

Adding the ILA and VIO Cores for Remote Monitoring and Control Lab

Introduction

This lab consists of adding the ILA and VIO cores with the Core Generator tool in the ChipScope™ Pro software for monitoring and controlling a design.

The files for this lab are for the Spartan®-3E 1600 device on the MicroBlaze Development Kit but can also be used for other Xilinx devices.

Objectives

After completing this lab, you will be able to:

- Use the Core Generator to add two ChipScope Pro software cores to a design
- Define and use the VIO core and the VIO Console
- Control and monitor a design by using the ChipScope Pro Analyzer tool
- Describe advantages and disadvantages of both ChipScope Pro software flows (Core Inserter versus Core Generator)

Procedure

Software requirements:

- ChipScope Pro 11.3 software
- ISE® 11.3 software

Hardware requirements:

- Spartan-3E 1600 FPGA MicroBlaze Development Kit
- USB configuration cable (included in the kit)
- Power supply for the Spartan-3E 1600 FPGA board (included the kit)

The design used for this lab is a counter design targeting the Spartan-3E 1600 FPGA MicroBlaze Development Kit board. The design generates an incrementing shift pattern on the LEDs which change at a 1-Hz rate.

This lab is separated into steps, followed by general instructions and supplementary detailed steps allowing you to make choices based on your skill level as you progress through the lab.

If you need help completing a general instruction, go to the detailed steps below it, or if you are ready, simply skip the step-by-step directions and move on to the next general instruction.

This lab comprises four primary steps: You will configure the design by using the ChipScope Pro Analyzer tool; use the Core Generator to add ILA and VIO cores to the design; modify the count design in order to add the cores; and, finally, configure and control the design.

Before beginning, verify that your software is properly installed.

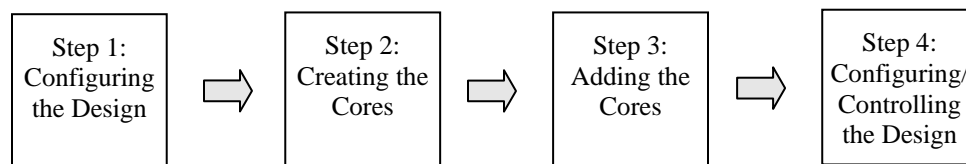
- ChipScope Pro software: Select **Start** → **Programs** → **Xilinx ISE Design Suite 11** → **ChipScope Pro** → **Analyzer**. Select **Help** → **About** to verify version 11.3
- Project Navigator in the ISE software: Select **Start** → **Programs** → **Xilinx ISE Design Suite 11** → **ISE** → **Project Navigator**. Select **Help** → **About** to verify version 11.3

Verify that the hardware is set up properly by checking the following settings.

- Connect the USB programming cable from your machine to the board (there is no need for the Platform Cable USB adapter, only the USB cable)

- Power the board on—the PC should detect the new Xilinx programming device (install the drivers if prompted)

General Flow for this Lab



Configuring the Design

Step 1

1-1. This is a simple counter design (identical to the Core Inserter tool flow lab) and is working properly. The task in this lab is not to debug, but rather gain experience with the Core Generator tool in the ChipScope Pro software as well as the ILA and VIO cores. The design uses one of the available pushbuttons to reset the clock to zero. The design uses eight LEDs to reflect activity by shifting at a 1-Hz rate. For the purposes of this lab, assume that the board is slated for environmental testing to confirm the proper operation of the FPGA in various environments.

For those not familiar with environmental testing, the board will be placed inside a sealed chamber and the temperature and humidity will be cycled through a number of profiles which are designed to mimic the stress and strain placed upon the board over multiple years of service. Because this is a sealed chamber, you will not have an opportunity to see the LEDs nor press the various buttons.

To facilitate the testing, you will add an ILA and VIO core to this design to allow remote control and monitoring of the design and board.

Open the *CoreGenFlow.xise* design located in the *C:\training\chipscope_pro\labs\CoreGenFlow-VHDL* (VHDL users) or *C:\training\chipscope_pro\labs\CoreGenFlow-Verilog* (Verilog users) folder.

- 1-1-1.** Select **Start** → **All Programs** → **Xilinx ISE Design Suite 11** → **ISE** → **Project Navigator** to open the Project Navigator in the ISE software.
- 1-1-2.** From the Project Navigator, select **File** → **Open Project**.
 - **VHDL users:** Browse to *C:\training\chipscope_pro\labs\CoreGenFlow-VHDL*
 - **Verilog users:** Browse to *C:\training\chipscope_pro\labs\CoreGenFlow-Verilog*
- 1-1-3.** Select *CoreGenFlow.xise* and click **Open**.
- 1-2.** **Download the bitstream to the device and verify its functionality.**
 - 1-2-1.** In the Sources window, select *exampleCoreGen*
 - 1-2-2.** In the Processes window, double-click **Generate Programming File** to generate the bitstream.
 - 1-2-3.** In the Processes window, double-click **Analyze Design Using ChipScope** which will launch the ChipScope Pro Analyzer tool.
 - 1-2-4.** When the Analyzer Tool opens, select **JTAG Chain** → **Xilinx Platform USB Cable**.

