

# Using the IP Catalog and IP Integrator

## Introduction

In this lab you will use the IP Catalog to generate a clock resource. You will instantiate the generated clock core in the provided waveform generator design. You will also use IP Integrator to generate a FIFO core and then use it in the HDL design.

## Objectives

After completing this lab, you will be able to:

- Include an IP in the project during the project creation
- Use IP Catalog to generate a clocking core
- Instantiate the generated clock
- Create a block design using IP Integrator
- Instantiate the block design
- Generate bitstream and verify the functionality in hardware

## Procedure

This lab is broken into steps that consist of general overview statements providing information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

**Note: You will notice certain procedures have different variations depending on development board being ZedBoard or Zybo. It will be explicitly mentioned in notes when such variation is encountered**

## Design Description

The design used in this lab is a programmable waveform generator, also known as a signal generator.

The waveform generator in this design is intended to be a “standalone” device that is controlled via a PC (or other terminal device) using RS-232 serial communication. The design described here implements the RS-232 communication channel, the waveform generator and connection to the external DAC, and a simple parser to implement a small number of “commands” to control the waveform generation.

The wave generator implements a look-up table of 1024 samples of 16 bits each in a RAM. The wave generator also implements three variables:

- `nsamp`: The number of samples to use for the output waveform. Must be between 1 and 1024.
- `prescale`: The prescaler for the sample clock. Must be 32 or greater.
- `speed`: The speed (or rate) for the output samples in units of the prescaled clock.

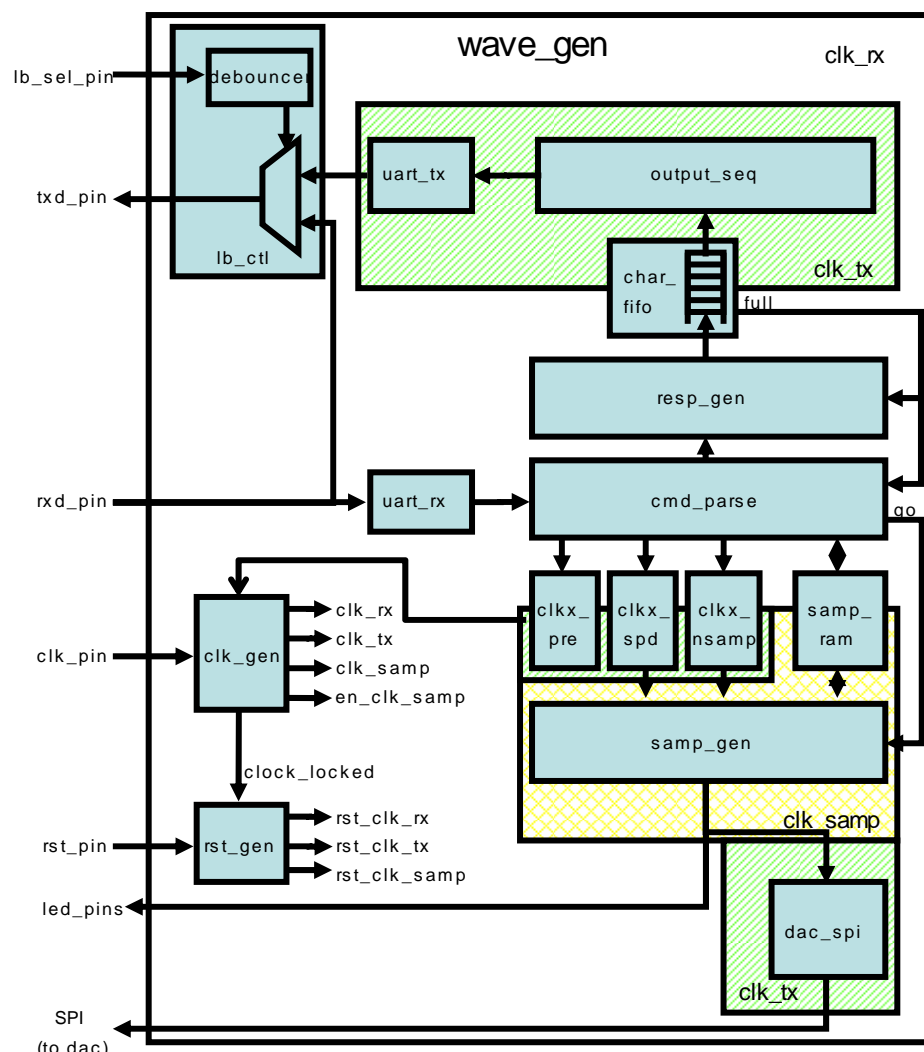
The wave generator can be instructed to send the appropriate number of samples once, cycling from 0 to `nsamp-1` once and then stopping, or continuously, where it continuously loops the `nsamp` samples. When enabled, either once or continuously, the wave generator will send one sample to the DAC every (`prescale x speed`) `clk_tx` clock cycles. The contents of the RAM, as well as the three variables, can be changed via commands sent over the RS-232 link, as can the mode of the wave generator. The wave generator will generate responses for all commands.

There are three clock domains within this design: **`clk_rx`**, **`clk_tx`**, and **`clk_samp`**. The clock generator module instantiates all the clocking resources required for generating these three clocks. All three clocks are derived from a single clock input, coming in on **`clk_pin`**. The frequency of the clock input depends on the oscillator available on the target board; for the ZedBoard it is 100MHz and for Zybo it is 125MHz.

The block diagram is as shown in **Figure 1**.

In this design we will use board's USB-UART which is controlled by the Zynq's ARM Cortex-A9 processor. Our PL design needs access to this USB-UART. So first thing we will do is to create a Processing System design which will put the USB-UART connections in a simple GPIO-style and make it available to the PL section. The complete system is shown in **Figure 2**.

The provided design places the UART (RX and TX) pins of the PS (Processing System) on the Cortex-A9 in a simple GPIO mode to allow the UART to be connected (passed through) to the Programmable Logic. The processor samples the RX signal and sends it to the EMIO channel 0 which is connected to Rx input of the HDL module provided in the Static directory. Similarly, the design samples the Tx output of the HDL module through another EMIO channel 1 and sends it on the PS UART TX pin. This is done through a software application provided in the lab4.sdk folder hierarchy. The design is shown in Figure 2.



**Figure 1. The complete design on PL**

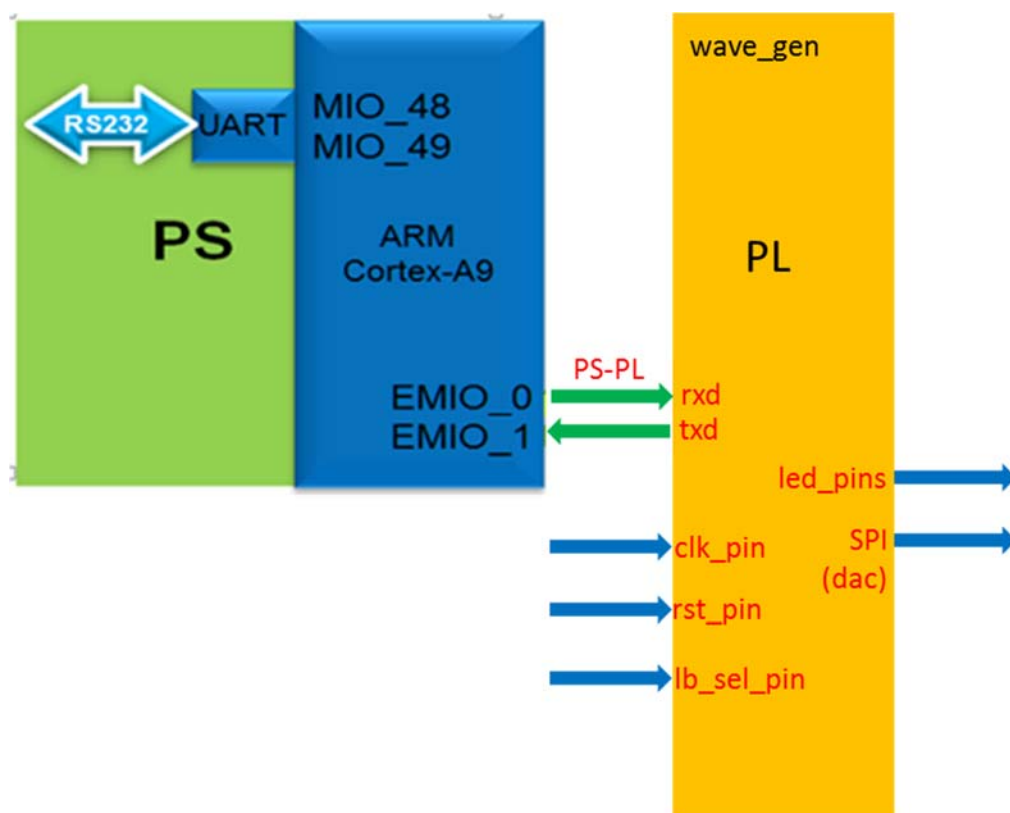
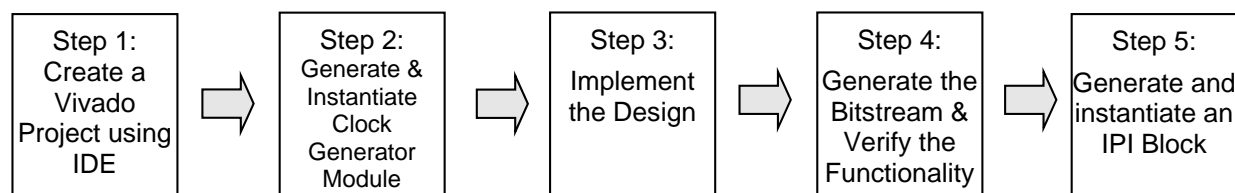


