Creating a Processor System Lab

Introduction

This lab introduces a design flow to generate an IP-XACT adapter from a design using Vivado HLS and use the generated IP-XACT adapter in a processor system using IP Integrator in Vivado.

Objectives

After completing this lab, you will be able to:

- Profile a software application
- Understand the steps and directives involved in creating an IP-XACT adapter in Vivado HLS
- Create a processor system using IP Integrator in Vivado
- Integrate the generated IP-XACT adapter into the created processor system
- Profile an application having an hardware accelerator

The Design

The design consists of a FIR filter to filter a 4 KHz tone added to CD quality (48 KHz) music. The characteristic of the filter is as follows:

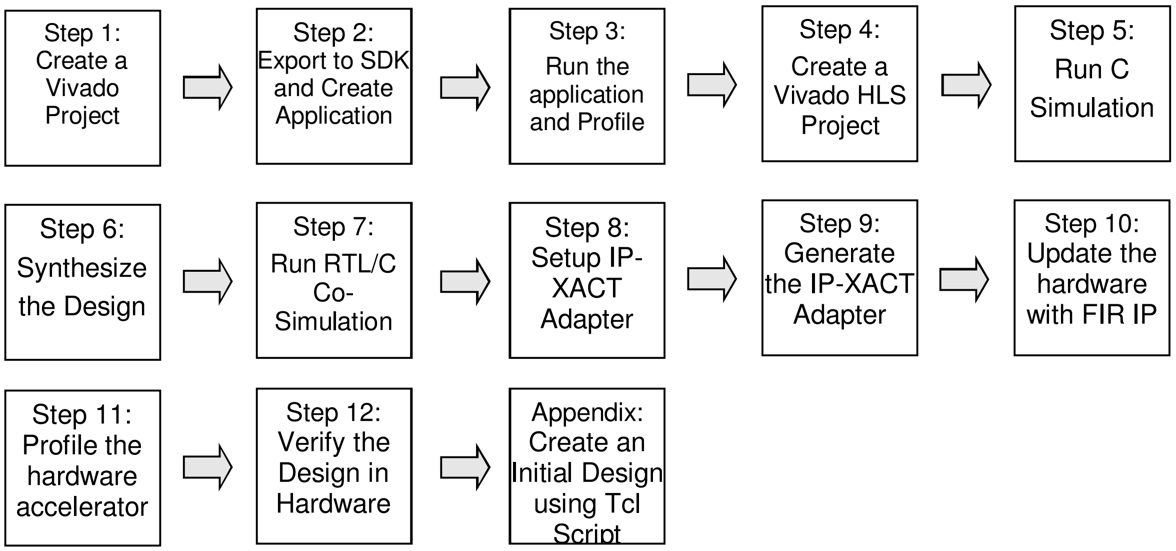
```
FS=48000 Hz
FPASS1=2000 Hz
PSTOP1=3800 Hz
FSTOP2=4200 Hz
FPASS2=6000 Hz
APASS1=APASS2=1 dB
ASTOP=60 dB
```

This lab requires you to develop a peripheral core of the designed filter that can be instantiated in a processor system.

Procedure

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

General Flow for this Lab


Step 1: Create a Vivado Project
Step 2: Export to SDK and Create Application
Step 3: Run the application and Profile
Step 4: Create a Vivado HLS Project
Step 5: Run C Simulation
Step 6: Synthesize the Design
Step 7: Run RTL/C Co-Simulation
Step 8: Setup IP-XACT Adapter
Step 9: Generate the IP-XACT Adapter
Step 10: Update the hardware with FIR IP
Step 11: Profile the hardware accelerator
Step 12: Verify the Design in Hardware
Appendix: Create an Initial Design using Tcl Script

Create a Vivado Project Step 1

1-1. Launch Vivado Tcl Shell and run the provided tcl script to create an initial system targeting either the ZedBoard (having xc7z020clg484-1 device) or the Zybo (having xc7z010clg400-1 device).

If you want to create the system from scratch then follow the steps provided in Appendix and then continue from step 1-2 below.

References to <2014_2_zynq_labs> means c:\xup\sys_design\2014_2_zynq_labs and <2014_2_zynq_sources> means c:\xup\sys_design\2014_2_zynq_sources directories.

1-1-1. Open Vivado Tcl Shell by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2014.2 > Vivado 2014.2 Tcl Shell**

1-1-2. In the shell window, either change the directory to *<2014_2_zynq_sources>/lab6_zed* for the *ZedBoard* or *<2014_2_zynq_sources>/lab6_zybo* for the *Zybo* using the **cd** account.

1-1-3. Run the provided script file to create an initial system having either zed_audio_ctrl for the Zedboard or zybo_audio_ctrl for the Zybo, and GPIO peripherals by typing the following command:

source audio_project_create.tcl

The script will be run and the initial system, shown below, will be created.

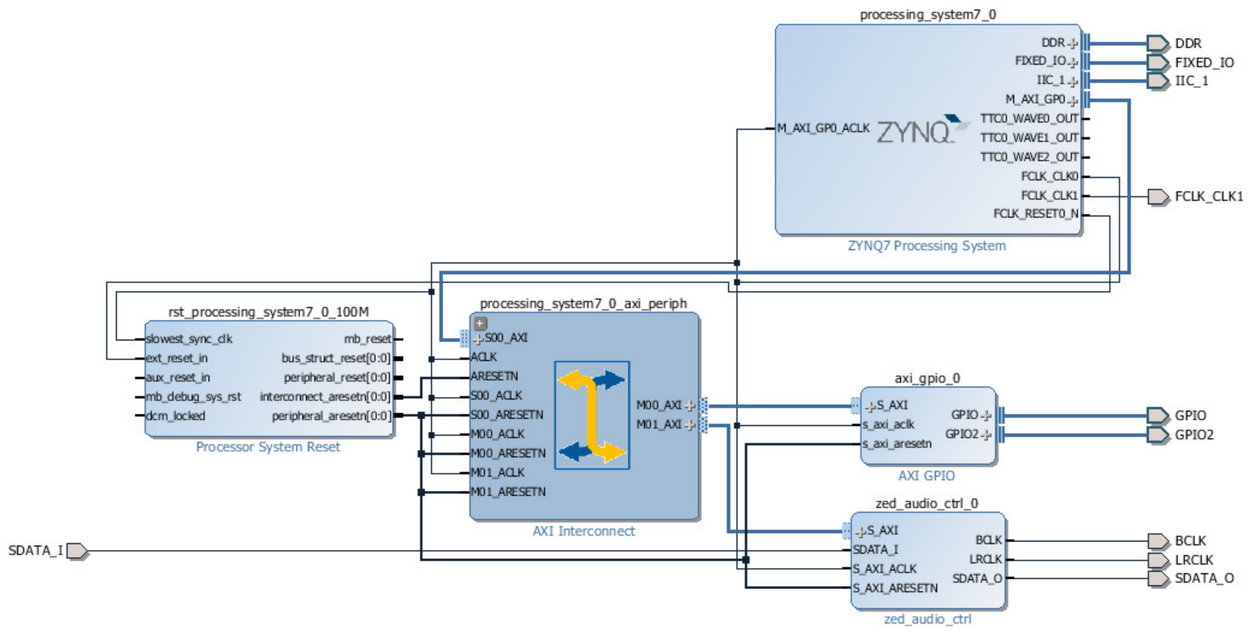


Figure 1. Block design after I2C based zed_audio_ctrl core added and connections made for the ZedBoard