- 1-2-5.** In the dialog box that opens, note the USB-JTAG speed and port number and click **OK**.
This is one of several way to initialize the JTAG chain.
- 1-2-6.** Click **OK** in response to the summary of devices in the JTAG chain.
- 1-2-7.** Right-click the **XC3S1600E** device in the Project pane and select **Configure**.
- 1-2-8.** Click **Select New File** from the JTAG Configuration area of the dialog box that opens.
- **VHDL users:** Browse to *C:\training\chipscope_pro\labs\CoreGenFlow-VHDL*
 - **Verilog users:** Browse to *C:\training\chipscope_pro\labs\CoreGenFlow-Verilog*
- 1-2-9.** Select *exampleCoreGen.bit* and click **Open**. Then click **OK** to close the Configuration dialog box and configure the device.
- Observe the configuration status bar in the lower-right corner of the ChipScope Pro Analyzer display during the configuration process. Close the ChipScope Pro Analyzer window.
- Observe the correct operation of the timer design (via the LEDs) on the Spartan-3 FPGA MicroBlaze Development Kit board.
- 1-2-10.** Exit the ChipScope Pro software. Do not save any changes.

Creating the Cores

Step 2

- 2-1.** You can use an ILA core to monitor signals within the design and a VIO core to control the reset and show slower board activity.

To accomplish this, use the ICON, ILA, and VIO cores. The ILA and VIO cores require the following parameters.

ILA core:

- Define one trigger port with nine inputs: eight for LEDs and one for reset
- Use RPMs and Data same as Trigger
- Disable storage qualification and trigger output port
- Sample on the rising edge of the clock and select a sample depth of 1024
- Use a single Match Unit configured as “Basic with Edges”

VIO core:

- Define eight inputs for LEDs and one input to monitor reset status
- Define two outputs: one as a reset and the other as a VIO console/board select, allowing you to control the board from either the VIO console or the board

Using the Core Generator, generate an ICON core to use in this design.

- 2-1-1.** From the ISE Project Navigator, select **Project** → **New Source** to open the New Source Wizard. Click on **IP (Core Generator & Architecture Wizard)** and enter a name of **Controller** in the File name box. Click **Next** to launch the CoreGen tool and create the ICON core.

2-1-2. Expand Debug & Verification. Expand ChipScope Pro. Select **ICON (ChipScope Pro -Integrated Controller)** and click **Next** and click **Finish**.

2-1-3. In the ICON Parameters dialog box, define the number of control ports as **2**. Leave the rest of the options at their default. Click **Generate**.

This generates the ICON core—this process can take between one and two minutes.

2-2. Using the specifications given at the beginning of this step, use the Core Generator to generate an ILA core to use in this design.

2-2-1. From the ISE Project Navigator, select **Project** → **New Source** to open the New Source Wizard. Click on **IP (Core Generator & Architecture Wizard)** and enter a name of **ILA** in the File name box.

2-2-2. Expand Debug & Verification. Expand ChipScope Pro. Select **ILA (ChipScope Pro - Integrated Logic Analyzer)** and click **Next** and click **Finish**.

2-2-3. For the Trigger Port Settings, set the following.

- **Number of Trigger Ports:** 1
- **Max Sequence Levels:** 1
- **Use RPMs:** enabled (checked)
- **Enable Trigger Output Port:** disabled (unchecked)

2-2-4. For the Storage Settings, set the following.

- **Sample On:** Rising
- **Sample Data Depth:** 1024
- **Enable Storage Qualification:** disabled (unchecked)
- **Data Same as Trigger:** enabled (checked)

Click **Next**.

2-2-5. For the Trigger Port 1 settings, set the following.

- **Trigger Port Width:** 9
- **Match Units:** 1
- **Counter Width:** Disabled
- **Match Type:** basic with edges
- **Exclude Trigger Port:** disabled (unchecked)

Click **Generate**.

2-3. Using the specifications given at the beginning of this step, use the Core Generator tool to generate an VIO core to use in this design.

2-3-1. From the ISE Project Navigator, select **Project** → **New Source** to open the New Source Wizard. Click on **IP (Core Generator & Architecture Wizard)** and enter a name of **VIO** in the File name box.

2-3-2. Expand Debug & Verification. Expand ChipScope Pro. Select **VIO (ChipScope Pro - Virtual Input/Output)** and click **Next** and click **Finish**.

2-3-4. Select **Enable Synchronous Input Port** (do not use Asynchronous) and define a width of **9** bits.

2-3-5. Select **Enable Synchronous Output Port** (do not use Asynchronous) and define a width of 2 bits.

2-3-6. Click **Generate**.

Adding the Cores (VHDL)

Step 3a

3a-1. Verilog users: Proceed to Step 3b.

Now that the cores have been built, they must be instantiated in the design.

Create the instantiation templates for the **ICON, ILA, and VIO** cores¹.

3a-1-1. In the Hierarchy window, select *Controller*.

3a-1-2. In the Processes window, expand **CORE Generator** and double-click **View HDL Instantiation Template**.

This opens the instantiation template for the ICON core in the editor window.

3a-1-3. In the Hierarchy window, select *ILA*.

3a-1-4. In the Processes window, double-click **View HDL Instantiation Template**.

This opens the instantiation template for the ILA core in the editor window.

3a-1-3. In the Hierarchy window, select *VIO*.

3a-1-4. In the Processes window, double-click **View HDL Instantiation Template**.