Figure 2. The Complete System

## General Flow



## Create a Vivado Project using IDE

### Step 1

In this design we will use board's USB-UART which is controlled by the Zynq's ARM Cortex-A9 processor. Our PL design needs access to this USB-UART. So first thing we will do is to create a Processing System design which will put the USB-UART connections in a simple GPIO-style and make it available to the PL section.

- 1-1. Launch Vivado and create a project targeting the XC7Z020clg484-1 device (ZedBoard), or the XC7Z010clg400-1 (Zybo), and use provided the tcl scripts (ps7\_create\_<board>.tcl) to generate the block design for the PS subsystem. Also, add the Verilog HDL files, wave\_gen\_pins\_<board>.xdc and wave\_gen\_timing.xdc files from the <2016\_2\_ZYNQ\_sources>\lab4 directory.**

References to <2016\_2\_ZYNQ\_labs> is a placeholder for the c:\xup\fpga\_flow\2016\_2\_ZYNQ\_labs directory and <2016\_2\_ZYNQ\_sources> is a placeholder for the c:\xup\fpga\_flow\2016\_2\_ZYNQ\_sources directory.

Reference to <board> means either the **ZedBoard** or the **Zybo**

- 1-1-1.** Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2016.2 > Vivado 2016.2**
- 1-1-2.** Click **Create New Project** to start the wizard. You will see *Create A New Vivado Project* dialog box. Click **Next**.
- 1-1-3.** Click the Browse button of the *Project location* field of the **New Project** form, browse to <2016\_2\_ZYNQ\_labs>, and click **Select**.
- 1-1-4.** Enter **lab4** in the *Project name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.
- 1-1-5.** Select **RTL Project** option in the *Project Type* form, and click **Next**.
- 1-1-6.** Using the drop-down buttons, select **Verilog** as the *Target Language* and *Simulator Language* in the *Add Sources* form.
- 1-1-7.** Click on the **Green Plus** button, then the **Add Files...** button and browse to the <2016\_2\_ZYNQ\_sources>\lab4 directory, select all the Verilog files, click **OK**, and then click **Next**.
- 1-1-8.** In the *Add IP* form, click on the **Green Plus** button, then the **Add Directories...** button, browse to the <2016\_2\_ZYNQ\_sources>\lab4\<board>\ip directory. Highlight the **ip** directory and click **Select**.
- 1-1-9.** Click **Next** to get to the *Add Constraints* form.
- 1-1-10.** Click on the **Green Plus** button, then **Add Files...** and browse to the c:\xup\fpga\_flow\2016\_2\_ZYNQ\_sources\lab4 directory (if necessary), select wave\_gen\_timing\_<board>.xdc and the appropriate wave\_gen\_pins\_<board>.xdc and click **Open**.
- 1-1-11.** Click **Next**.
- 1-1-12.** In the *Default Part* form, Use the **Boards** option, you may select the **Zedboard** or the **Zybo** depending on your board from the Display Name drop down field.

You may also use the **Parts** option and various drop-down fields of the **Filter** section. If using the ZedBoard, select the **XC7Z020clg484-1** part. If using the Zybo, select the **XC7Z010clg400-1** part.

**Note:** Notice that Zedboard and Zybo may not be listed under **Boards** menu as they are not in the tools database. If not listed then you can download the board files for the desired boards either from Digilent Inc website or from the XUP website's workshop material pages.

**1-1-13.** Click **Next**.

**1-1-14.** Click **Finish** to create the Vivado project.

## **1-2. Correct the errors by adding file.**

**1-2-1.** You will notice four *Syntax Error Files* are being highlighted in the **Sources** pane.

If you check the messages Tab, you will see that these errors are due to missing files.

**1-2-2.** Click on **Add Sources** in the *Flow Navigator* pane.

**1-2-3.** Select *Add or Create Design Sources* and click **Next**.

**1-2-4.** Click on the **Green Plus** button, then the **Add Files...** button and browse to **<2016\_2\_ZYNQ\_sources>\lab4\**.

**1-2-5.** In the *File Type* field, select **All Files**, and then select **clogb2.txt** file.

**1-2-6.** Click **OK** and then **Finish**.

The error messages should go away.

**1-2-7.** In the *Sources* pane, expand *Design Sources* and *wave\_gen\_top* and *wave\_gen* if necessary, and double-click on the **clk\_gen\_i0** entry.

Scroll down the file and notice that around line 79 there is an instruction to instantiate a clock core.

**1-2-8.** In the Tcl Shell window enter the following command to change to the lab directory and hit **Enter**.

```
cd c:/xup/fpga_flow/2016_2_ZYNQ_sources/lab4
```

**1-2-9.** Generate the PS design by executing the provided Tcl script.

```
source ps7_create_zed.tcl (for ZedBoard) or
```

```
source ps7_create_zybo.tcl (for Zybo)
```

This script will create a block design called *system*, instantiate ZYNQ PS two GPIO channels 48 and 49 and two EMIO channels. It will create *system.bd* that is instantiated under wrapper file called *system\_wrapper.v*. You can check the contents of the tcl files to confirm the commands that are being run.

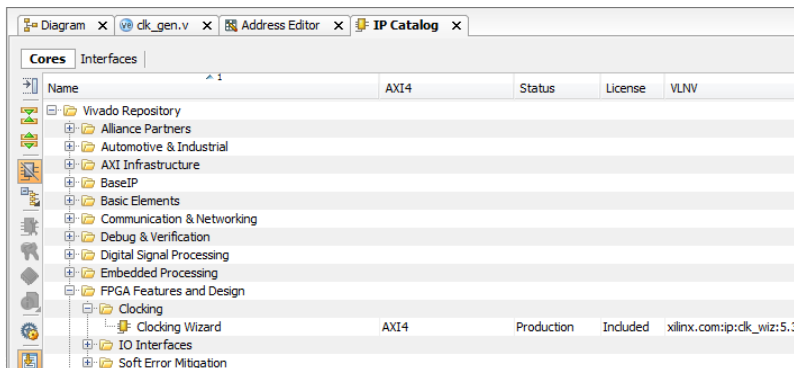
## Generate and Instantiate Clock Generator Module

## Step 2

- 2-1. Launch the clocking wizard from the IP Catalog of Vivado and generate the clock core with input frequency of 100.00 MHz and two output clocks of 50.000 MHz each.**