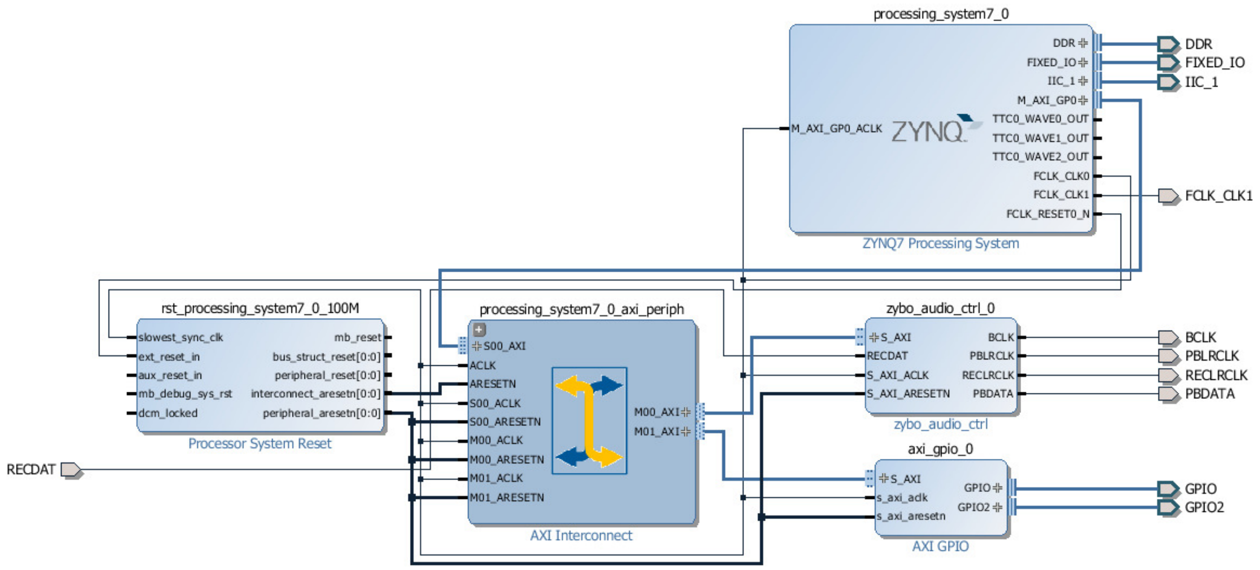
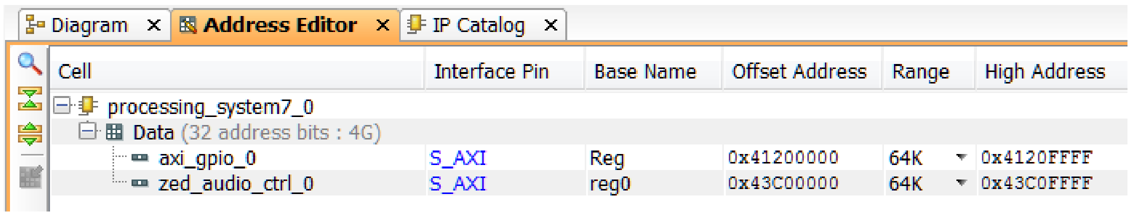


Figure 1. Block design after I2C based zybo_audio_ctrl core added and connections made for the Zybo

1-2. Verify addresses and validate the design. Generate the system_wrapper file, and add the provided Xilinx Design Constraints (XDC) file either from the *<2014_2_zynq_sources>\lab6_zed* for the *ZedBoard* or *<2014_2_zynq_sources>\lab6_zybo* for the *Zybo* directory.

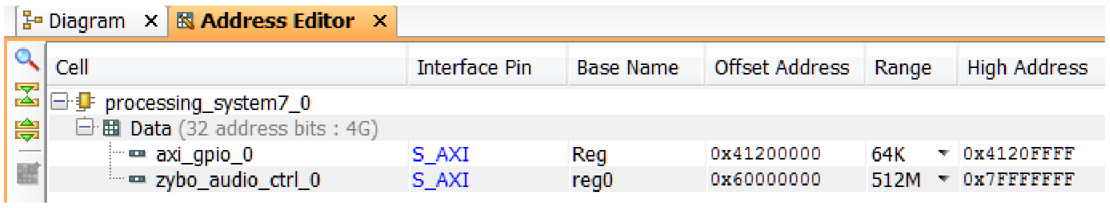
1-2-1. Click on the *Address Editor*, and expand the **processing_system7_0 > Data** if necessary.

The generated address map should look like as shown below.



Cell	Interface Pin	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 4G)					
axi_gpio_0	S_AXI	Reg	0x41200000	64K	0x4120FFFF
zed_audio_ctrl_0	S_AXI	reg0	0x43C00000	64K	0x43C0FFFF

Figure 2. Generated address map for the ZedBoard



Cell	Interface Pin	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 4G)					
axi_gpio_0	S_AXI	Reg	0x41200000	64K	0x4120FFFF
zybo_audio_ctrl_0	S_AXI	reg0	0x60000000	512M	0x7FFFFFFF

Figure 2. Generated address map for the Zybo

1-2-2. Run *Design Validation* (**Tools > Validate Design**) and verify there are no errors

1-2-3. In the *sources* view, right-click on the block diagram file, **system.bd**, and select **Create HDL Wrapper** to update the HDL wrapper file. When prompted, click **OK** with the *Let Vivado manage wrapper and auto-update* option.

1-2-4. Click **Add Sources** in the *Flow Navigator* pane, select **Add or Create Constraints**, and click **Next**.

1-2-5. Click the **Add Files** button, browse either to **<2014_2_zynq_sources>\lab6_zed** folder and select **zed_audio_constraints.xdc** or the **<2014_2_zynq_sources>\lab6_zybo** folder and select **zybo_audio_constraints.xdc**

1-2-6. Click **Finish** to add the file.

1-2-7. Click on the **Generate Bitstream** in the Flow Navigator to run the synthesis, implementation, and bitstream generation processes.

1-2-8. Click **Save** and **Yes** if prompted.

1-2-9. When the bit generation is completed, a selection box will be displayed with the *Open Implemented Design* option selected. Click **OK**.

Export to SDK and Create an Application Project Step 2

2-1. Export the hardware along with the generated bitstream to SDK. Create a Board Support Package enabling profiling.

To Export the hardware, the block diagram must be open and the Implemented design must be open.

2-1-1. If it is not already open, click **Open Block Design** (under IP Integrator in the Flow Navigator)

2-1-2. If it is not already open, click **Open Implemented Design** (under Implementation)

2-1-3. Select **File > Export > Export Hardware**

2-1-4. Make sure that **Include Bitstream** option is selected and click **OK**, leaving the target directory set to local project directory.

