGMUX_X2Y3>. There is only a select set
   of IOBs that can use the fast path to the Clocker buffer, and they are not
   being used. You may want to analyze why this problem exists and correct it.
   If this sub optimal condition is acceptable for this design, you may use the
   CLOCK_DEDICATED_ROUTE constraint in the .ucf file to demote this message to a
   WARNING and allow your design to continue. However, the use of this override
   is highly discouraged as it may lead to very poor timing results. It is
   recommended that this error condition be corrected in the design. A list of
   all the COMP.PINs used in this clock placement rule is listed below. These
   examples can be used directly in the .ucf file to override thiERROR:Pack:1654
   - The timing-driven placement phase encountered an error.
```

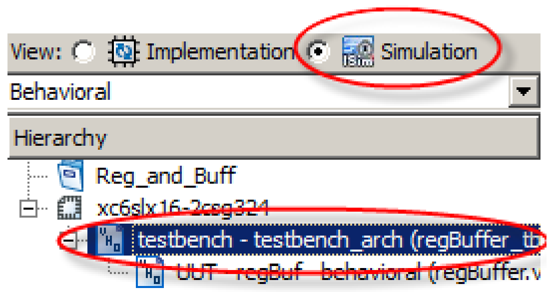
Step 3: Testbench Overview

When designing a VHDL system usually individual modules are created and tested. Once all the modules are built and tested they are then combined into a larger system which is then tested. The testing or simulation is done with testbench files.

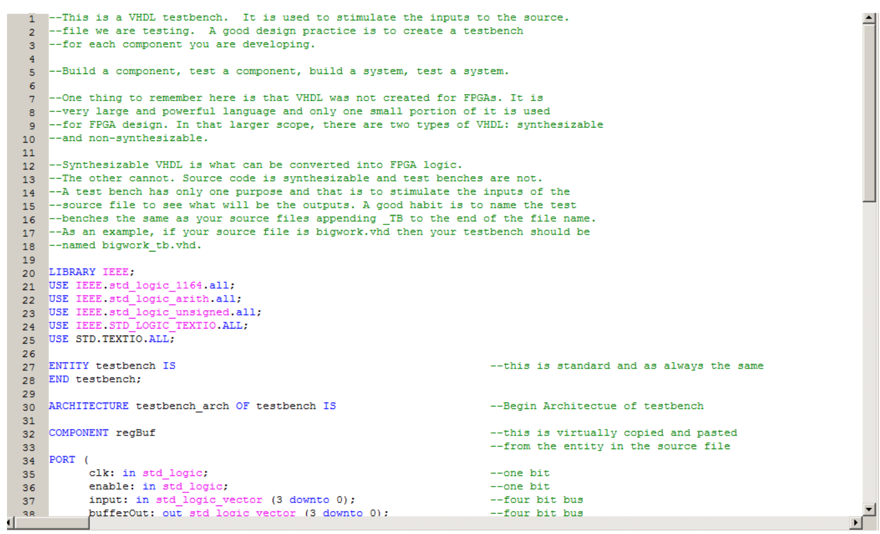
A testbench is not synthesizable code. It cannot be implemented in the hardware. Its only purpose is to stimulate your design to perform testing. Always make sure the testbench is highlighted before attempting to run the simulator.

The test module is called the testbench. It is good practice to give the testbench file the same name as the file it is simulating and by adding **_TB** to the end of the file name. E.g. if your vhd file is called **adder.vhd**, name your testbench **adder_TB.vhd**.

1. Click the **Simulation** View.
2. Select the **regBuffer_tb.vhd** file. The **ISim Simulator** displays in the **Processes** Pane.
3. Expand the **ISim Simulator** if it is not already expanded by clicking the **plus** sign.
4. Double click the testbench file to open it in the work space.

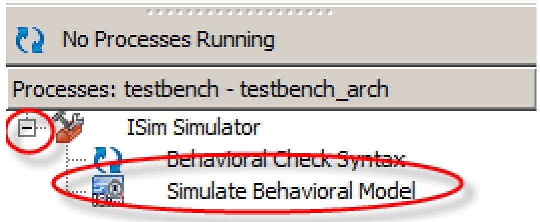


5. **Review** the testbench file and make sure you understand how it is working. The file is heavily commented for clarity.