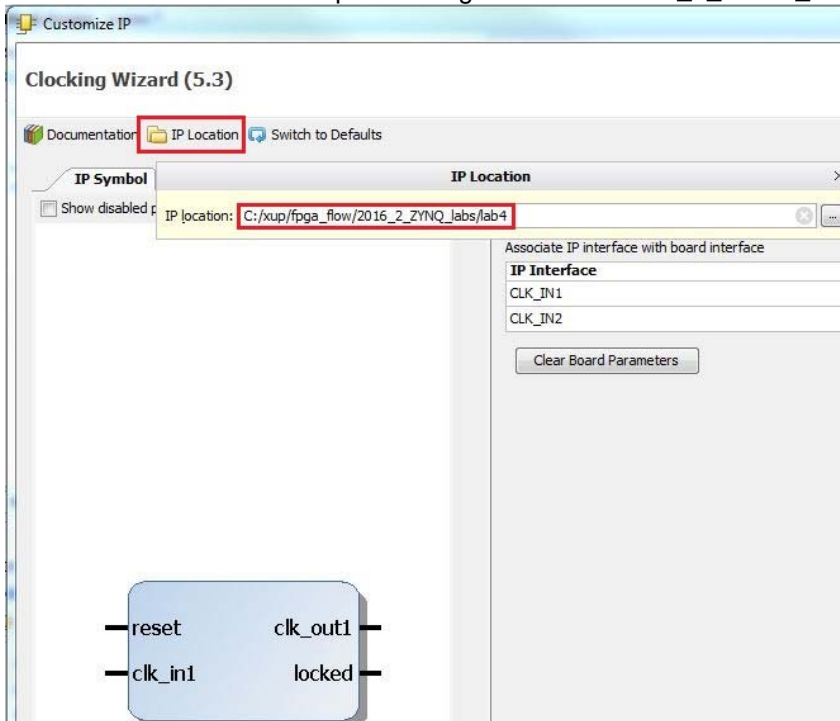
**Note:** For Zybo since the on-board clock is 125 MHz we will set the input frequency to 125 Mhz.

- 2-1-1.** Click on **IP Catalog** in the *Flow Navigator* pane. The IP Catalog will open in the auxiliary pane.
- 2-1-2.** Expand the **FPGA Features and Design > Clocking** sub-folders and double-click on the **Clocking Wizard** entry.



**Figure 3. Accessing the clocking wizard**

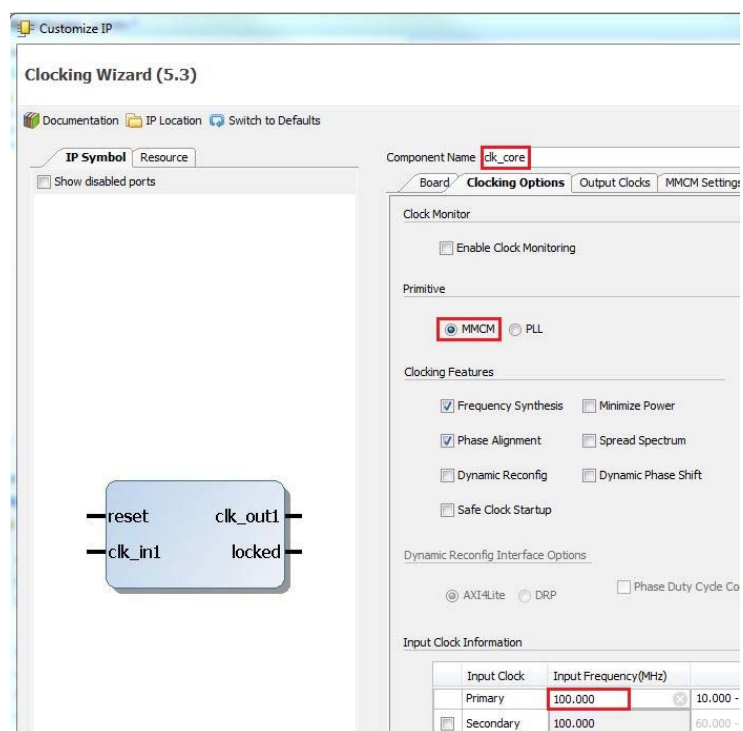
- 2-1-3.** Click **Customize IP** on the following Add IP window. The clocking wizard will open.
- 2-1-4.** Check that the **IP Location** points to right location **<2016\_2\_ZYNQ\_labs>lab4**.



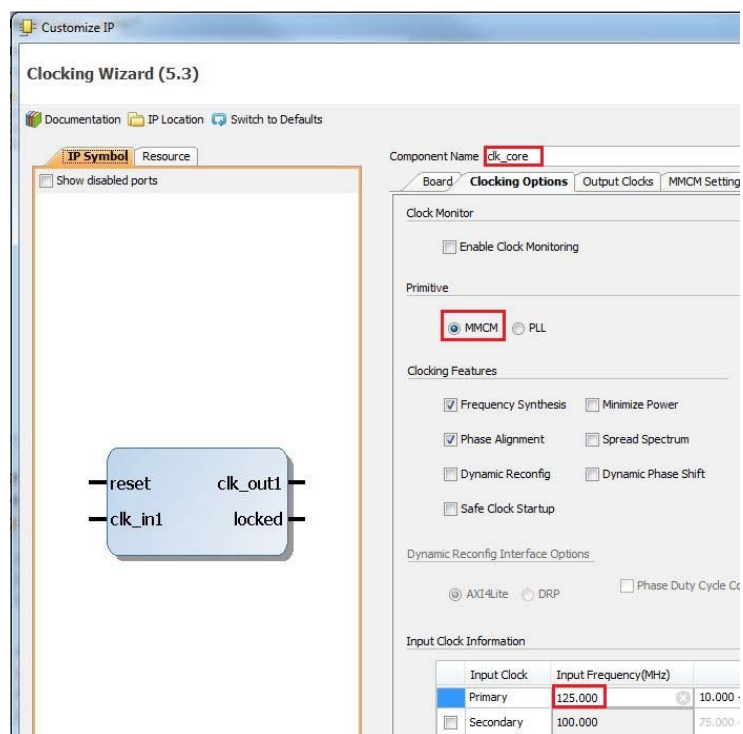
**Figure 4. Accessing the clocking wizard**

- 2-1-5.** Change the core name to **clk\_core**. Make sure that the *Primary* input clock frequency is **100.000** MHz and the primitive used is **MMCM**.

**Note:** For Zybo since the on-board clock is 125 MHz we will set the input frequency to 125 Mhz.



**Figure 5. The clocking wizard for ZedBoard**



**Figure 5. The clocking wizard for Zybo**

- 2-1-6.** Select the **Output Clocks** tab. Click on the check box to enable the second clock output. Make sure that the requested output frequency is 50 MHz for both clocks.

Component Name

Board Clocking Options **Output Clocks** MMCM Settings Port Renaming Summary

The phase is calculated relative to the active input clock.

Output Clock	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)	
	Requested	Actual	Requested	Actual	Requested	Actual
<input checked="" type="checkbox"/> clk_out1	50	50.000	0.000	0.000	50.000	50.0
<input checked="" type="checkbox"/> clk_out2	50	50.000	0.000	0.000	50.000	50.0

**Figure 6. Setting output clocks**

- 2-1-7.** Click on the **Summary** tab and check the information.

Component Name

Board Clocking Options Output Clocks MMCM Settings Port Renaming **Summary**

Attribute	Value
<b>Input Clock (MHz)</b>	100.000
Phase Shift	None
Divide Counter	1
Mult Counter	10.000
CLKOUT0 Divider	20.000
CLKOUT1 Divider	20
CLKOUT2 Divider	OFF
CLKOUT3 Divider	OFF
CLKOUT4 Divider	OFF
CLKOUT5 Divider	OFF
CLKOUT6 Divider	OFF

**Figure 7. Summary page of the clock core being generated for ZedBoard**

Component Name

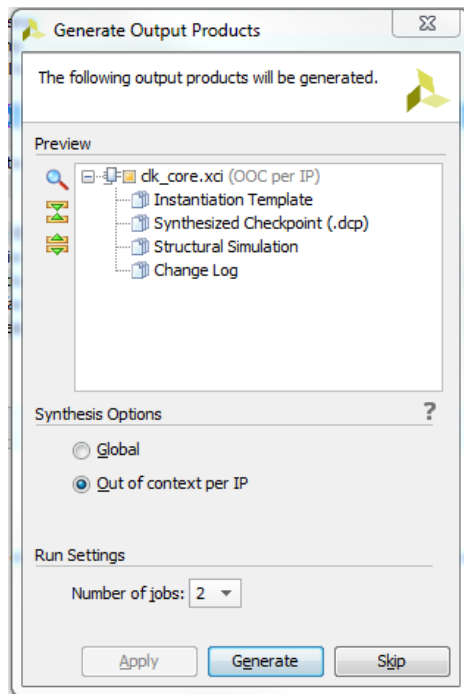
Board Clocking Options Output Clocks MMCM Settings Port Renaming **Summary**

Attribute	Value
<b>Input Clock (MHz)</b>	125.000
Phase Shift	None
Divide Counter	1
Mult Counter	8.000
CLKOUT0 Divider	20.000
CLKOUT1 Divider	20
CLKOUT2 Divider	OFF
CLKOUT3 Divider	OFF
CLKOUT4 Divider	OFF
CLKOUT5 Divider	OFF
CLKOUT6 Divider	OFF

**Figure 7. Summary page of the clock core being generated for Zybo**

- 2-1-8.** Click **OK** to see the *Generate Output Products* form.