2-1-5. Select **File > Launch SDK**

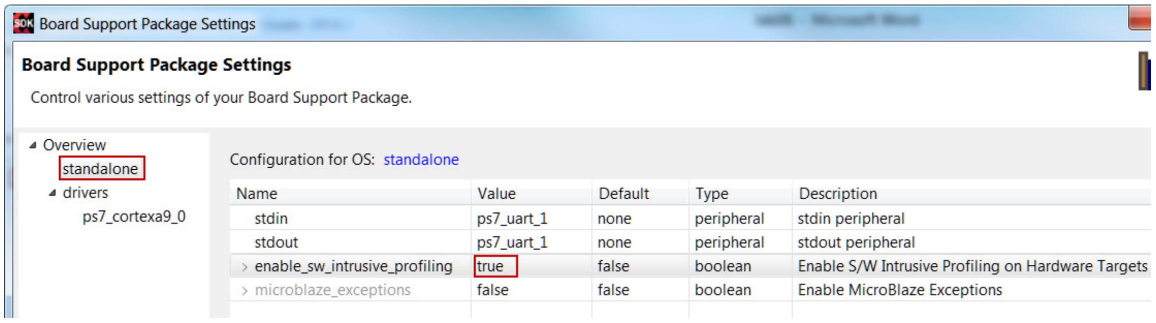
2-1-6. Click **OK**.

2-1-7. In **SDK**, select **File** > **New** > **Board Support Package.**

2-1-8. Click **Finish** with the default settings (with standalone operating system).

This will open the Software Platform Settings form showing the OS and libraries selections.

2-1-9. Select the **Overview > standalone** entry in the left pane, click on the drop-down arrow of the *enable_sw_intrusive_profiling Value* field and select **true**.



Board Support Package Settings

Board Support Package Settings

Control various settings of your Board Support Package.

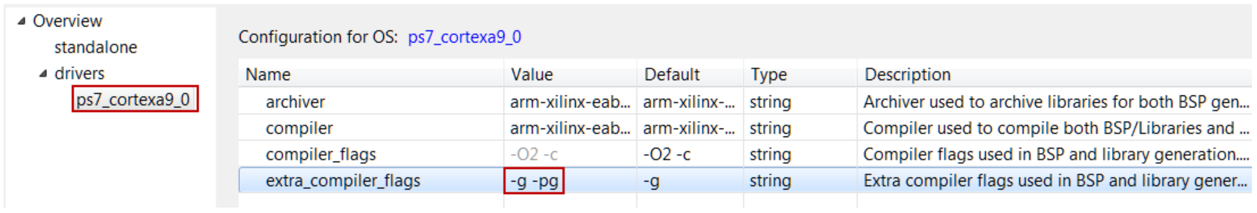
- Overview
  - standalone
  - drivers
    - ps7_cortexa9_0

Configuration for OS: standalone

Name	Value	Default	Type	Description
stdin	ps7_uart_1	none	peripheral	stdin peripheral
stdout	ps7_uart_1	none	peripheral	stdout peripheral
enable_sw_intrusive_profiling	true	false	boolean	Enable S/W Intrusive Profiling on Hardware Targets
microblaze_exceptions	false	false	boolean	Enable MicroBlaze Exceptions

Figure 3. Setting up for profiling

2-1-10. Select the **Overview > drivers > cpu_cortexa9** and add **–pg** in addition to the –g in the *extra_compiler_flags Value* field.



- Overview
  - standalone
  - drivers
    - ps7_cortexa9_0

Configuration for OS: ps7_cortexa9_0

Name	Value	Default	Type	Description
archiver	arm-xilinx-eab...	arm-xilinx-...	string	Archiver used to archive libraries for both BSP gen...
compiler	arm-xilinx-eab...	arm-xilinx-...	string	Compiler used to compile both BSP/Libraries and ...
compiler_flags	-O2 -c	-O2 -c	string	Compiler flags used in BSP and library generation....
extra_compiler_flags	-g -pg	-g	string	Extra compiler flags used in BSP and library gener...

Figure 4. Adding profiling switch

2-1-11. Click **OK** to accept the settings and create the BSP.

2-2. Create an application for profiling.

2-2-1. Select **File** > **New** > **Application Project.**

2-2-2. Enter **lab6** as the *Project Name*, and for *Board Support Package*, choose **Use Existing** (*standalone_bsp_0* should be the only option).

2-2-3. Click **Next**, and select *Empty Application* and click **Finish.**

2-2-4. Select **lab6** in the project view, right-click the *src* folder, and select **Import.**

2-2-5. Expand **General** category and double-click on **File System.**

2-2-6. Browse to either **<2014_2_zynq_sources>\lab6_zed** for the ZedBoard or **<2014_2_zynq_sources>\lab6_zybo** for the Zybo and click **OK.**

2-2-7. Select **lab6.c, audio.h,** and **fir_coef.dat**, and click **Finish** to add the file to the project. (Ignore any errors/warnings for now).

Run the Application and Profile Step 3

5-1. Zybo: Make sure that the JP7 is set to select USB power.

Connect a micro-usb cable between a PC and the JTAG port of the board. Power ON the board. Program the PL section and run the application using the user defined SW_PROFILE symbol.

5-1-1. **Zybo only:** Make sure that the JP7 is set to select USB power.

5-1-2. Connect a micro-usb cable between a PC and the JTAG port of the board.

5-1-3. Power ON the board.

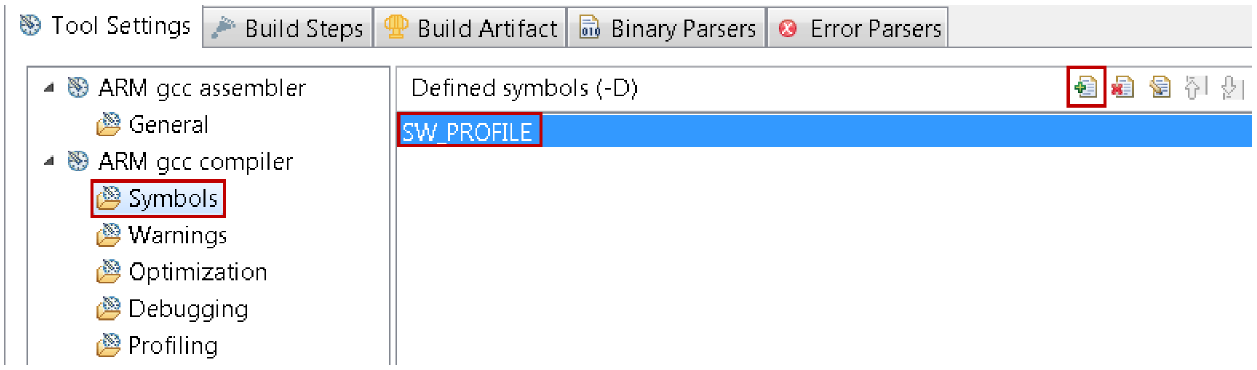
5-1-4. Select **Xilinx Tools > Program FPGA**.

5-1-5. Click on the **Program** button to download the bitstream and program the PL section.

5-1-6. Select the *lab6* application, right-click, and select **C/C++ Build Settings**.