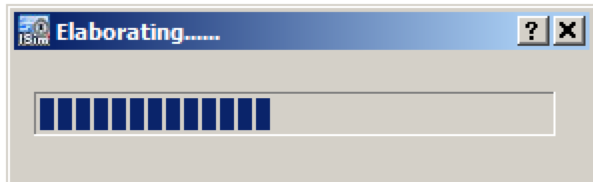


```
--This is a VHDL testbench.  It is used to stimulate the inputs to the source.
--file we are testing.  A good design practice is to create a testbench
--for each component you are developing.

--Build a component, test a component, build a system, test a system.

--One thing to remember here is that VHDL was not created for FPGAs. It is
--very large and powerful language and only one small portion of it is used
--for FPGA design. In that larger scope, there are two types of VHDL: synthesizable
--and non-synthesizable.

--Synthesizable VHDL is what can be converted into FPGA logic.
--The other cannot. Source code is synthesizable and test benches are not.
--A test bench has only one purpose and that is to stimulate the inputs of the
--source file to see what will be the outputs. A good habit is to name the test
--benches the same as your source files appending _TB to the end of the file name.
--As an example, if your source file is bigwork.vhd then your testbench should be
--named bigwork_tb.vhd.

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE IEEE.std_logic_unsigned.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY testbench IS                                 --this is standard and as always the same
END testbench;

ARCHITECTURE testbench_arch OF testbench IS         --Begin Architectue of testbench

COMPONENT regBuf                                    --this is virtually copied and pasted
                                                    --from the entity in the source file
PORT (
      clk: in std_logic;                            --one bit
      enable: in std_logic;                         --one bit
      input: in std_logic_vector (3 downto 0);      --four bit bus
      bufferOut: out std_logic_vector (3 downto 0); --four bit bus
```

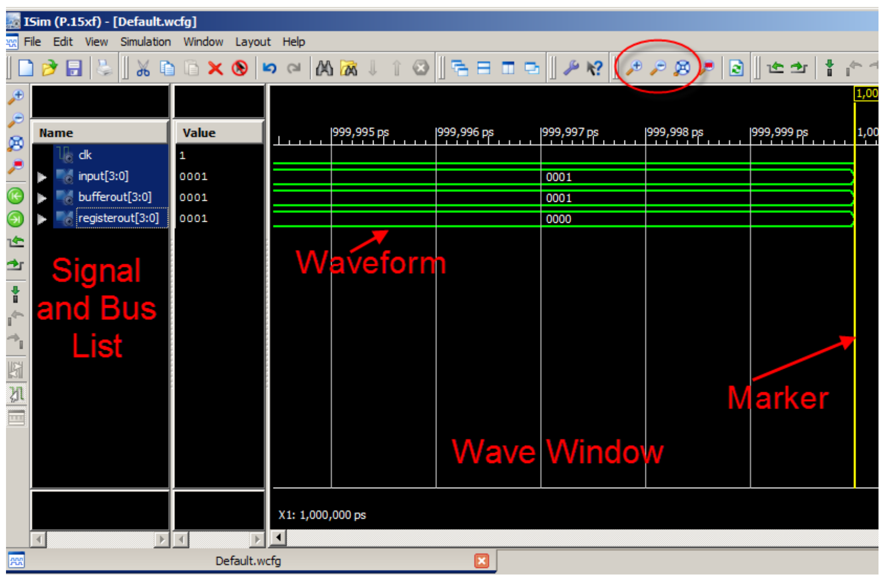
6. Double click the **Simulate Behavioral Model** option under the **ISim Simulator**.



7. The simulator will start and the **Elaborating** status will display. Once the simulation is complete the ISim application window will open.