**Figure 8. Generate output products form**

**2-1-9.** Click on **Generate** to generate the output products including the instantiation template. Click **OK** to proceed.

## **2-2. Instantiate the generated clock core.**

**2-2-1.** Select the **IP Sources** tab in the *Sources* pane.

**2-2-2.** Expand the **IP (2)** branch. Notice the two IP entries. The `char_fifo` IP is the core that was included while creating project. The second core `clk_core` is the one that you have generated.

**2-2-3.** Expand `clk_core > Instantiation Template` and double-click on `clk_core.veo` to see the instantiation template.

**2-2-4.** Copy lines 71 through 80 and paste them at or around line 79 of the `clk_gen.v` file.

**2-2-5.** Change the instance name and net names to as shown in the figure below to match the names of existing signals in the design.

```

78 // Instantiate clk_core - generated by the Clocking Wizard
79 clk_core clk_core_i0
80 (// Clock in ports
81  .clk_in1(clk_pin),           // input clk_in1
82  // Clock out ports
83  .clk_out1(clk_rx),           // output clk_out1
84  .clk_out2(clk_tx),           // output clk_out2
85  // Status and control signals
86  .reset(rst_i), // input reset
87  .locked(clock_locked));      // output locked
88

```

**Figure 9. Assigning instance name and net connections**

**2-2-6.** Select **File > Save File** to save **clk\_gen.v**

**2-2-7.** Select the Hierarchy tab and expand the **wave\_gen > clk\_gen\_i0** hierarchy and verify that **clk\_core.xci** is in the hierarchy. The IP has a bordered yellow square icon next to it.

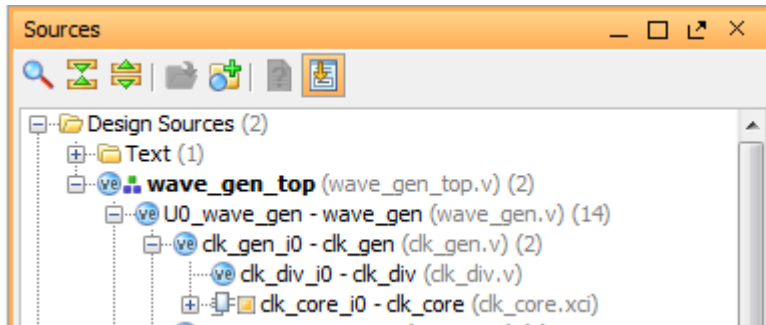


Figure 10. The **clk\_core** instantiated and shown in the hierarchy

## Implement the Design

## Step 3

### 3-1. Implement the design.

**3-1-1.** Click on the **Run Implementation** in the *Flow Navigator* pane.

**3-1-2.** Click **OK** and run the synthesis first before running the implementation process. Click **Save** to Save Project if prompted.

When the implementation is completed, a dialog box will appear with three options.

**3-1-3.** Select the *Open Implemented Design* option and click **OK**.

### 3-2. View the amount of FPGA resources consumed by the design using Report Utilization.

**3-2-1.** In the *Flow Navigator* pane, select **Implemented Design > Report Utilization**.

The Report Utilization dialog box opens.

**3-2-2.** Click **OK**.

**3-2-3.** Verify that the design is using the clock resource.

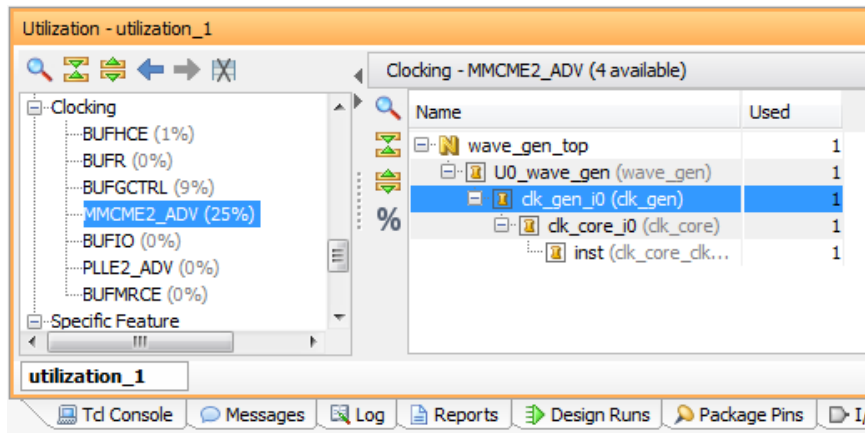


Figure 11. Clock resource utilization for the ZedBoard

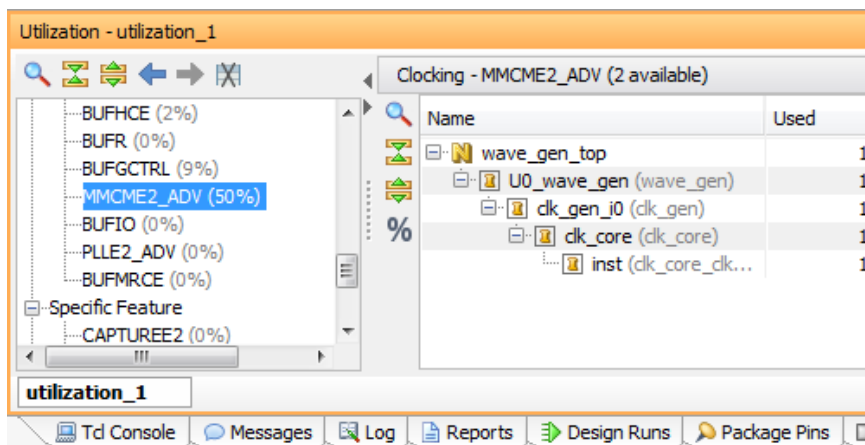


Figure 11. Clock resource utilization for the Zybo

## Generate the Bitstream and Verify the Functionality

## Step 4

### 4-1. Generate the bitstream.

4-1-1. In the Flow Navigator, under Program and Debug, click **Generate Bitstream**.

4-1-2. Click **Cancel** when the bitstream generation is completed.

### 4-2. Connect the board and power it ON. Open a hardware session, and program the FPGA.

4-2-1. Make sure that the Micro-USB cable is connected to the JTAG PROG connector next to the power supply connector for the Zedboard. The Zybo JTAG PROG connector is located next to the power supply switch).

4-2-2. Select the *Open Hardware Manager* option and click **OK**.

The Hardware Manager window will open indicating “unconnected” status.

**4-2-3.** Click on the **Open target** link, then **Auto Connect** from the dropdown menu.

You can also click on the **Open recent target** link if the board was already targeted before.

**4-2-4.** The Hardware Manager status changes from Unconnected to the server name and the device is highlighted. Also notice that the Status indicates that it is not programmed.

**4-2-5.** Select the device and verify that the **wave\_gen.bit** is selected as the programming file in the General tab.

**4-3. Start a terminal emulator program such as TeraTerm or HyperTerminal. Select an appropriate COM port (you can find the correct COM number using the Control Panel). Set the COM port for 115200 baud rate communication. Program the FPGA**

**4-3-1.** Start a terminal emulator program such as TeraTerm or HyperTerminal.

**4-3-2.** Select the appropriate COM port (you can find the correct COM number using the Control Panel).

**4-3-3.** Set the COM port for **115200** baud rate communication.

**4-3-4.** Right-click on the FPGA entry in the Hardware window and select **Program Device...**

**4-3-5.** Click on the **Program** button.

The programming bit file be downloaded and the DONE light will be turned ON indicating the FPGA has been programmed.

**4-4. Start a SDK session, point it to the c:/xup/fpga\_flow/2016\_2\_ZYNQ\_Sources/lab4/<board>/lab4.sdk workspace.**

**4-4-1.** Open **SDK** by selecting **Start > All Programs > Xilinx Design Tools > SDK 2016.2 > Xilinx SDK 2016.2**

**4-4-2.** In the **Select a workspace** window, click on the browse button, browse to c:/xup/fpga\_flow/2016\_2\_ZYNQ\_Sources/lab4/ directory and select either c:/xup/fpga\_flow/2016\_2\_ZYNQ\_Sources/lab4/Zybo/lab4.sdk or c:/xup/fpga\_flow/2016\_2\_ZYNQ\_Sources/lab4/ZedBoard/lab4.sdk and click **OK**.