5-1-7. Under the **ARM gcc compiler** group, select the **Symbols** sub-group**,** click on the **+** button to open the value entry form, enter **SW_PROFILE**, and click **OK**.

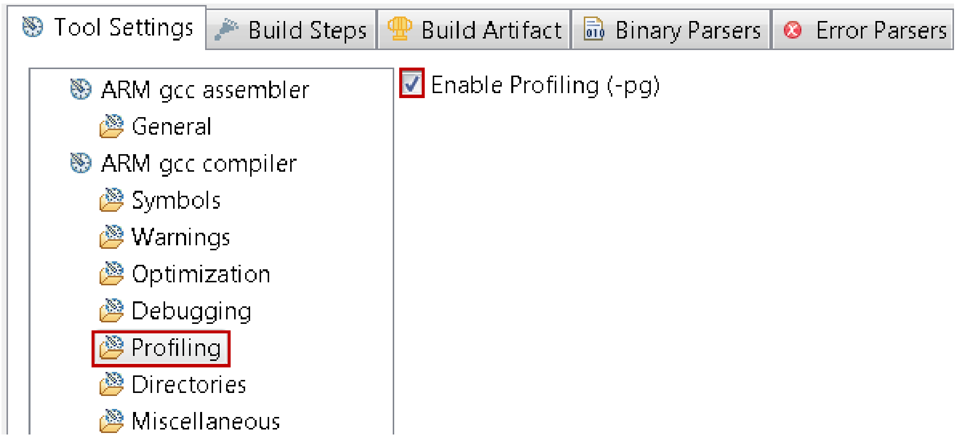
This will allow us to profile the software loop of the FIR application.



SW_PROFILE

Figure 5. Add a user-defined symbol

5-1-8. Under the **ARM gcc compiler** group, select the **Profiling** sub-group, then check the **Enable Profiling** box, and click **OK**.



Enable Profiling (-pg) [✓]

Figure 6. Compiler setting for enabling profiling

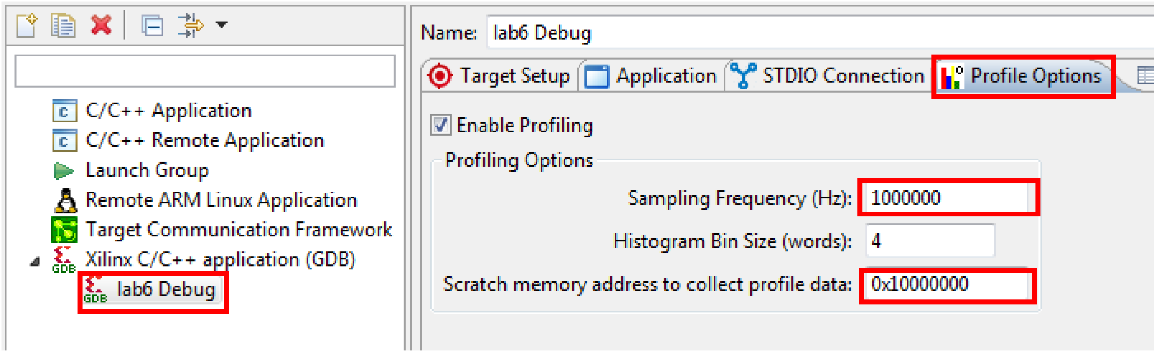
5-2. Set SW[1] to the up position. Create a Run Configuration, enabling the Profile option and setting up the profiling parameters. Run the profiler and gprof and analyze the results.

5-2-1. Make sure that the SW[1] is in the ON (up) position.

5-2-2. Select lab6, right-click and select **Run As > Run Configurations…** and double-click **Xilinx C/C++ Application** to create a new configuration.

5-2-3. Select the **Profile Options** tab. Click on the *Enable Profiling* check box, enter **1000000** (1 MHz) in the Sampling Frequency field, enter **0x10000000** in the scratch memory address field, and click **Apply**.

Use the high end address as the scratch memory address.



Name: lab6 Debug

Enable Profiling [✓]

Sampling Frequency (Hz): 1000000

Histogram Bin Size (words): 4

Scratch memory address to collect profile data: 0x10000000

Figure 7. Profiling options

5-2-4. Click the **Run** button to download the application and execute it.

When program execution has completed, a message will be displayed indicating that the profiling results are being saved in gmon.out file at the lab6\Debug directory.

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

5-2-6. Expand the *Debug* folder under the **lab6** project in the Project Explorer view, and double click on the **gmon.out** entry.

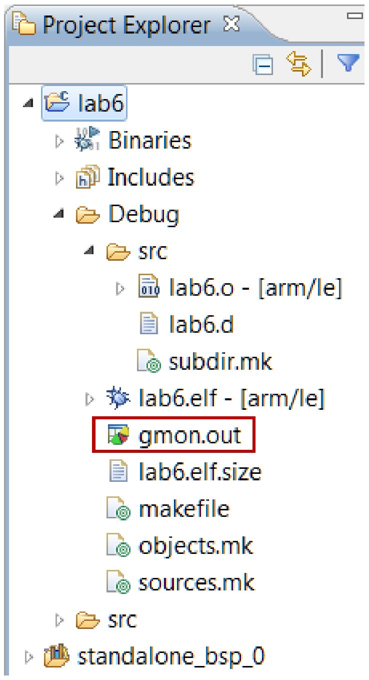
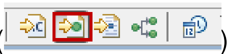
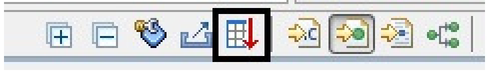


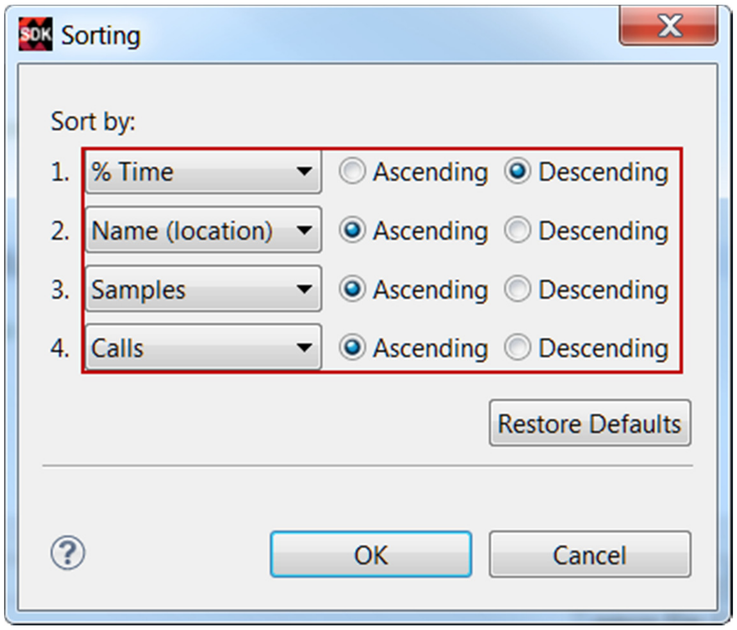
Figure 8. Invoking gprof on gmon.out

5-2-7. The Gmon File Viewer dialog box will appear showing *lab6.elf* as the corresponding binary file. Click **OK**.