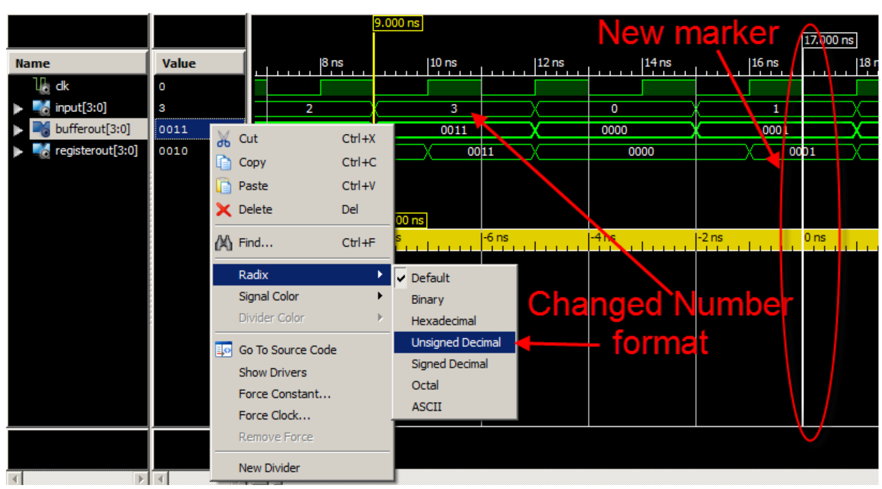
8. Click the **Zoom to Full View** button , then click **Zoom In** to better view the simulation.



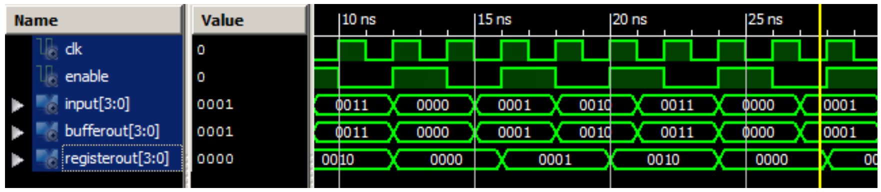
Some things to **Note**:

a. Add markers by right clicking in Wave Window and selecting **Markers → Add Marker**.

b. To change the way the numbers display Right-click the **Value** label and select **Radix → *Number type***.



9. Observe that the **bufferout** signal is not controlled by the **clk**. **Bufferout** drops when the input signal drops. However, the **registerout** signal is only affected by the combination of the **input** signal and the **clk**.

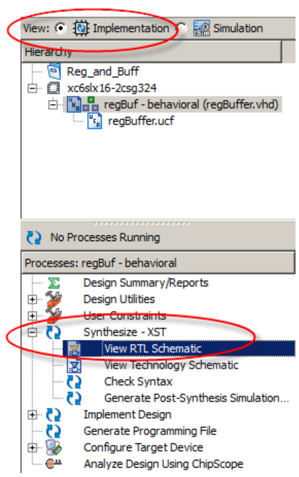


Step 4: View RTL (Resistor Transfer Level) Schematic

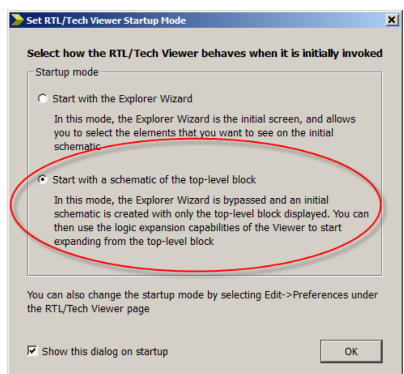
The View RTL Schematic tool is an excellent product for visualizing the circuit that the Hardware Description Language (VHDL) creates. This view allows a designer to see how the Xilinx software believes the design should look, which provides an excellent medium for understanding and debugging circuits. Having read the previous section, you are aware that the circuit is created with registers and LUTs.

In larger designs, often a signal will show up one clock pulse before or after when you think it will arrive. This view will help you understand why.