**4-4-3.** Click **OK**.

In the *Project Explorer*, right-click on the wave\_gen\_uart, select *Run As*, and then **Launch on Hardware (System Debugger)**

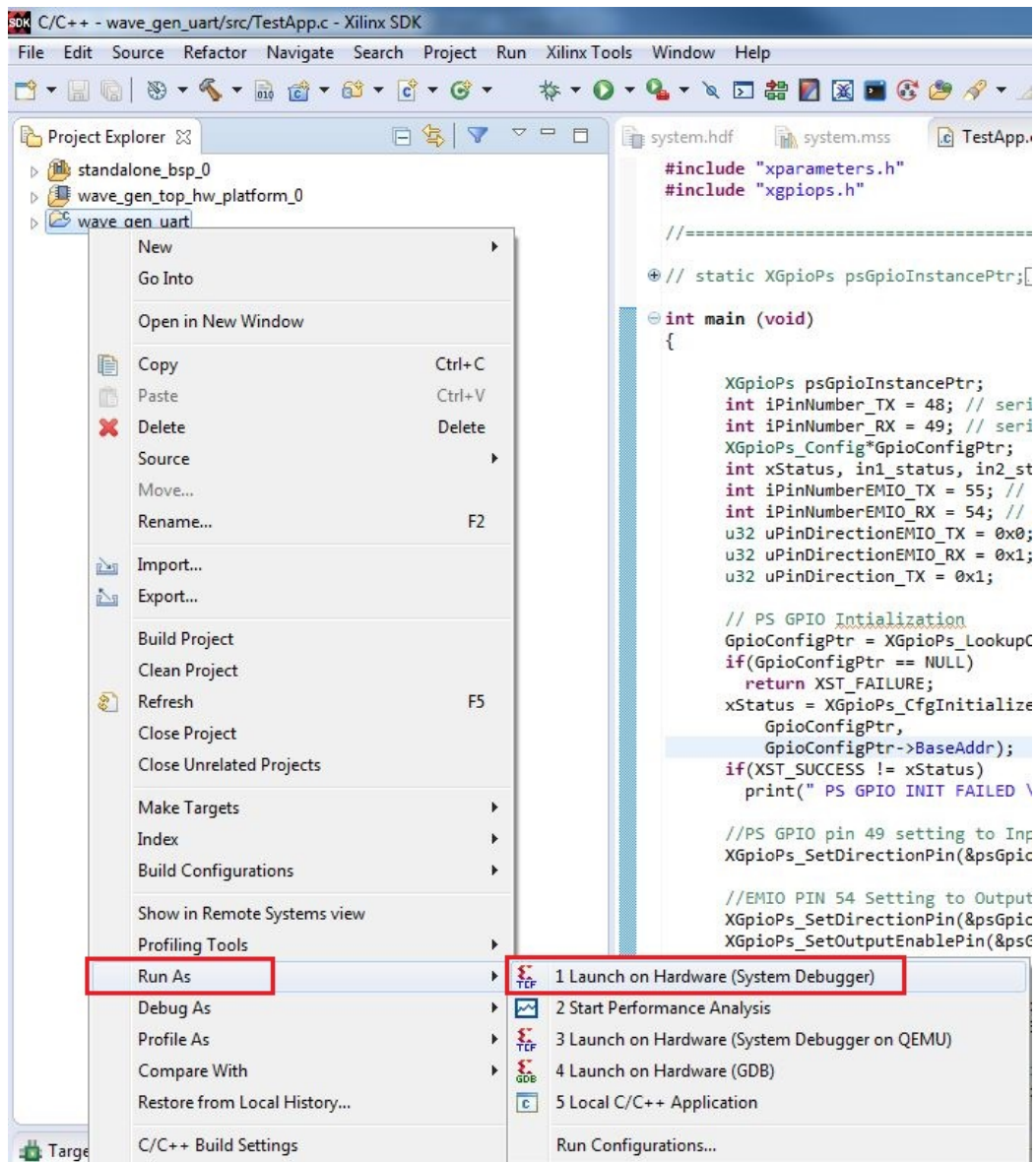


Figure 12. Running the application

- 4-4-4. Slide *Switch 0* to the **ON** position and type in some characters in the terminal window and see the character is echoed back. Setting Switch 0 to the ON position makes the design function as a loopback.
- 4-4-5. Set *Switch 0* back to **OFF** position (down) so it is no longer in the loopback mode.
- 4-4-6. Select **File > Send File ...** in the Tera Term window.
- 4-4-7. Browse to <2016\_2\_ZYNQ\_>\lab4, select **testpattern.txt** file, and click **Open**.

The file content will be send to the design. The file content is as follows:

```
*PFFFF      < -- specifies the pre-scaling
*S0fff      < -- specifies the speed value
*N000f      < -- specifies the number of samples to play
```

```

*W00000000      < -- write first sample of value 0 at location 0000
*W00011111      < -- write second sample of value 0x1111 at location 0001
*W00022222
*W00033333
*W00044444
*W00055555
*W00066666
*W00077777
*W00088888
*W00099999
*W000AAAAA
*W000BBBBB
*W000CCCCC
*W000DDDDD
*W000EEEEEE
*W000FFFFF

```

**4-4-8.** The design understands various commands as listed in figure below. All values are in hexadecimal. All values and addresses are in hexadecimal.

Cmd	Input	Response	Description
*W	aaaavvvv	-OK or -ERR	03ff ≥ aaaa ≥ 0000. Value "vvvv" is written into RAM at location "aaaa" and "-OK" is return.
*R	aaaa	-hhhh dddd or -ERR	03ff ≥ aaaa ≥ 0000. If in range, then the value at "aaaa" is returned in hex and decimal.
*N	vvvv	-OK or -ERR	0400 ≥ vvvv ≥ 0001. Specifies the number of samples before recycling.
*P	vvvv	-OK or -ERR	ffff ≥ vvvv ≥ 0020. Specifies prescaling value to divide <i>clk_tx</i> by to produce <i>clk_samp</i> .
*S	vvvv	-OK or -ERR	ffff ≥ vvvv ≥ 0001. Specifies "speed" value to divide <i>clk_samp</i> by to produce the rate of read from RAM.
*n/*p/*s		-hhhh dddd	Returns current value of nsamp, prescale, and speed.
*G		-OK	Triggers a single pass through nsamp memory locations.
*C		-OK	Starts continuous triggering.
*H		-OK	Halts continuous loop at end of current cycle.

**Figure 13. Commands**

**4-4-9.** Next type \*G in the terminal window and observe the LED pattern changing slowly as written by the above file.

**4-4-10.** You can type \*s to see the sample value, \*p to see the prescale value, and \*n to see how many samples are being played back.

**4-4-11.** You can also type \*H to halt the play.



```

*PFFFF-OKf
*S0fff-OK
*N000f-OK
*U00000000-OK
*U00011111-OK
*U00022222-OK
*U00033333-OK
*U00044444-OK
*U00055555-OK
*U00066666-OK
*U00077777-OK
*U00088888-OK
*U00099999-OK
*U000AAAAA-OK
*U000BBBBB-OK
*U000CCCCC-OK
*U000DDDDD-OK
*U000EEEEEE-OK
*U000FFFFFF-OK
*C-OK
*p-ffff 65535
*n-000f 00015
*s-0fff 04095
*C-OK
*H-OK

```

Figure 14. Terminal window display

4-4-12. Select **File > Close Hardware Manager**. Click **OK** to close it.

4-4-13. Close the **SDK** program by selecting **File > Exit** and click **OK**.

## Generate and Instantiate an IPI Block

## Step 5

5-1. Save the project as **lab4\_ipi**. Remove the **char\_fifo** IP from the design.

5-1-1. Select **File > Save Project As...** and save it as **lab4\_ipi** in the **<2016\_2\_ZYNQ\_labs>** directory making sure that the *Create Project Subdirectory* option is checked.

5-1-2. Select the **IP Sources** tab in the *Sources* pane.

5-1-3. Right-click on **char\_fifo**, and select **Remove File from Project...**

5-1-4. If a following window appears, click on the check-box of *Also delete the project local file/directory from disk*, and click **OK** or else continue to next step.