5-2-8. Click on the **Sort samples per function** button ().

5-2-9. Click in the **%Time** column to sort in the descending order. You can also click on the sort button .

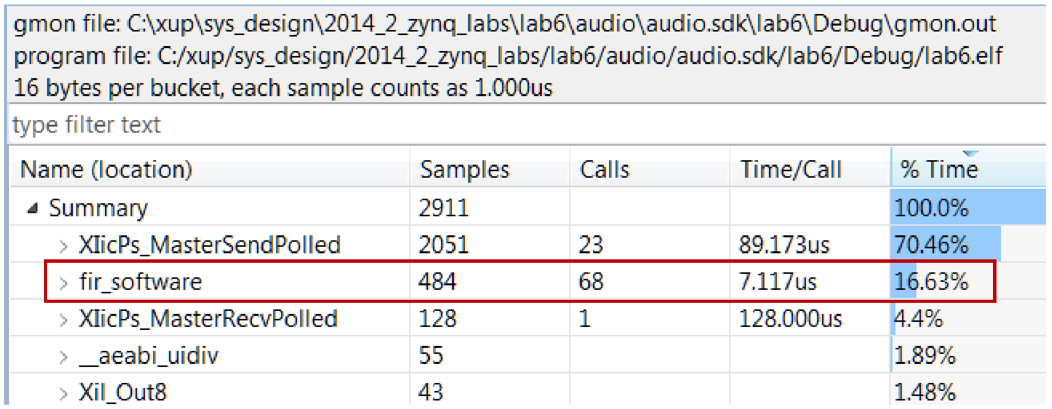
If you click on the sort button, set the sorting items, orders, and methods as shown below:


Sorting

Sort by:

1.	% Time	Ascending	[✓ Descending]
2.	Name (location)	[✓ Ascending]	Descending
3.	Samples	[✓ Ascending]	Descending
4.	Calls	[✓ Ascending]	Descending

Note that the fir_software routine is called 68 times, 478 samples were taken during the profiling, and on an average of 7.117 microseconds for ZedBoard or 7.029 microseconds for Zybo were spent per call. Note that the samples and time/call may vary little bit.

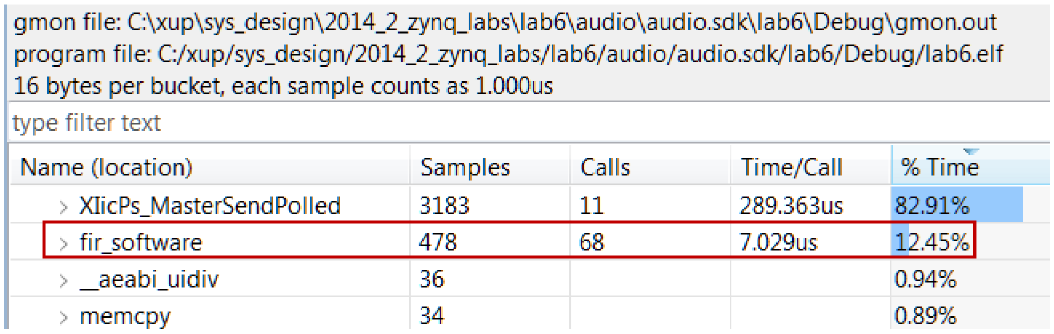


gmon file: C:\xup\sys_design\2014_2_zynq_labs\lab6\audio\audio.sdk\lab6\Debug\gmon.out
program file: C:/xup/sys_design/2014_2_zynq_labs/lab6/audio/audio.sdk/lab6/Debug/lab6.elf
16 bytes per bucket, each sample counts as 1.000us

type filter text

Name (location)	Samples	Calls	Time/Call	% Time
Summary	2911			100.0%
XIicPs_MasterSendPolled	2051	23	89.173us	70.46%
fir_software	484	68	7.117us	16.63%
XIicPs_MasterRecvPolled	128	1	128.000us	4.4%
__aeabi_uidiv	55			1.89%
Xil_Out8	43			1.48%

Figure 9. Sorting results for ZedBoard



gmon file: C:\xup\sys_design\2014_2_zynq_labs\lab6\audio\audio.sdk\lab6\Debug\gmon.out
program file: C:/xup/sys_design/2014_2_zynq_labs/lab6/audio/audio.sdk/lab6/Debug/lab6.elf
16 bytes per bucket, each sample counts as 1.000us

type filter text

Name (location)	Samples	Calls	Time/Call	% Time
XIicPs_MasterSendPolled	3183	11	289.363us	82.91%
fir_software	478	68	7.029us	12.45%
__aeabi_uidiv	36			0.94%
memcpy	34			0.89%

Figure 9. Sorting results for Zybo

5-2-10. In Vivado, select **File > Close Implemented Design**.

5-2-11. Click **OK**.

Create a Vivado HLS Project Step 4

6-1. Create a new project in Vivado HLS targeting either xc7z010clg484-1 for the ZedBoard or the xc7z010clg400-1 for the Zybo.

6-1-1. Launch Vivado HLS: Select **Start > All Programs > Xilinx Design Tools > Vivado 2014.2 > Vivado HLS > Vivado HLS 2014.2**

A Getting Started GUI will appear.

6-1-2. In the Quick Start section, click on **Create New Project.** The New Vivado HLS Project wizard opens.

6-1-3. Click **Browse…** button of the **Location** field, browse to **<2014_2_zynq_labs>\lab6,** and then click **OK.**

6-1-4. For Project Name, type ***fir.prj***

6-1-5. Click **Next.**

6-1-6. In the *Add/Remove Files* for the source files, type **fir** as the function name (the provided source file contains the function, to be synthesized, called fir).

6-1-7. Click the **Add Files…** button, select *fir.c* and *fir_coef.dat* files either from the **<2014_2_zynq_sources>\lab6_zed** for the ZedBoard or **<2014_2_zynq_sources>\lab6_zybo** for the Zybo, and then click **Open**.

6-1-8. Click **Next**.

6-1-9. In the *Add/Remove Files* for the testbench, click the **Add Files…** button, select *fir_test.c* file either from the **<2014_2_zynq_sources>\lab6_zed** for the ZedBoard or **<2014_2_zynq_sources>\lab6_zybo** for the Zybo and click **Open.**

6-1-10. Click **Next.**

6-1-11. In the *Solution Configuration* page, leave *Solution* Name field as **solution1** and either leave the default period of **10** for the ZedBoard or change the *clock period* to **8** for the Zybo. Leave *Uncertainty* field blank as it will use 12.5% as the default value.