This opens the instantiation template for the VIO core in the editor window.

3a-2. Add the component core definitions to the *exampleCoreGen.vhd* file from the *Controller.vho*, *ILA.vho*, and *VIO.vho* files.

3a-2-1. In the *Controller.vho* file, select and copy the text between the *Begin Cut here for COMPONENT Declaration* statement and the *End COMPONENT Declaration* statement.

3a-2-2. At the bottom of the editor pane, select the *exampleCoreGen.vhd* tab.

3a-2-3. Locate the comment indicating where the ChipScope Pro tool component definitions should be pasted near line 77.

3a-2-4. Paste the *Controller* component definition here.

3a-2-5. Repeat this procedure for the ILA component using the *ILA.vho* file.

3a-2-4. Repeat this procedure for the VIO component using the *VIO.vho* file.

Because the *attribute syn_black_box : boolean;* line of code is repeated in the second and third component definition, these lines of code must be deleted or commented out for the second and third occurrence.

3a-2-5. Clean up the attributes.

The completed code is at the end of this lab.

¹ The language that the template will be created in is selected in Design Properties under Preferred Language.

3a-3. Instantiate the Controller (ICON) core.

3a-3-1. In the *Controller.vho* file, copy the text between the *INST_TAG*s (**Figure 1**).

```

-- COMP_TAG_END ----- End COMPONENT Declaration -----
-- The following code must appear in the VHDL architecture
-- body. Substitute your own instance name and net names.
----- Begin Cut here for INSTANTIATION Template ----- INST_TAG

your_instance_name : Controller
  port map (
    CONTROL0 => CONTROL0,
    CONTROL1 => CONTROL1);

-- INST_TAG_END ----- End INSTANTIATION Template -----

```

Figure 1. Locating and Capturing the ICON Instance (VHDL)

3a-3-2. In the *exampleCoreGen.vhd* file, move to just after the architecture's *begin* statement and paste the instantiation immediately following the comment which indicates the ChipScope Pro tool instantiations (near line 135).

Technically, these cores can be instantiated anywhere between the architecture's *begin* and *end* statements, but for uniformity, please place the core at the location specified.

3a-3-3. Rename the instantiation to *exampleICON*.

3a-4. Instantiate the ILA core.

3a-4-1. In the *ILA.vho* file, copy the text of the instantiation between the *INST_TAG*s.

3a-4-2. In the *exampleCoreGen* file, paste the instantiation just after the instantiation of the ICON core (near line 139).

3a-4-3. Rename the instantiation to *exampleILA*.

3a-5. Instantiate the VIO core.

3a-5-1. In the *VIO.vho* file, copy the text between the *INST_TAG*s.

3a-5-2. In the *exampleCoreGen.vhd* file, paste the instantiation just after the instantiations of the ICON and ILA cores.

3a-5-3. Rename the instantiation to *exampleVIO*.

3a-6. Connect the ICON, ILA, and VIO cores.

3a-6-1. Rename the ICON core's CONTROL0 signal and the ILA core's CONTROL signal to *ILA_control*.

3a-6-2. Rename the ICON core's CONTROL1 signal to *VIO_control*. Rename the VIO core's CONTROL signal to *VIO_control*.

These control signals must be defined as a 36-bit *std_logic_vectors*.

3a-6-3. Immediately following the component definitions, add the following statements:

```

signal ILA_control : std_logic_vector(35 downto 0) := (others=>'0');
signal VIO_control : std_logic_vector(35 downto 0) := (others=>'0');

```

3a-7. Connect the sample clock to the ILA and VIO cores.

3a-7-1. Associate clk50 with CLK in both the VIO and ILA cores.

3a-8. Connect the ILA to examine the LEDs and the status of the reset.

Because the ILA core is a single input for the trigger, the reset and LEDs must be combined into a single signal.

3a-8-1. Add the following line of code with the other ChipScope Pro tool signal definitions to create a 9-bit signal for this purpose.

```
Signal sample_sigs: std_logic_vector(8 downto 0) := (others=>'U');
```

3a-8-2. Connect this new signal to the ILA core.

Now all that needs to be done is to assign this new signal the values of the LEDs and reset.

3a-8-3. Add the following line of code just below the ILA core.

```
sample_sigs <= rst & led;
```

3a-9. Connect the VIO to monitor the same signals as the ILA core.

3a-9-1. Connect sample_sigs to the sync_in port of the VIO.

3a-10. Define the output of the VIO core (SYNC_OUT).

3a-10-1. Add the following line of code to the group of signals for the ChipScope Pro tool immediately following the ChipScope Pro tool core definitions.

```
signal SYNC_OUT : std_logic_vector (1 downto 0) := (others=>'U');
```

3a-11. Connect the outputs of the VIO core to control the off-chip enable and the off-chip reset.

3a-11-1. Comment out the following line.

```
rst <= pb;
```

3a-11-2. Uncomment the following line.

```
rst <= pb when (sync_out(0) = '1') else sync_out(1);
```

3a-11-3. Save *exampleCoreGen.vhd*.

3a-12. Implement the design and generate the bitstream.

3a-12-1. In the Hierarchy window, select *exampleCoreGen.vhd*.

3a-12-1. In the Processes window, double-click **Generate Programming File**.

3a-12-1. Proceed to Step 4.