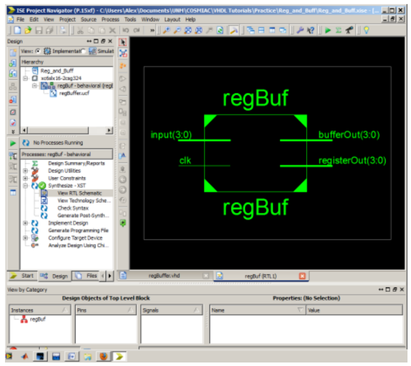
1. Switch to the **Implementation** view.
2. Select the **regBuf – behavioral (regBuffer.vhd)** file in the **Hierarchy Pane**.
3. Expand the **Synthesize – XST** process in the **Processes Pane**.
4. Double click the **View RTL Schematic** option.



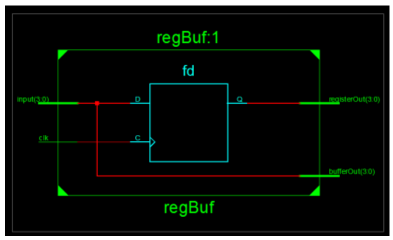
5. Select **Start with a schematic of the top-level block**, if it isn't already selected.
6. Click the **OK** button.



7. The schematic displays in the workspace. This first image shows how the overall inputs (left) and outputs (right) are developed.



8. Double click the **regBuf** box in the workspace. The circuit will expand to the next level.

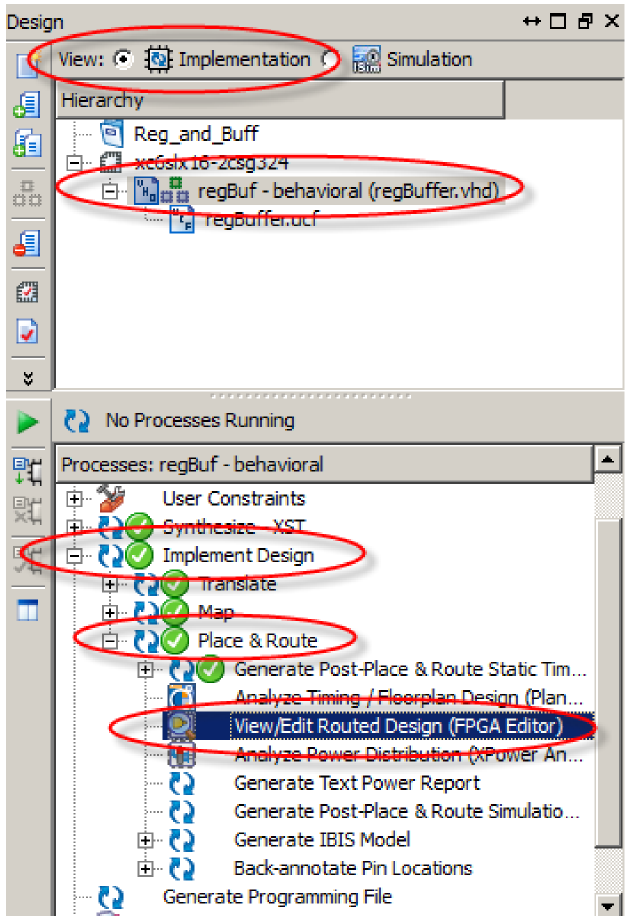


As shown, the registered output goes through D-flip flop (fd). This is because the registered output is driven by the clock while the buffered output is not. As a result, its operation will be erratic. It is good practice when designing a circuit with a clock (synchronous) to drive all outputs based on the rising edge (or falling edge) of the clock.

9. **Close** the RTL view.

Step 5: Using the FPGA Editor

The final section of this tutorial will explore a little more about what was discussed previously in the **Inside the FPGA – CLBs, Slices and LUTs** section.