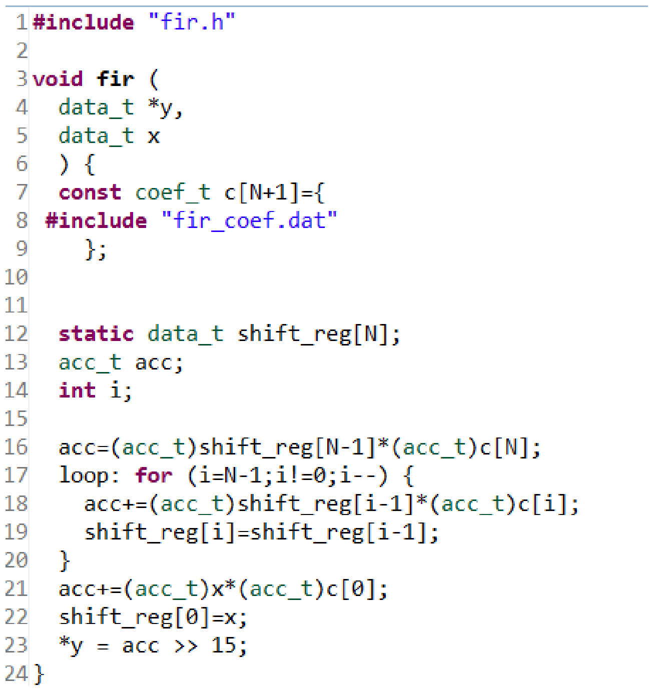
6-1-12. Click on Part's Browse button, and select the following filters to select either the **xc7z010clg484-1** for the ZedBoard or the **xc7z010clg400-1** for the Zybo, and click **OK**:

Family: **Zynq**
Sub-Family: **Zynq**
Package: **clg484** (for ZedBoard) or **clg400** (for Zybo)
Speed Grade: **–1**

6-1-13. Click **Finish.**

You will see the created project in the Explorer view. Expand various sub-folders to see the entries under each sub-folder.

6-1-14. Double-click on the **fir.c** under the source folder to open its content in the information pane.

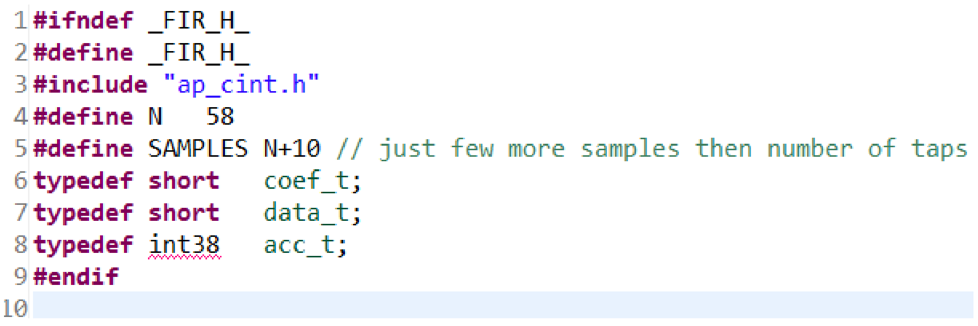


```
 1 #include "fir.h"
 2
 3 void fir (
 4   data_t *y,
 5   data_t x
 6   ) {
 7   const coef_t c[N+1]={
 8  #include "fir_coef.dat"
 9     };
10
11
12   static data_t shift_reg[N];
13   acc_t acc;
14   int i;
15
16   acc=(acc_t)shift_reg[N-1]*(acc_t)c[N];
17   loop: for (i=N-1;i!=0;i--) {
18     acc+=(acc_t)shift_reg[i-1]*(acc_t)c[i];
19     shift_reg[i]=shift_reg[i-1];
20   }
21   acc+=(acc_t)x*(acc_t)c[0];
22   shift_reg[0]=x;
23   *y = acc >> 15;
24 }
```

Figure 10. The design under consideration

The FIR filter expects x as a sample input and pointer to the computed sample out. Both of them are defined of data type data_t. The coefficients are loaded in array c of type coef_t from the file called fir_coef.dat located in the current directory. The sequential algorithm is applied and accumulated value (sample out) is computed in variable acc of type acc_t.

6-1-15. Double-click on the **fir.h** in the *outline* tab to open its content in the information pane.



```
#ifndef _FIR_H_
#define _FIR_H_
#include "ap_cint.h"
#define N   58
#define SAMPLES N+10 // just few more samples then number of taps
typedef short   coef_t;
typedef short   data_t;
typedef int38   acc_t;
#endif

```

Figure 11. The header file

The header file includes ap_cint.h so user defined data width (of arbitrary precision) can be used. It also defines number of taps (N), number of samples to be generated (in the testbench), and data types coef_t, data_t, and acc_t. The coef_t and data_t are short (16 bits). Since the algorithm iterates (multiply and accumulate) over 59 taps, there is a possibility of bit growth of 6 bits and hence acc_t is defined as int38. Since the acc_t is bigger than sample and coefficient width, they have to cast before being used (like in lines 16, 18, and 21 of fir.c).

6-1-16. Double-click on the **fir_test.c** under the testbench folder to open its content in the information pane.

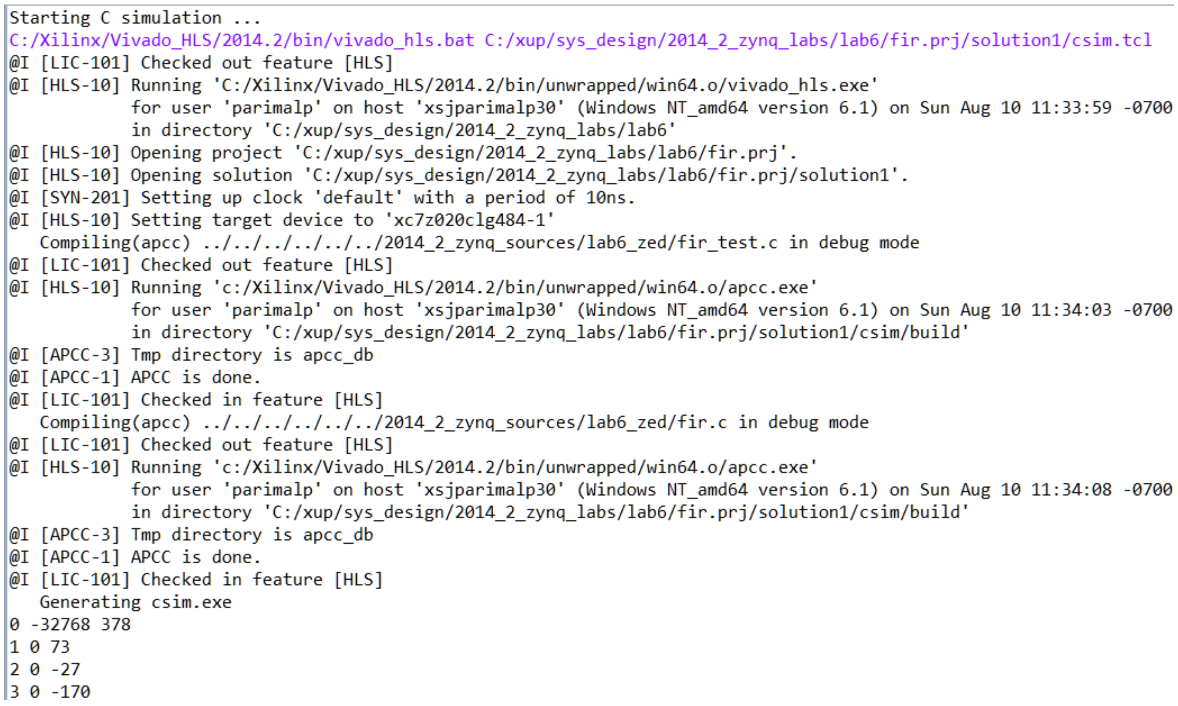
Notice that the testbench opens fir_impulse.dat in write mode, and sends an impulse (first sample being 0x8000.