The next major step (Step 3b) is for Verilog users only.

Adding the Cores (Verilog)

Step 3b

3b-1. Now that the cores have been built, they must be instantiated in the design.

Create the instantiation templates for the ICON, ILA, and VIO cores².

3b-1-1. In the Hierarchy window, select *Controller*.

3b-1-2. In the Processes window, expand **CORE Generator** and double-click **View HDL Instantiation Template**.

This opens the instantiation template for the ICON core in the editor window.

3b-1-3. In the Hierarchy window, select *ILA*.

3b-1-4. In the Processes window, double-click **View HDL Instantiation Template**.

This opens the instantiation template for the ILA core in the editor window.

3b-1-5. In the Hierarchy window, select *VIO*.

3b-1-6. In the Processes window, double-click **View HDL Instantiation Template**.

This opens the instantiation template for the VIO core in the editor window.

3b-2. Instantiate the Controller (ICON) core.

3b-2-1. In the *Controller.vho* file, copy the text between the *INST_TAG*s (Figure 2).

```
//----- Begin Cut here for INSTANTIATION Template ---// INST_TAG
Controller YourInstanceName (
    .CONTROL0(CONTROL0), // INOUT BUS [35:0]
    .CONTROL1(CONTROL1) // INOUT BUS [35:0]
);

// INST_TAG_END ----- End INSTANTIATION Template -----
```

Figure 2. Locating and Capturing the ICON Instance (Verilog)

3b-2-2. In the *exampleCoreGen.v* file, move to just after the comment that indicates the ChipScope Pro tool instantiations (near line 81).

Technically, these cores can be instantiated anywhere between the modules' port list and *endmodule* statements is fine, but for uniformity, please place the core at the specified location.

3b-2-3. Rename the instantiation to *exampleICON*.

3b-3. Instantiate the ILA core.

3b-3-1. In the *ILA.veo* file, copy the text of the instantiation between the *INST_TAG*s.

3b-3-2. In the *exampleCoreGen* file, paste the instantiation just after the instantiation of the ICON core (near line 139).

3b-3-3. Rename the instantiation to *exampleILA*.

² The language that the template will be created in is selected in Design Properties under Preferred Language.

3b-4. Instantiate the VIO core.

3b-4-1. In the *VIO.veo* file, copy the text between the *INST_TAGs*.

3b-4-2. In the *exampleCoreGen.v* file, paste the instantiation just after the instantiations of the ICON and ILA cores.

3b-4-3. Rename the instantiation to *exampleVIO*.

3b-5. Connect the ICON, ILA, and VIO cores.

3b-5-1. Rename the ICON core's CONTROL0 signal and the ILA core's CONTROL signal to *ILA_control*.

3b-5-2. Rename the ICON core's CONTROL1 signal to *VIO_control*. Rename the VIO core's CONTROL signal to *VIO_control*.

These control signals must be defined as a 36-bit wires.

3b-5-3. Immediately prior to the component instantiations, add the following statements:

```
wire [35:0] ILA_control;
wire[35:0] VIO_control;
```

3b-6. Connect the sample clock to the ILA and VIO cores.

3b-6-1. Associate clk50 with CLK in both the VIO and ILA cores.

3b-7. Connect the ILA to examine the LEDs and the status of the reset.

Because the ILA core is a single input for the trigger, the reset and LEDs must be combined into a single signal.

3b-7-1. Add the following line of code with the other ChipScope Pro tool signal definitions to create a 9-bit signal for this purpose.

```
wire [8:0] sample_sigs;
```

3b-7-2. Connect this new signal to the ILA core

Now all that needs to be done is to assign this new signal the values of the LEDs and reset.

3b-7-3. Add the following line of code just below the ILA core.

```
assign sample_sigs = {rst,led};
```

3b-8. Connect the VIO to monitor the same signals as the ILA core.

3b-8-1. Connect sample_sigs to the sync_in port of the VIO.

3b-9. Define the output of the VIO core (SYNC_OUT).

3b-9-1. Add the following line of code to the group of signals for the ChipScope Pro tool immediately following the ChipScope Pro core tool definitions.

```
wire [1:0] sync_out;
```

3b-10. Connect the outputs of the VIO core to control the off-chip enable and the off-chip reset.

3b-10-1. Comment out the following line.

```
assign rst = pb;
```

3b-10-2. Uncomment the following line.

```
assign rst = sync_out[0] ? pb : sync_out[1];
```

3b-10-3. Save *exampleCoreGen.v*.

3b-11. Implement the design and generate the bitstream.

3b-11-1. In the Hierarchy window, select *exampleCoreGen.v*.

3b-11-2. In the Processes window, double-click **Generate Programming File**.

Configuring and Controlling the Design

Step 4

4-1. Configure the FPGA.

4-1-1. In the Processes window, double-click **Analyze Design Using ChipScope**.

4-1-2. Select **JTAG Chain** → **Xilinx USB Cable** or click the Initialize JTAG Chain button.

4-1-3. In the dialog box that opens, note the USB-JTAG speed and port number, and click **OK**.

This is one of many ways to initialize the JTAG chain.

4-1-4. Click **OK**.

4-1-5. In the New Project pane, right-click **XC3S1600E** and select **Configure**.

4-1-6. Click **Select New File** from the dialog box that opens.