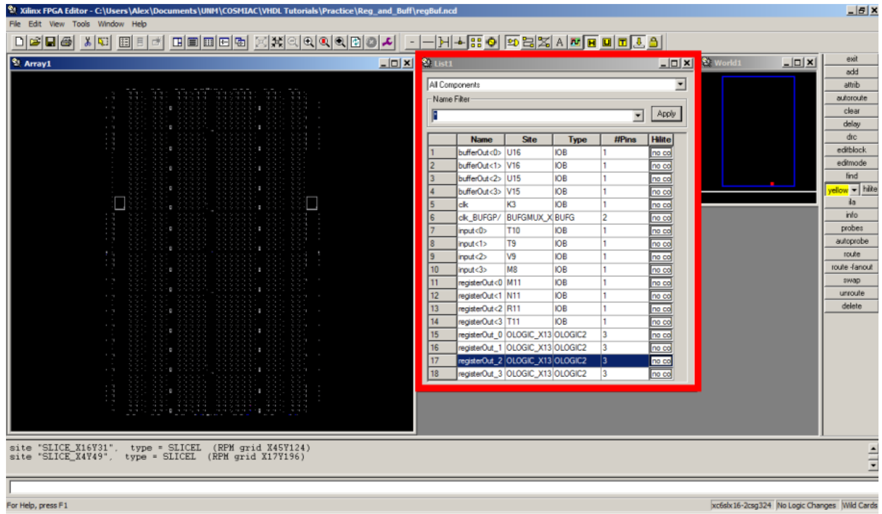


1. Switch to the **Implementation** view and select the **regBuf – behavioral (regBuffer.vhd)** file in the **Hierarchy Pane** (if not already displayed and selected).
2. Expand the **Implement Design** option in the **Processes Pane**.
3. Expand the **Place & Route** section.
4. Double click on the **View/Edit Routed Design (FPGA Editor)** selection.

NOTE: If you get mapping errors you may be using a board other than the Nexys 3, Spartan 6 XC6LX16-CS324 board. The pin assignments in the UCF file are for the Spartan 6. If you are using a different board, open the UCF file and change the LOC = pin assignment as appropriate.

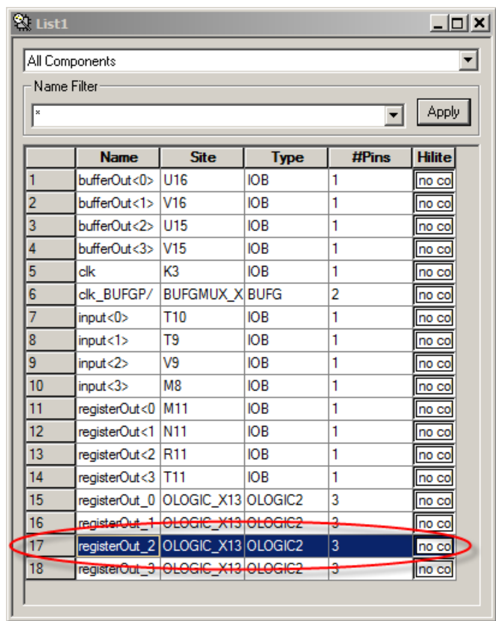
5. The Xilinx FPGA Editor starts and displays an image of the Spartan 6 chip (or whatever chip you are using).

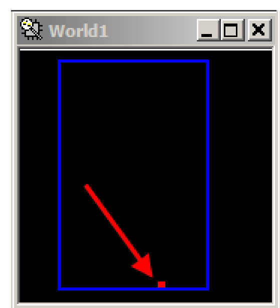
NOTE: We've resized the windows in the workspace area for better visibility. You may want to resize the **List1** window so that you can view all the inputs and outputs of the circuit.



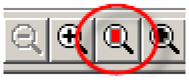
6. Select any component from the list. We used Component 17 registerOut_2.

7. **NOTE**: When a component is selected notice that a red square displays in the World window. This is a small preview of where that particular component is located on the chip.