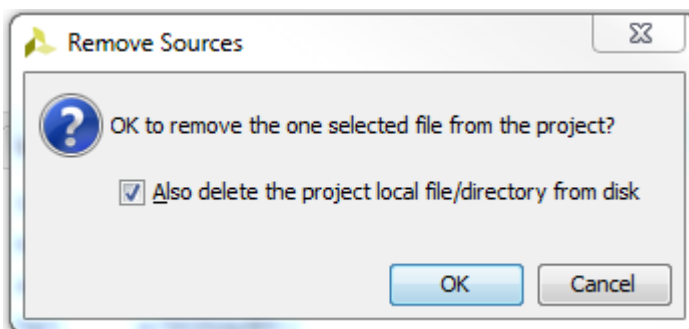


Figure 15. Removing an existing IP from the project

5-1-5. Select **Hierarchy** tab in the *Sources* pane and observe that the **char\_fifo** instance has a ? mark indicating that it is missing the source file.

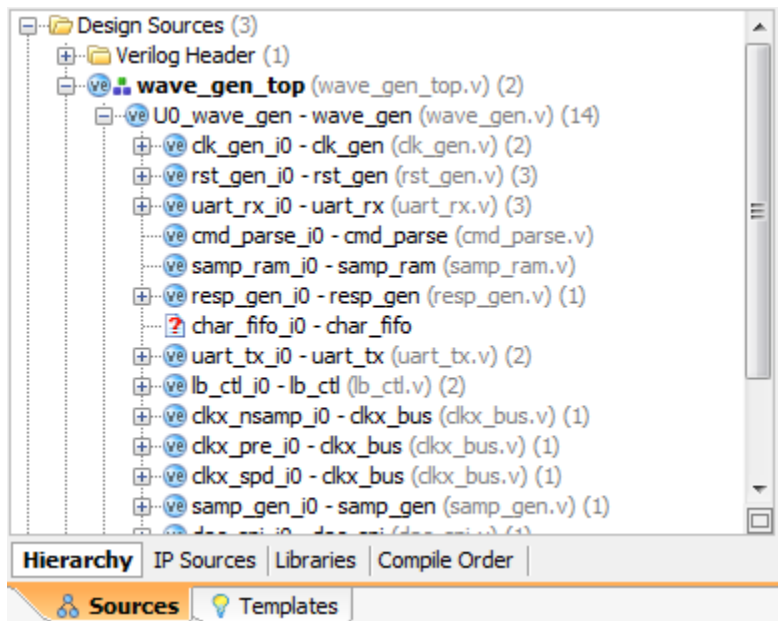


Figure 16. Removed source file

5-1-6. Double-click on the **wave\_gen.v** to open it in the editor window.

5-1-7. Remove the instantiation of the **char\_fifo** from the file around line 336.

5-1-8. Select **File > Save File**.

## 5-2. Create a block design naming it as **char\_fifo** and add an instance of an **FIFO Generator IP**.

5-2-1. Click on **Create Block Design** in the Flow Navigator block.

5-2-2. Enter **char\_fifo** as the block design name.

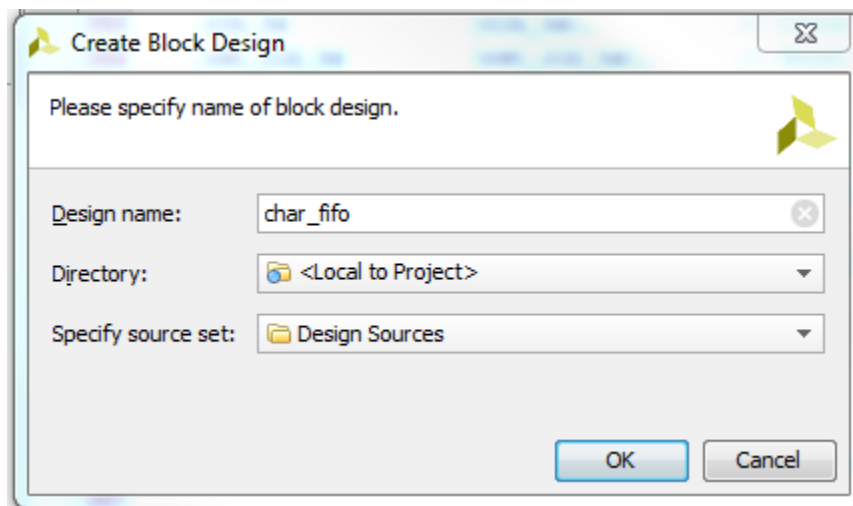


Figure 17. Naming the new block design

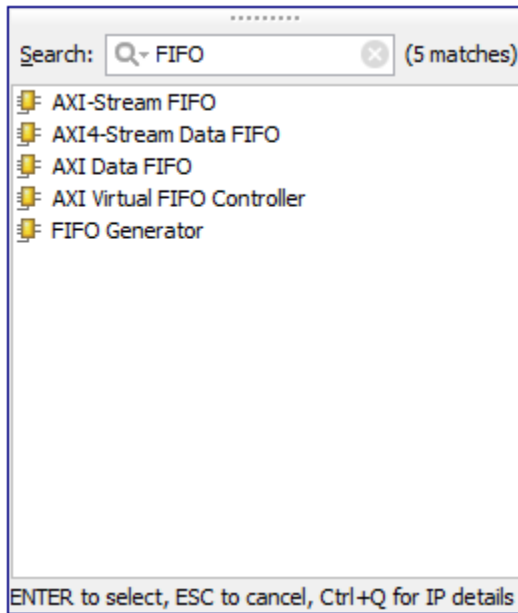


**5-2-3. Click OK.**

The IP Integrator workspace opens and, in the information area, invites you to begin adding IP.

**5-2-4. Right-click in the IP Integrator design canvas and select **Add IP**.**

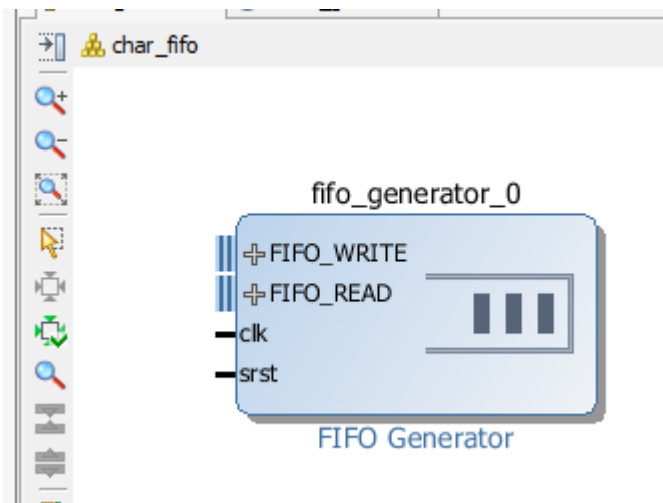
The IP Integrator IP Catalog opens, displaying a list of IP available in the IP Integrator.

**5-2-5. Type **FIFO** in the search box at the top of the IP Integrator Catalog to see FIFO related available IPs.**

**Figure 18. Searching for an IP in the IP Catalog**

**5-2-6. Double-click **FIFO Generator**.**

The FIFO is added to the IP Integrator design canvas.