Run C Simulation Step 5

7-1. Run C simulation to observe the expected output.

7-1-1. Select **Project > Run C Simulation** or click on from the tools bar buttons, and Click **OK** in the *C Simulation Dialog* window.

The testbench will be compiled using apcc compiler and csim.exe file will be generated. The csim.exe will then be executed and the output will be displayed in the console view.



```
Starting C simulation ...
C:/Xilinx/Vivado_HLS/2014.2/bin/vivado_hls.bat C:/xup/sys_design/2014_2_zynq_labs/lab6/fir.prj/solution1/csim.tcl
@I [LIC-101] Checked out feature [HLS]
@I [HLS-10] Running 'C:/Xilinx/Vivado_HLS/2014.2/bin/unwrapped/win64.o/vivado_hls.exe'
            for user 'parimalp' on host 'xsjparimalp30' (Windows NT_amd64 version 6.1) on Sun Aug 10 11:33:59 -0700
            in directory 'C:/xup/sys_design/2014_2_zynq_labs/lab6'
@I [HLS-10] Opening project 'C:/xup/sys_design/2014_2_zynq_labs/lab6/fir.prj'.
@I [HLS-10] Opening solution 'C:/xup/sys_design/2014_2_zynq_labs/lab6/fir.prj/solution1'.
@I [SYN-201] Setting up clock 'default' with a period of 10ns.
@I [HLS-10] Setting target device to 'xc7z020clg484-1'
   Compiling(apcc) ../../../../../../2014_2_zynq_sources/lab6_zed/fir_test.c in debug mode
@I [LIC-101] Checked out feature [HLS]
@I [HLS-10] Running 'c:/Xilinx/Vivado_HLS/2014.2/bin/unwrapped/win64.o/apcc.exe'
            for user 'parimalp' on host 'xsjparimalp30' (Windows NT_amd64 version 6.1) on Sun Aug 10 11:34:03 -0700
            in directory 'C:/xup/sys_design/2014_2_zynq_labs/lab6/fir.prj/solution1/csim/build'
@I [APCC-3] Tmp directory is apcc_db
@I [APCC-1] APCC is done.
@I [LIC-101] Checked in feature [HLS]
   Compiling(apcc) ../../../../../../2014_2_zynq_sources/lab6_zed/fir.c in debug mode
@I [LIC-101] Checked out feature [HLS]
@I [HLS-10] Running 'c:/Xilinx/Vivado_HLS/2014.2/bin/unwrapped/win64.o/apcc.exe'
            for user 'parimalp' on host 'xsjparimalp30' (Windows NT_amd64 version 6.1) on Sun Aug 10 11:34:08 -0700
            in directory 'C:/xup/sys_design/2014_2_zynq_labs/lab6/fir.prj/solution1/csim/build'
@I [APCC-3] Tmp directory is apcc_db
@I [APCC-1] APCC is done.
@I [LIC-101] Checked in feature [HLS]
   Generating csim.exe
0 -32768 378
1 0 73
2 0 -27
3 0 -170
```

Figure 12. Initial part of the generated output in the Console view

You should see the filter coefficients being computed.

Synthesize the Design Step 6

8-1. Synthesize the design with the defaults. View the synthesis results and answer the question listed in the detailed section of this step.

8-1-1. Select **Solution > Run C Synthesis > Active Solution** to start the synthesis process.

8-1-2. When synthesis is completed, several report files will become accessible and the Synthesis Results will be displayed in the information pane.

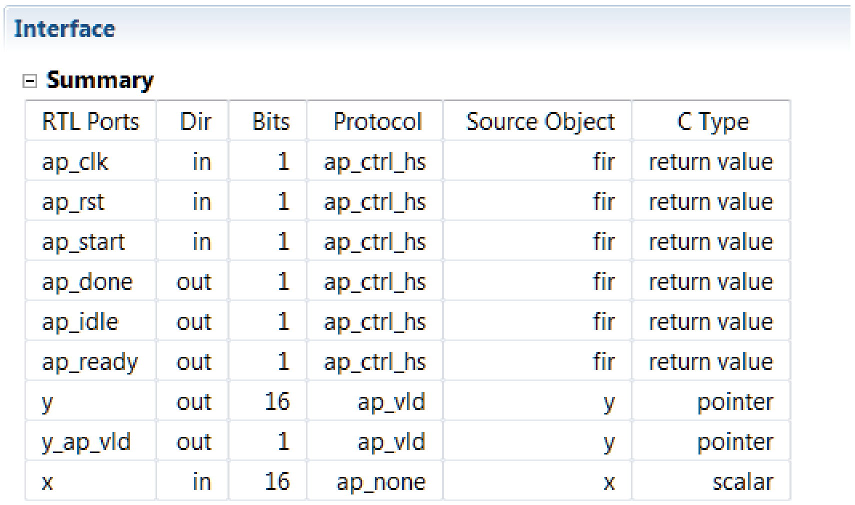
8-1-3. The Synthesis Report shows the performance and resource estimates as well as estimated latency in the design.

8-1-4. Using scroll bar on the right, scroll down into the report and answer the following question.

Question 1

Estimated clock period:	______________________
Worst case latency:	______________________
Number of DSP48E used:	______________________
Number of BRAMs used:	______________________
Number of FFs used:	______________________
Number of LUTs used:	______________________

8-1-5. The report also shows the top-level interface signals generated by the tools.



Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	fir	return value
ap_rst	in	1	ap_ctrl_hs	fir	return value
ap_start	in	1	ap_ctrl_hs	fir	return value
ap_done	out	1	ap_ctrl_hs	fir	return value
ap_idle	out	1	ap_ctrl_hs	fir	return value
ap_ready	out	1	ap_ctrl_hs	fir	return value
y	out	16	ap_vld	y	pointer
y_ap_vld	out	1	ap_vld	y	pointer
x	in	16	ap_none	x	scalar

Figure 13. Generated interface signals

You can see the design expects x input as 16-bit scalar and outputs y via pointer of the 16-bit data. It also has ap_vld signal to indicate when the result is valid.

8-2. Add PIPELINE directive to loop and re-synthesize the design. View the synthesis results.

8-2-1. Make sure that the **fir.c** is open in the information view.

8-2-2. Select the **Directive** tab, right click on *loop* and select the **PIPELINE** directive and click **OK** to apply it.

8-2-3. Select **Solution > Run C Synthesis > Active Solution** to start the synthesis process.

8-2-4. When synthesis is completed, the Synthesis Results will be displayed in the information pane.

8-2-5. Note that the latency has reduced to 64 clock cycles. The DSP48 and BRAM consumption remains same; however, LUT and FF consumptions have changed.

Run RTL/C CoSimulation Step 7

9-1. Run the RTL/C Co-simulation, selecting SystemC and skipping VHDL and Verilog. Verify that the simulation passes.