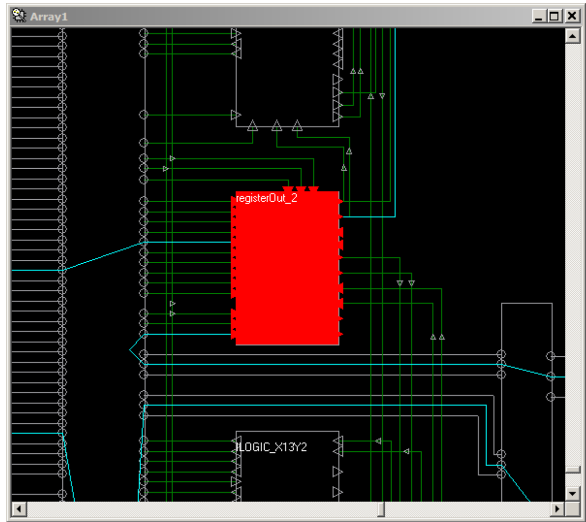




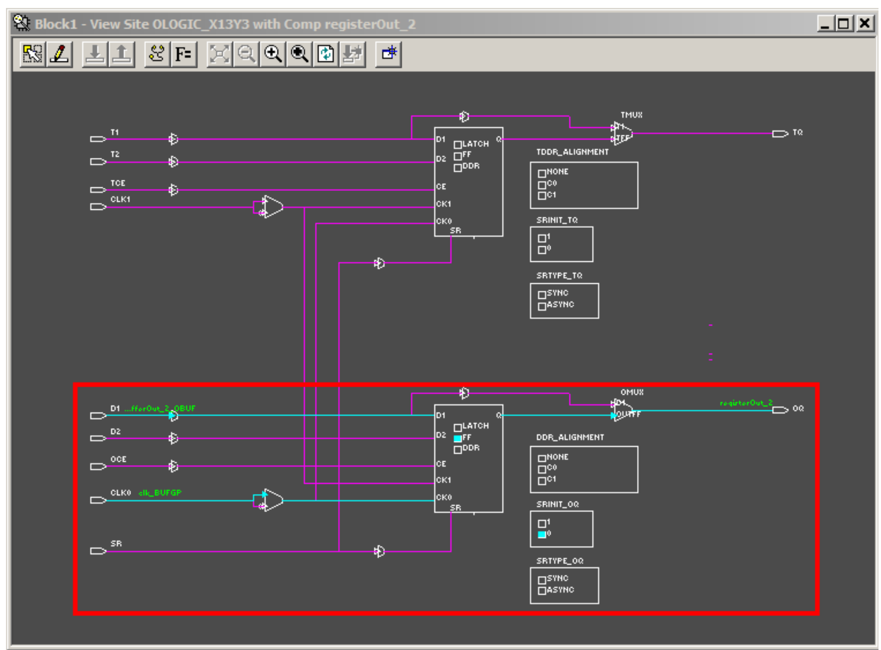
8. Click the Zoom Selection button on the toolbar to zoom into the selected component's slice.



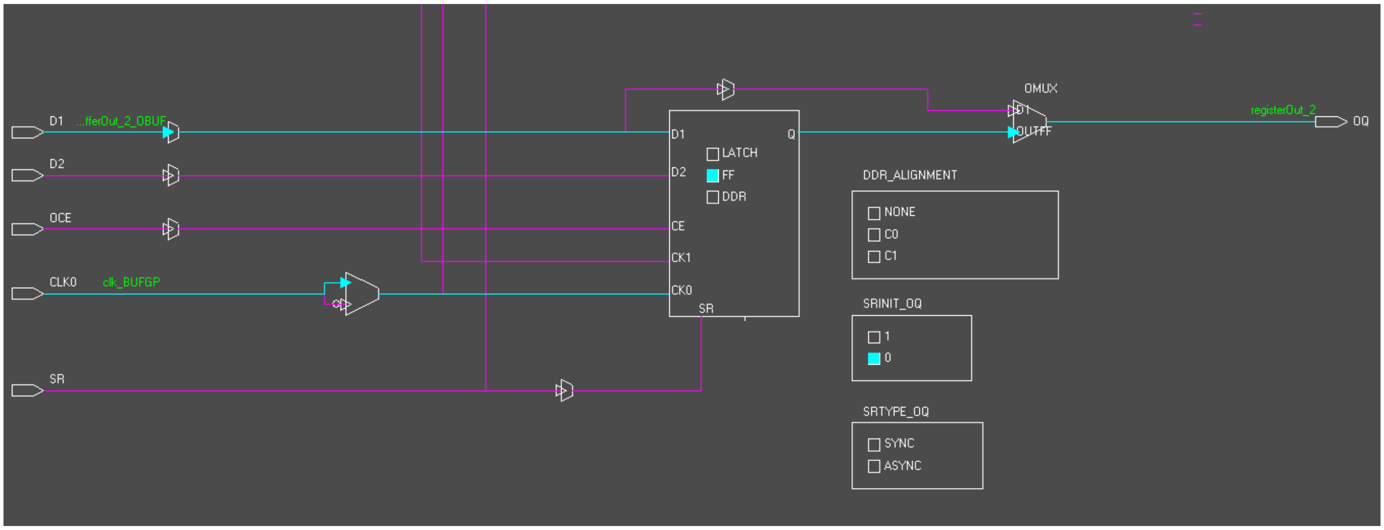
The editor zooms into the section of the chip where the slice is located as displayed in the Array1 window.



9. Double click the red slice in the Array1 window. The FPGA Editor displays the Block window which shows how the design is created in the slice.

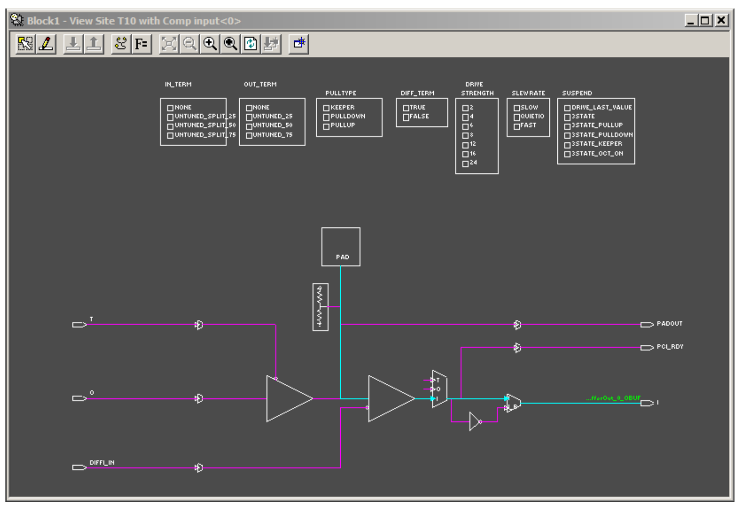


10. Zoom in to display the register.



11. Close the Block window and select another component to view. We chose component 7, input<0>.

    It is important to remember there is no “black box” magic about FPGA projects. If necessary, the design can drill down to the exact circuit for every item described in the VHDL and the register can be instantiated as several different items.



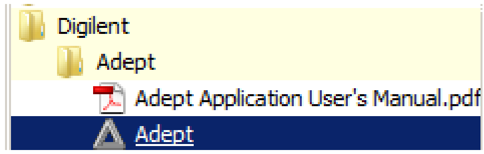
12. Close the FPGA Editor when you are finished exploring.

Step 5: Implement design to FPGA board

Because the FPGAs we are working with use a single USB cable for power and PC connection we need to use Digilent Adept to implement the .vhd code to the board.