**Figure 19. FIFO Generator instantiated**

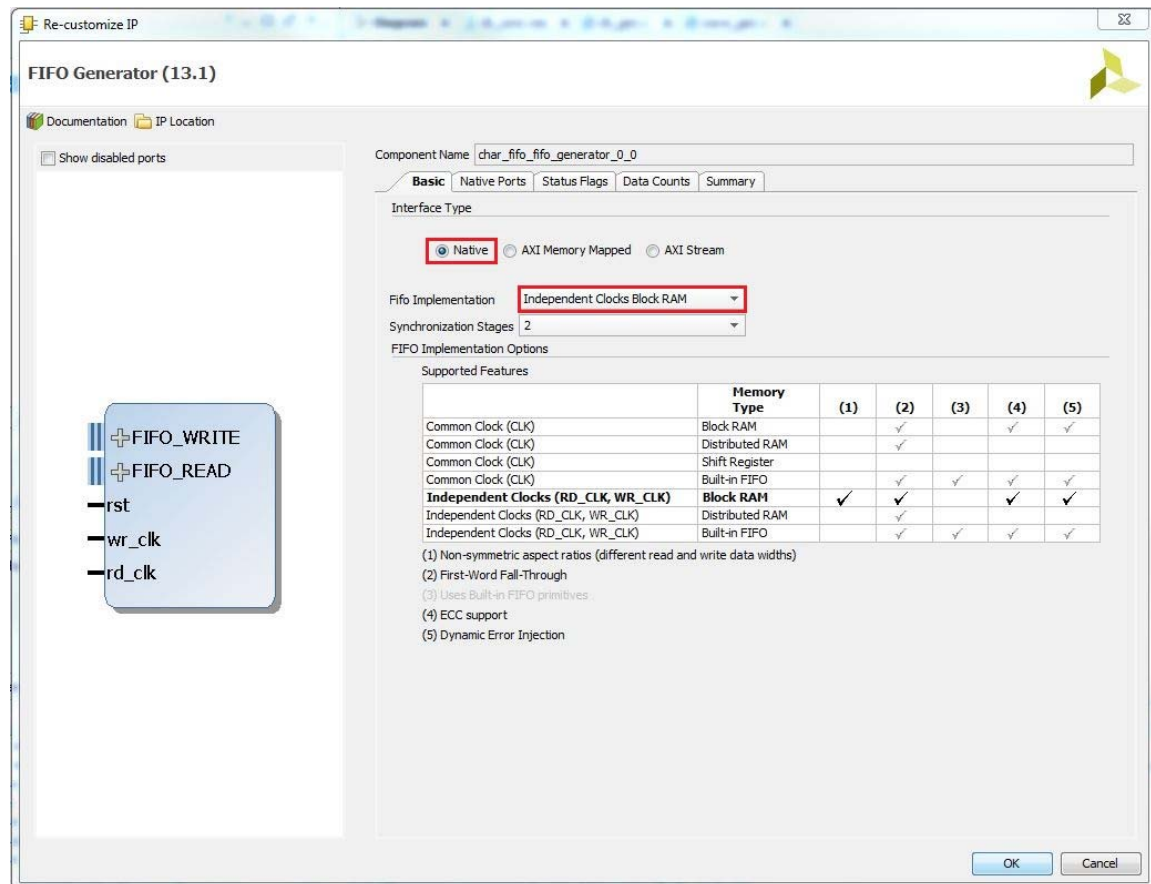
### 5-3. Customize the FIFO Generator IP instance.

#### 5-3-1. Double-click the **FIFO Generator** IP.

The FIFO Generator displays in the Re-customize IP dialog box.

#### 5-3-2. Make sure that the default **Native** option is selected for the interface type.

#### 5-3-3. Select **Independent Clocks Block RAM** from the Fifo Implementation drop-down list.



**Figure 20. Configuring BRAM for separate read and write clocks**

#### 5-3-4. Select the **Native Ports** tab.

From the *Native Ports* tab you can configure the read mode, built-in FIFO options, data port parameters, and implementation options.

#### 5-3-5. Select **First Word Fall Through** as the read mode.

#### 5-3-6. Set the write width to be **8** bits.

#### 5-3-7. Click in the *Read Width* field to change it automatically to match the write width.

#### 5-3-8. Leave everything else at their default settings.

Component Name: char\_fifo\_fifo\_generator\_0\_0

Basic Native Ports Status Flags Data Counts Summary

Read Mode

☐ Standard FIFO ☒ First Word Fall Through

Data Port Parameters

Write Width: 8 (1, 2, 3, ..., 1024)

Write Depth: 1024 (Actual Write Depth: 1025)

Read Width: 8

Read Depth: 1024 (Actual Read Depth: 1025)

ECC, Output Register and Power Gating Options

☐ ECC (Hard ECC) ☐ Single Bit Error Injection ☐ Double Bit Error Injection

☐ ECC Pipeline Reg ☐ Dynamic Power Gating

☐ Output Registers (Embedded Registers)

Initialization

☒ Reset Pin ☒ Enable Reset Synchronization ☐ Enable Safety Circuit

Reset Type: Asynchronous Reset

Full Flags Reset Value: 1

☒ Dout Reset Value: 0 (Hex)

Read Latency: 0

Figure 21. Configuring port width and read mode

### 5-3-9. Browse through the settings of the **Status Flags** and **Data Counts** tabs.

These tabs configure other options for the FIFO Generator. For this design, leave everything at their default settings

### 5-3-10. Select the **Summary** tab.

This tab displays a summary of all the selected configuration options, as well as listing resources used for this configuration.

Basic Native Ports Status Flags Data Counts Summary

**WARNING:** The use of Asynchronous Reset can lead to BRAM data corruption (AR 42571). It is recommended to Enable Sa

Block RAM resource(s) (18K BRAMs): 1

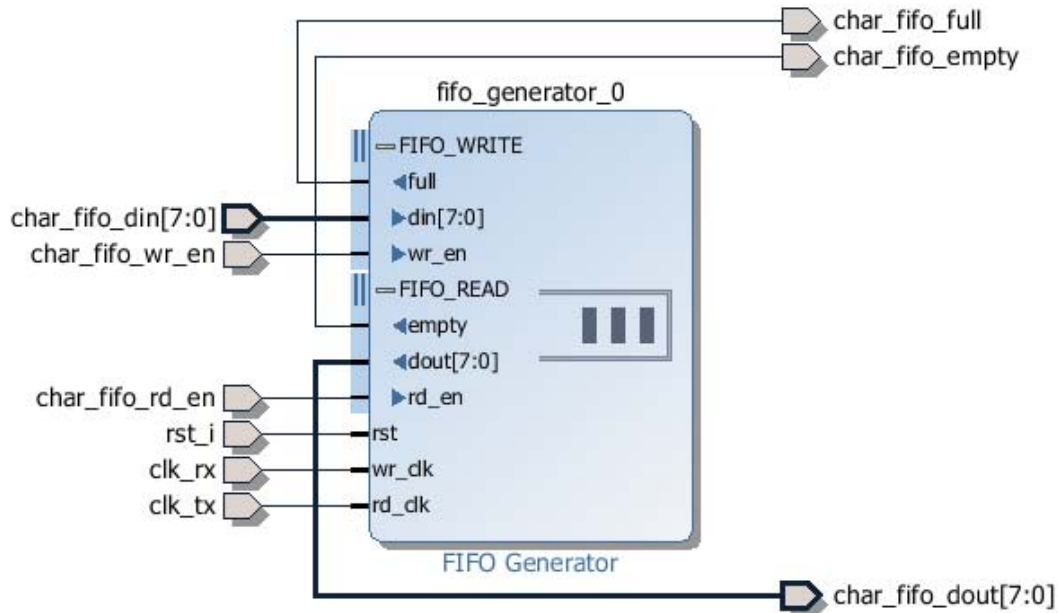
Block RAM resource(s) (36K BRAMs): 0

Clocking Scheme	Independent Clocks
Memory Type	Block RAM
Model Generated	Behavioral Model
Write Width	8
Write Depth	Write Depth
Read Width	8
Almost Full/Empty Flags	Not Selected/Not Selected
Programmable Full/Empty Flags	Not Selected/Not Selected
Data Count Outputs	Not Selected
Handshaking	Not Selected
Read Mode / Reset	First-word Fall-through / Asynchronous
Read Latency (From Rising Edge of Read Clock)	0

Figure 22. Summary page

**5-3-11.** Verify that the information is correct. For this configuration you are using one 18K block RAM. Click **OK**.

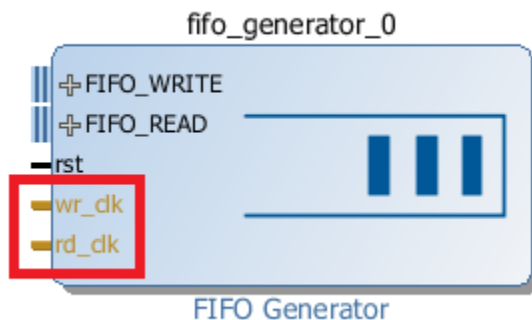
**5-4.** Make the ports external naming them as shown below.