- **VHDL users:** Browse to *C:\training\chipscope_pro\coreGenFlow-VHDL*
- **Verilog users:** Browse to *C:\training\chipscope_pro\coreGenFlow-Verilog*

4-1-7. Select *exampleCoreGen.bit* and click **Open**. Click **OK**.

After successful configuration, observe normal operation of the LEDs on the board.

Observe that the ChipScope Pro Analyzer tool has found two cores, Unit0: (ILA) and Unit1: (VIO), and that the waveform display is populated with the defined trigger inputs.

4-2. Set up the VIO Console with LEDs for the output signals and buttons for the inputs.

Note: The signals are the default names. You must customize the CDC file to import your signal names. This task is out of the scope of this lab.

4-2-1. In the New Project pane, click **DEV**, click **UNIT:1 MyVIO1 (VIO)**, and double-click **VIO Console**.

Notice the flickering arrows within the value field associated with the blue input signals—these are the LEDs that are operating.

Also observe that the green signals represent outputs and the blue signals represent inputs. Familiarize yourself with the different controls available in the horizontal toolbar, such as JTAG Scan Rate, which is equivalent to the Sampling Period.

4-2-2. In the VIO Console window, right-click **SyncIn[0]** and select **Type** → **LED** → **RED** → **High**.

Notice that the value now displays a red LED when active.

4-2-3. Right-click **SyncIn[0]** and select **Rename**. Enter **LED[0]** and click **OK**.

4-2-4. Repeat detailed steps 4-2-2 and 4-2-3 for SyncIn[1-7].

4-2-5. Observe that SyncIn[8] is tied to reset.

4-2-6. Right-click **SyncIn[8]** and select **Rename**. Enter **RST** and click **OK**.

This creates a green LED which is on when LOW for this signal.

4-2-7. Right-click **SyncOut[0]** and select **Rename**. Enter **VIO/BoardSelect** and click **OK**.

4-2-8. Right-click **VIO/BoardSelect** and select **Type** → **Toggle Button**.

4-2-9. Right-click **SyncOut[1]** and select **Rename**. Enter **VIO/RST** and click **OK**.

4-2-10. Right-click **VIO/RST** and select **Type** → **Pushbutton** → **High**.

4-3. Interact with the cores that you have placed in the design.

4-3-1. On the Spartan-3 FPGA board, press the **User Reset** button (the east pushbutton).

Question 1

Did anything occur when you pushed the User Reset button?

4-3-2. In the ChipScope Pro Analyzer tool GUI, click the **VIO/BoardSelect** button in the VIO Console to set it to 1.

4-3-3. On the Spartan-3 FPGA board, press the User Reset button again.

Question 2

Did anything occur when you pushed the User Reset button? What caused this to happen?

4-3-4. Click the **VIO/BoardSelect** button in the VIO Console to change it back to 0.

4-3-5. Click the **VIO/RST** button in the VIO console.

Question 3

Did anything occur when you pushed the VIO\RST button changing it back to 0 in the VIO Console? Is this what you expected?

Question 4

What are the advantages of using the ChipScope Pro software Core Inserter (the flow used in Lab 1)? What are the disadvantages?

Question 5

What are the advantages of using ChipScope Pro software Core Generator and instantiating it in the HDL (the flow used in this lab)? What are the disadvantages?

4-4. Expand Unit:0 and open the Trigger Setup window. Use Trigger Immediate to capture data.

4-4-1. Expand **UNIT:0 MyILA0 (ILA)** and double-click the **T!** button to cause the ILA to immediately capture and upload data.

4-4-2. In the Project window, double-click **Waveform** and examine the data.

Question 6

What are some differences between the waveform view and the VIO console?

Additional Exercises (Optional)

Step 5

5-1. The ChipScope Pro Analyzer tool is more than just blinking LEDs and a waveform. There are more ways to look at data.

- In the Sync Input Port of the VIO, collect all of the LED signals and create a bus.
- Right-click the new bus and add it to the VIO view.
- Right-click the new bus again and select Type → text field. Notice that the series of LEDs has been replaced by a text counter.
- Explore! Look at the different types of buttons, switches, and other display mechanisms provided by the VIO analyzer.

Conclusion

In this lab, you added ChipScope Pro software cores directly to the HDL code rather than using the Core Inserter to insert the core directly into the netlist. Both methods offer distinct advantages.

Inserting the cores directly into the HDL allows you to have access to all the internal HDL signals. However, inserting it into the HDL requires you to remove these cores from the source code later.

Inserting the cores into the netlist leaves the HDL untouched. However, some signals may be optimized by the synthesis tool in such a way that it is no longer available in the netlist.

Answers

1. Did anything occur when you pushed the User Reset button?

No.

2. Did anything occur when you pushed the User Reset button? What caused this to happen?

Yes, the count reset due to the change made in the code that connected the VIO sync_out(0) button to the internal reset.

3. Did anything occur when you pushed the VIO\RST button changing it back to 0 in the VIO Console? Is this what you expected?

The board reset. Yes, this was expected; again based on the change to the reset in the code that connected the VIO sync_out(1) signal.

4. What are the advantages of using the ChipScope Pro software Core Inserter (the flow used in Lab 1)? What are the disadvantages?

Inserting the cores into the netlist leaves the HDL untouched. However, some signals may be optimized by the synthesis tool in such a way that it is no longer available in the netlist.