1. Open **Digilent Adept** by selecting Start → All Programs→ Digilent → Adept → Adept

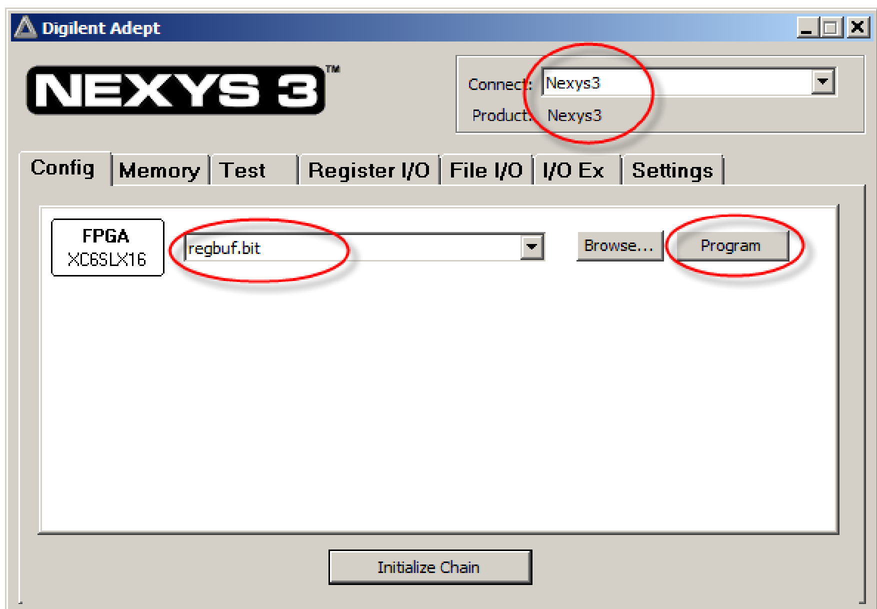
   Your system might have slightly different Start Menu options.



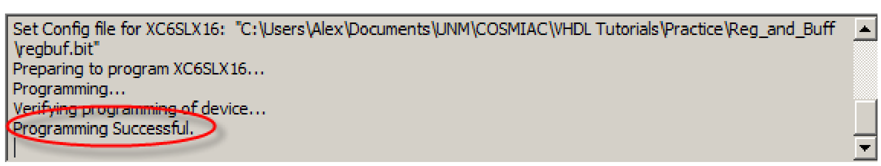
4. Choose **Nexys3** from the **Connect Product** drop down menu.

NOTE: If you do not see Nexys3 as an option, check the USB connection to ensure the board is connected to the computer.

5. Click the **Browse** button. The Open dialog box displays.
6. Navigate to the folder of the **Reg_and_Buf** project and select the **regbuf.bit** file.
7. 21. Click the **Open** button. The file is displayed in the dropdown list next to the FPGA icon.
8. Click the **Program** button. The bit file is programmed to the FPGA board.



9. Once the program is downloaded to the FPGA board the status window will display **Programming Successful**.



10. Close Adept and close the FULL_ADDER project.

Lab Procedure 2: Test your Knowledge

1. Create a new ISE project called **New_RegBuf**.
2. Find and include the three files from the **Reg_and_Buff** project:

   regBuffer.ucf
   regBuffer.vhd
   regBuffer_tb.vhd

3. Run the **testbench**.
4. View the **RTL schematic**.
5. Display the **FPGA Editor**.
6. Program the FPGA.
7. Answer the following questions:

   a. How does the regBuffer.vhd work? What does it do?

   ____

   ____

   ____

   b. How many CLBs are in the Spartan 6 FPGA board?

   ____

   c. How many LUTs and Flip-Flops are in a Spartan 6 slice?

   ____

   d. Why are comments a good idea?

   ____

   e. Is VHDL case sensitive?

   ____

   f. What punctuation ends each VHDL statement?

   ____

g. What are the three assignment operators and what does each do?

h. What are the three sections of a synthesizable VHDL file?

i. Where are inputs and outputs defined?

j. Where are signals defined?

k. Do multiple processes run one at a time or all at once?

l. What functions are defined in IEEE.STD_LOGIC_ARITH?

m. What functions are included in the following statement use IEEE.STD_LOGIC_1164.ALL; ?

n. What is good practice for naming testbenches?

o. Can testbench files be programmed to the FPGA board?

Bibliography

Pedroni, Volnei A. *Circuit Design with VHDL*. Cambridge: MIT Press, 2004. Print.