**Figure 23. FIFO Generator IP fully generated and connected**

**5-4-1.** Expand the **FIFO\_WRITE** and **FIFO\_READ** interfaces.

**5-4-2.** Select **wr\_clk** and then press and hold the **Ctrl** key and select the **rd\_clk** ports of the FIFO.



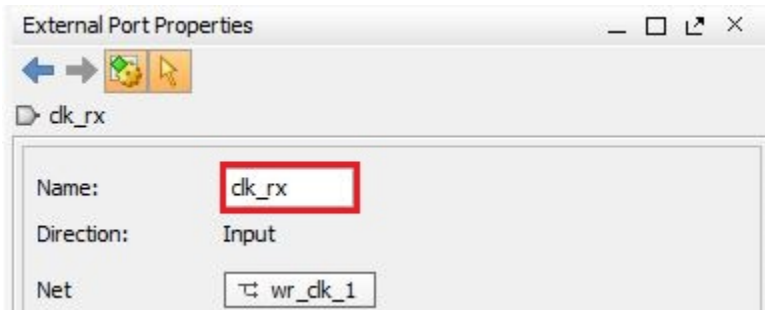
**Figure 24. Selecting multiple ports**

**5-4-3.** With the ports highlighted, right-click and select **Make External**.

Two external connections are created for the selected FIFO ports. Notice that the external connections have the same name as the IP module port that they connect to. You can rename these connections by selecting them and changing the name in the External Port Properties window.

**5-4-4.** Select the external connection port named **wr\_clk**.

**5-4-5.** In the External Port Properties window, in the Name field of the General tab, type the name **clk\_rx** and press **Enter**. Similarly, select the external connection port named **rd\_clk** and change its name to **clk\_tx**.



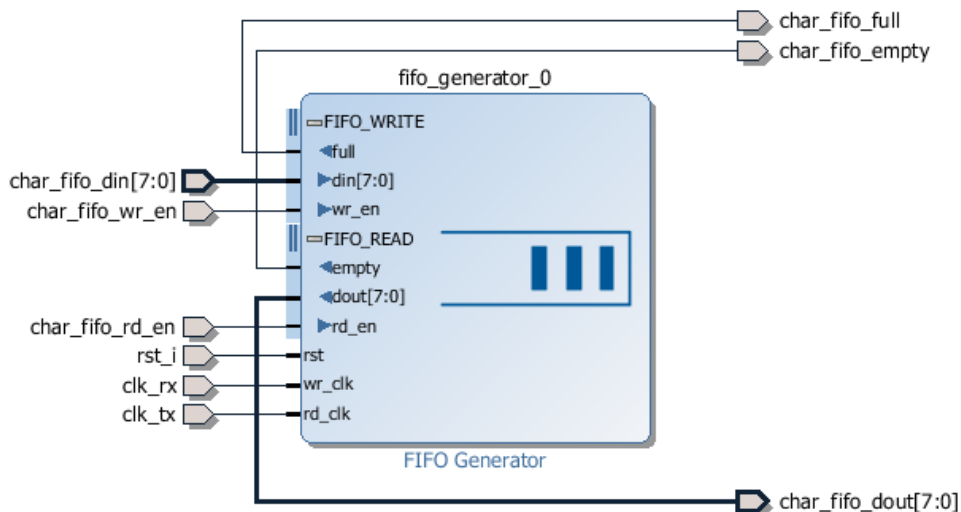
**Figure 25. Changing the external port name**

**5-4-6.** You will need to expand FIFO\_WRITE and FIFO\_READ to see the signal names by clicking on the “+” symbols next to the bus names. While pressing the **Ctrl** key, click all of the remaining FIFO input and output ports and make them external.


**5-4-7.** Change their names as listed below:

- din = char\_fifo\_din
- dout = char\_fifo\_dout
- empty = char\_fifo\_empty
- full = char\_fifo\_full
- rd\_en = char\_fifo\_rd\_en
- wr\_en = char\_fifo\_wr\_en
- rst = rst\_i

When you have finished, your subsystem design should look like the figure below.



**Figure 26. Renamed external ports**

**5-4-8.** Click on Refresh (  ) icon from the vertical toolbar to see the above diagram.

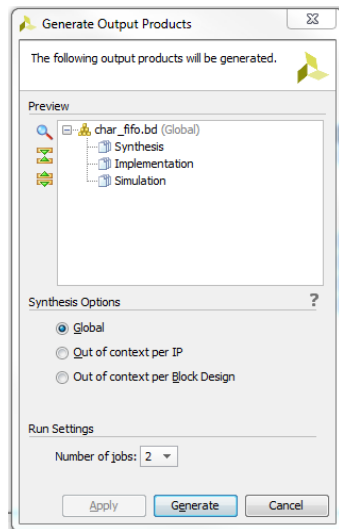
**5-4-9.** Select **Tools > Validate Design**.

You should see a message that validation was successful.

## **5-5. Generate the output product.**

**5-5-1.** In the IP Sources tab of the Sources window, select the **char\_fifo** under the Block Designs.

**5-5-2.** Right-click and select **Generate Output Products**.



**Figure 27. Generating the output products so the IP can be instantiated in the design**

**5-5-3.** Click **Generate** and **OK** to generate the output products.

You should see the various IP output products displayed in the IP Sources tab of the Sources window.

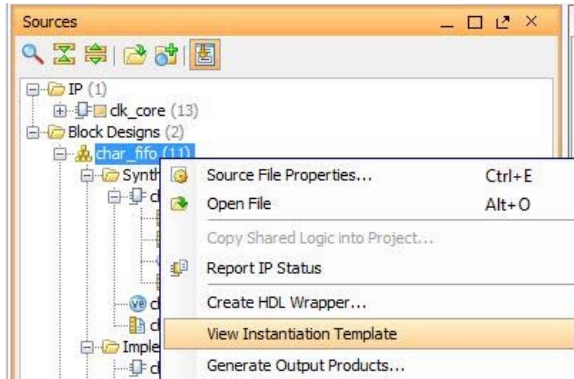


**Figure 28. Generated output products**

## 5-6. Instantiate the char\_fifo IP in the project.

5-6-1. From the *IP Sources* tab of the *Sources* window, select the **char\_fifo** module.

5-6-2. Right-click and select **View Instantiation Template**.



**Figure 29. Generating an instantiation template**

The char\_fifo\_wrapper.v instantiation template is opened in the text editor in the Vivado IDE.

```

42 char_fifo char_fifo_i
43     (.char_fifo_din(char_fifo_din),
44      .char_fifo_dout(char_fifo_dout),
45      .char_fifo_empty(char_fifo_empty),
46      .char_fifo_full(char_fifo_full),
47      .char_fifo_rd_en(char_fifo_rd_en),
48      .char_fifo_wr_en(char_fifo_wr_en),
49      .clk_rx(clk_rx),
50      .clk_tx(clk_tx),
51      .rst_i(rst_i));
52 endmodule

```

**Figure 30. Part of the instantiation template**

5-6-3. Copy lines 42 through line 51, and paste them at or around line 334 in the **wave\_gen.v** file.

5-6-4. Save the Verilog file.

## 5-7. Generate the bitstream and verify the functionality in hardware.

5-7-1. Click on the **Run Implementation** in the *Flow Navigator* pane.

If prompted, click **Yes** to Save the project before proceeding.

5-7-2. Click **OK** to re-run the synthesis process, followed by clicking **Save** to save the design.

5-7-3. When completed, generate the resource utilization report and verify that one FIFO is being used.

5-7-4. In the Flow Navigator, under Program and Debug, click **Generate Bitstream**.

5-7-5. Open the hardware manager and program the FPGA.

**5-7-6.** Open **SDK** by selecting **Start > All Programs > Xilinx Design Tools > SDK 2016.2 > Xilinx SDK 2016.2**

**5-7-7.** In the **Select a workspace** window, click on the browse button, browse to `c:/xup/fpga_flow/2016_2_ZYNQ_Sources/lab4/` directory and select either `c:/xup/fpga_flow/2016_2_ZYNQ_Sources/lab4/Zybo/lab4.sdk` or `c:/xup/fpga_flow/2016_2_ZYNQ_Sources/lab4/ZedBoard/lab4.sdk` and click **OK**.

**5-7-8.** Click **OK**.

In the *Project Explorer*, right-click on the `wave_gen_uart`, select *Run As*, and then **Launch on Hardware (System Debugger)**

verify the functionality of the design in the hardware.

**5-7-9.** When done, close the **Vivado** program by selecting **File > Exit** and click **OK**.

**5-7-10.** Close the **SDK** program by selecting **File > Exit** and click **OK**.

## Conclusion

In this lab, you learned how to add an existing IP during the project creation. You also learned how to use IP Catalog and generate a core. You then instantiated the core in the design, implemented the design, and verified the design in hardware. You also used the IP Integrator capability of the tool to generate a FIFO and then use it in the HDL design.