5. What are the advantages of using ChipScope Pro software Core Generator and instantiating it in the HDL (the flow used in this lab)? What are the disadvantages?

Inserting the cores directly into the HDL allows you to have access to all the internal HDL signals. However, inserting it into the HDL requires you to remove these cores from the source code later. Certain cores, such as the VIO core, are not available in the inserter flow as minor code changes are usually required for VIO output.

6. What are some differences between the waveform view and the VIO console?

The waveform window shows static data. Once the ILA is triggered and the data is stored and uploaded to the PC, it remains fixed for examination or storage to a text file. It will only retrigger when the user commands it to.

In contrast, the VIO console shows dynamic data that is retriggered based on the JTAG scan rate.

VHDL Source Code

```

-----
-- Company:
-- Engineer:
--
-- Create Date: 09:38:30 12/18/2008
-- Design Name:
-- Module Name: UART_project - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-----
--
-- Disclaimer: LIMITED WARRANTY AND DISCLAIMER. These designs are
-- provided to you "as is". Xilinx and its licensors make, and you
-- receive no warranties or conditions, express, implied,
-- statutory or otherwise, and Xilinx specifically disclaims any
-- implied warranties of merchantability, non-infringement, or
-- fitness for a particular purpose. Xilinx does not warrant that
-- the functions contained in these designs will meet your
-- requirements, or that the operation of these designs will be
-- uninterrupted or error free, or that defects in the Designs
-- will be corrected. Furthermore, Xilinx does not warrant or
-- make any representations regarding use or the results of the
-- use of the designs in terms of correctness, accuracy,
-- reliability, or otherwise.
--
--
-- LIMITATION OF LIABILITY. In no event will Xilinx or its
-- licensors be liable for any loss of data, lost profits, cost
-- or procurement of substitute goods or services, or for any
-- special, incidental, consequential, or indirect damages
-- arising from the use or operation of the designs or
-- accompanying documentation, however caused and on any theory
-- of liability. This limitation will apply even if Xilinx
-- has been advised of the possibility of such damage. This
-- limitation shall apply notwithstanding the failure of the
-- essential purpose of any limited remedies herein.
--
--
-- Copyright © 2002, 2008 Xilinx, Inc.
-- All rights reserved
--
-----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

library UNISIM;
use UNISIM.VComponents.all;

```

```

entity uart_led is
  Generic (CLOCK_RATE : integer := 27_000_000;
          BAUD_RATE : integer := 115_200
          );
  Port ( clk_pin      : in STD_LOGIC;
        rst_pin      : in STD_LOGIC;
        btn_pin      : in STD_LOGIC;
        sel_pin       : in STD_LOGIC;
        rxd_pin       : in STD_LOGIC;
        led_pins      : out STD_LOGIC_VECTOR (7 downto 0)
        );
end uart_led;

architecture Behavioral of uart_led is

  --
  -- module definitions
  --

  component CSP_ICON
    PORT (
      CONTROL0 : INOUT STD_LOGIC_VECTOR(35 DOWNT0 0));
  end component;

  component VIO_LED_BUTT
    PORT (
      CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNT0 0);
      CLK : IN STD_LOGIC;
      ASYNC_OUT : OUT STD_LOGIC_VECTOR(0 TO 0);
      SYNC_IN : IN STD_LOGIC_VECTOR(7 DOWNT0 0));

  end component;

  -- Synplicity black box declaration (cleaned up for this lab)
  attribute syn_black_box : boolean;
  attribute syn_black_box of CSP_ICON: component is true;
  attribute syn_black_box of VIO_LED_BUTT: component is true;

  component uart_rx is
    generic (
      BAUD_RATE      : integer := 57600;          -- serves as clock divisor
      CLOCK_RATE     : integer := 100000000      -- freq of clk
    );
    Port ( rst_clk_rx      : in STD_LOGIC;
          clk_rx          : in STD_LOGIC;
          rxd_i           : in STD_LOGIC;
          rx_data         : out STD_LOGIC_VECTOR (7 downto 0);
          rx_data_rdy     : out STD_LOGIC;
          frm_err         : out STD_LOGIC
          );
  end component uart_rx;

  component reset_bridge
    Port(clk_dst          : IN std_logic;
         rst_in           : IN std_logic;
         rst_out          : OUT std_logic
    );

```

```

    );
end component reset_bridge;

component meta_harden is
  Port ( clk_dst      : in STD_LOGIC;
        rst_dst      : in STD_LOGIC;
        signal_src   : in STD_LOGIC;
        signal_dst   : out STD_LOGIC);
end component meta_harden;

component LED_manager
  Port(
    Channel_1_data      : IN std_logic_vector(7 downto 0);
    Channel_2_data      : IN std_logic_vector(7 downto 0);
    Channel_1_enable    : IN std_logic;
    Channel_2_enable    : IN std_logic;
    upperLower          : IN std_logic;
    Selector             : IN std_logic;
    clock               : IN std_logic;
    reset               : IN std_logic;
    data_out            : OUT std_logic_vector(7 downto
0)
  );
end component LED_manager;

----
--
-- naming convention
--
-- _pin                - indicates a connection to a pin of the FPGA
-- _i                  - indicates buffered version of a _pin signal
-- _<clkname>          - indicates that a signal has been synchronized to the specified clock domain
--
----

-- clock and controls
signal rst_i, rst_clk_rx : std_logic := 'U';
signal btn_i, btn_clk_rx : std_logic := 'U';
signal clk_i, clk_rx    : std_logic := 'U';
signal rxd_i            : std_logic := 'U';
signal sel_i            : std_logic := 'U';
signal led_o            : std_logic_vector(7 downto 0) := (others=>'U');
signal sel_clk_rx       : std_logic := 'U';

signal rx_data_rdy      : std_logic := 'U';
signal rx_data          : std_logic_vector(7 downto 0) := (others=>'U');

constant gnd           : std_logic_vector(7 downto 0) := (others=>'0');

signal VIO_control      : std_logic_vector(35 downto 0) := (others=>'0');
signal virtual_button   : std_logic_vector( 0 downto 0) := (others=>'0');
signal btn_combined_async : std_logic := 'U';

```

```
begin
```

```

--
-- define the buffers for the incoming data, clocks, and control
IBUF_rst_i0: IBUF port map (I=>rst_pin, O=>rst_i);
IBUF_btn_i0: IBUF port map (I=>btn_pin, O=>btn_i);
IBUF_rxd_i0: IBUF port map (I=>rx_pin, O=>rx_i);
IBUFG_clk_i0: IBUFG port map (I=>clk_pin, O=>clk_i);
BUFG_clk_rx_i0: BUFG port map (I=>clk_i, O=>clk_rx);
    IBUF_sel_i0: IBUF port map (I=>sel_pin, O=>sel_i);

--
-- define the buffers for the outgoing data
OBUF_led_ix: for i in 0 to 7 generate
    OBUF_led_i: OBUF port map (I=>LED_o(i), O=>LED_pins(i));
end generate;

--
-- instantiate a metastability hardener for the incoming data
meta_harden_rst_i0: reset_bridge port map (
    clk_dst => clk_rx,
    rst_in => rst_i,
    rst_out => rst_clk_rx
);

--
-- And the button to switch LSB and MSB
meta_harden_btn_i0: meta_harden port map (rst_dst=>rst_clk_rx, clk_dst=>clk_rx,
signal_src=>btn_combined_async, signal_dst=>btn_clk_rx);

-- since the UART will be tested separately from the application module, the meta hardener has been moved to the
uart_rx
--meta_harden_rxd_i0: meta_harden port map (rst_dst=>rst_clk_rx, clk_dst=>clk_rx, signal_src=>rx_i,
signal_dst=>rx_clk_rx);

--
-- instantiate a metastability hardener for the incoming data
meta_harden_sel_i0: meta_harden port map (rst_dst=>gnd(0), clk_dst=>clk_rx, signal_src=>sel_i,
signal_dst=>sel_clk_rx);

--
-- instantiate the receiver side of the UART
uart_rx_i0: uart_rx
generic map (
    BAUD_RATE => BAUD_RATE, -- serves as clock divisor
    CLOCK_RATE => CLOCK_RATE -- freq of clk
)
Port map (
    rst_clk_rx => rst_clk_rx,
    clk_rx => clk_rx,
    rxd_i => rxd_i,
    rx_data => rx_data,
    rx_data_rdy => rx_data_rdy,
    frm_err => open -- this signal not used in this design
);

--
-- instantiate the LED controller
LEDman: LED_manager PORT MAP(

```

```

Channel_1_data      => rx_data,
Channel_2_data      => gnd,
Channel_1_enable    => rx_data_rdy,
Channel_2_enable    => gnd(0),
upperLower         => btn_clk_rx,
Selector            => sel_clk_rx,
clock               => clk_rx,
reset               => rst_clk_rx,
data_out           => LED_o
);

--
-- ICON core
designICON : CSP_ICON port map (CONTROL0 => VIO_control);

--
-- ILA core
LEDBUTT: VIO_LED_BUTT
  port map (CONTROL => VIO_control,
            CLK      => clk_rx,
            ASYNC_OUT => virtual_button,
            SYNC_IN  => rx_data);

-- combine the asynchronous external physical button with the asynchronous VIO virtual button
btn_combined_async <= virtual_button(0) OR btn_i;

end Behavioral;
```

Verilog Source Code

```

////////////////////////////////////
//
// Xilinx, Inc. 2004          www.xilinx.com
//
////////////////////////////////////
//
// File name :    exampleInserter.v
//
//
// Date - revision : 4th February 2004 - v 1.0
//
// Author :      NJS
//
//
//
// Contact : e-mail hotline@xilinx.com
//           phone +1 800 255 7778
//
// Disclaimer: LIMITED WARRANTY AND DISCLAIMER. These designs are
//             provided to you "as is". Xilinx and its licensors make, and you
//             receive no warranties or conditions, express, implied,
//             statutory or otherwise, and Xilinx specifically disclaims any
//             implied warranties of merchantability, non-infringement, or
//             fitness for a particular purpose. Xilinx does not warrant that
//             the functions contained in these designs will meet your
//             requirements, or that the operation of these designs will be
//             uninterrupted or error free, or that defects in the Designs
//             will be corrected. Furthermore, Xilinx does not warrant or
//             make any representations regarding use or the results of the
//             use of the designs in terms of correctness, accuracy,
//             reliability, or otherwise.
//
//             LIMITATION OF LIABILITY. In no event will Xilinx or its
//             licensors be liable for any loss of data, lost profits, cost
//             or procurement of substitute goods or services, or for any
//             special, incidental, consequential, or indirect damages
//             arising from the use or operation of the designs or
//             accompanying documentation, however caused and on any theory
//             of liability. This limitation will apply even if Xilinx
//             has been advised of the possibility of such damage. This
//             limitation shall apply notwithstanding the failure of the
//             essential purpose of any limited remedies herein.
//
// Copyright © 2002, 2008 Xilinx, Inc.
// All rights reserved
//
////////////////////////////////////
//
//
// Paste ChipScope Pro Core Modules here
//

module exampleCoreGen(input clk50in,
                                input pb_in,
                                output [7:0] led_out);

```

```

// ChipScope signals for the Core Gen Flow portion of the lab
wire [35:0]      iconToILA;
wire [35:0]      iconToVIO;

// clock signals
wire            clk50;
wire            userReset;

// inter-connect signals
wire            rst;
wire            clk50int;
wire            pb;
wire [3:0]      sw;
reg [7:0]       led;

reg [5:0]       mhertz_count;
reg [9:0]       khertz_count;
reg [9:0]       hertz_count;
reg             mhertz_en;
reg             khertz_en;
reg             hertz_en;

wire [35:0]     ILA_control;
wire [35:0]     VIO_control;
wire [ 8:0]     sample_sigs;
wire [ 1:0]     sync_out;

//
// Paste ChipScope Pro Core instances here
Controller exampleICON (
    .CONTROL0(ILA_control), // INOUT BUS [35:0]
    .CONTROL1(VIO_control) // INOUT BUS [35:0]
);

ILA exampleILA (
    .CONTROL(ILA_control), // INOUT BUS [35:0]
    .CLK(clk50),           // IN
    .TRIG0(sample_sigs) // IN BUS [8:0]
);
assign sample_sigs = {rst,led};

VIO exampleVIO (
    .CONTROL(VIO_control), // INOUT BUS [35:0]
    .CLK(clk50),           // IN
    .SYNC_IN(sample_sigs), // IN BUS [8:0]
    .SYNC_OUT(SYNC_OUT)   // OUT BUS [1:0]
);

// reset      (Inserter Flow)
//assign rst = pb;
// the push button is fed directly into the FPGA internal reset

// reset (CoreGen Flow)
assign rst = sync_out[0] ? pb : sync_out[1];

```

```
// reset (CoreGen Flow) Alternate coding
// assign rst = pb | sync_out[1];
// either the push button switch or the VIO reset can cause a reset
```

```
assign userReset = rst;
```

```
// clock stuff
IBUFG_LVCMOS33 clk50in_ibufg
(.I(clk50in),
.O(clk50int));
```

```
BUFG rxclka_bufg
(.I(clk50int),
.O(clk50));
```

```
IBUF_LVCMOS33 pb_ibufg
(.I(pb_in),
.O(pb));
```

```
// led and slide switch buffers
generate
  genvar k;
  for (k = 0; k <= 7 ; k = k + 1 )
  begin: loop1 // label required, controls instance naming
    OBUF_LVCMOS33 led_obuf
      (.I(led[k]),
      .O(led_out[k]));
  end
endgenerate
```

```
// generates a 1 Mhz signal from a 50 Mhz signal
always @ (posedge clk50, posedge userReset)
begin
  if (userReset)
  begin
    mhertz_count <= 0;
    mhertz_en <= 0;
  end
  else begin
    // default
    mhertz_en <= 0;
    mhertz_count <= mhertz_count + 1;
    if (mhertz_count == 6'b110010)
    begin
      mhertz_en <= 1;
      mhertz_count <= 0;
    end
  end // else: !if(userReset)
end // always @ (posedge clk50, posedge userReset)
```

```
// generates a 1 kHz signal from a 1Mhz signal
always @ (posedge clk50, posedge userReset)
begin
  if (userReset)
  begin
```

```

        khertz_count <= 0;
        khertz_en <= 0;
    end
    else begin
        //default
        khertz_en <= 0;

        if (mhertz_en)
            khertz_count <= khertz_count + 1 ;

        if (khertz_count == 10'b1111101000)
            begin
                khertz_en <= 1;
                khertz_count <= 0;
            end
        end // else: !if(userReset)
    end // always @ (posedge clk50, posedge userReset)

//generates a 1 Hz signal from a 1 kHz signal
always @ (posedge clk50, posedge userReset)
begin
    if (userReset)
        begin
            hertz_count <= 0;
            hertz_en <= 0;
        end
    else begin
        //default
        hertz_en <= 0;
        if (khertz_en)
            hertz_count <= hertz_count + 1;

        if (hertz_count == 10'b1111101000)
            begin
                hertz_en <= 1;
                hertz_count <= 0;
            end
        end // else: !if(userReset)
    end // always @ (posedge clk50, posedge userReset)

// moved the led counter every second.
always @ (posedge clk50, posedge userReset)
begin
    if (userReset)
        led <= 8'b00000001;
    else if (hertz_en)
        led <= {led[6:0], led[7]};
    end // always @ (posedge clk50, posedge userReset)

endmodule // exampleInserter

```