9-1-1. Select **Solution > Run C/RTL Cosimulation** or click on the ☑ button to open the dialog box so the desired simulations can be run.

A C/RTL Co-simulation Dialog box will open.

9-1-2. Select SystemC option and click **OK** to run the SystemC simulation.

The Co-simulation will run, generating and compiling several files, and then simulating the design. In the console window you can see the progress. When done the RTL Simulation Report shows that it was successful and the latency reported was 64.

Setup IP-XACT Adapter Step 8

10-1. Add RESOURCE directives to create AXI4LiteS adapters so IP-XACT adapter can be generated during the RTL Export step.

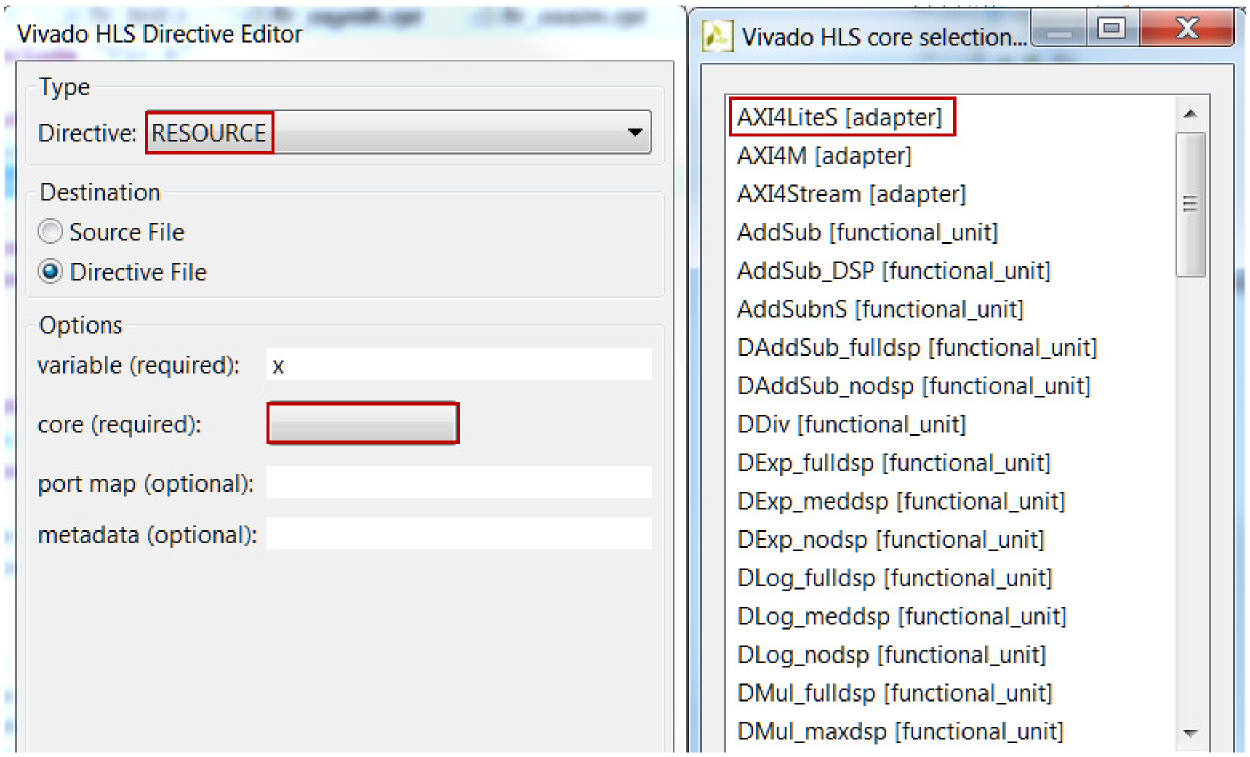
10-1-1. Make sure that **fir.c** file is open and in focus in the information view.

10-1-2. Select the **Directive** tab.

10-1-3. Right-click **x**, and click on **Insert Directive….**

10-1-4. In the **Vivado HLS Directive Editor** dialog box, select **RESOURCE** directive using the drop-down button.

10-1-5. Click on the button beside **core (required)**. Available cores list will pop-up. Select **AXI4LiteS [adapter]**, and click **OK**.



Vivado HLS Directive Editor

Type

Directive: RESOURCE

Destination

( ) Source File
(●) Directive File

Options

variable (required): x

core (required):

port map (optional):

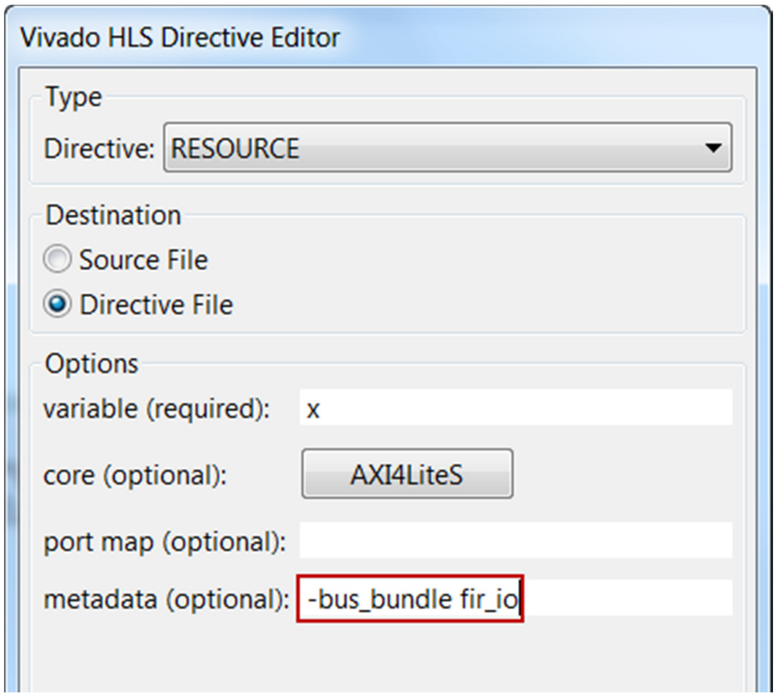
metadata (optional):

Vivado HLS core selection...

AXI4LiteS [adapter]
AXI4M [adapter]
AXI4Stream [adapter]
AddSub [functional_unit]
AddSub_DSP [functional_unit]
AddSubnS [functional_unit]
DAddSub_fulldsp [functional_unit]
DAddSub_nodsp [functional_unit]
DDiv [functional_unit]
DExp_fulldsp [functional_unit]
DExp_meddsp [functional_unit]
DExp_nodsp [functional_unit]
DLog_fulldsp [functional_unit]
DLog_meddsp [functional_unit]
DLog_nodsp [functional_unit]
DMul_fulldsp [functional_unit]
DMul_maxdsp [functional_unit]

Figure 14. Selecting the AXI4LiteS adapter

10-1-6. In the **metadata (optional)** field, type **-bus_bundle fir_io** to associate input, output, and handshaking signals (done through next two steps) into one AXI4Lite adapter called **fir_io**. Click **OK**.



Vivado HLS Directive Editor

Type

Directive: RESOURCE

Destination

( ) Source File
(●) Directive File

Options

variable (required): x

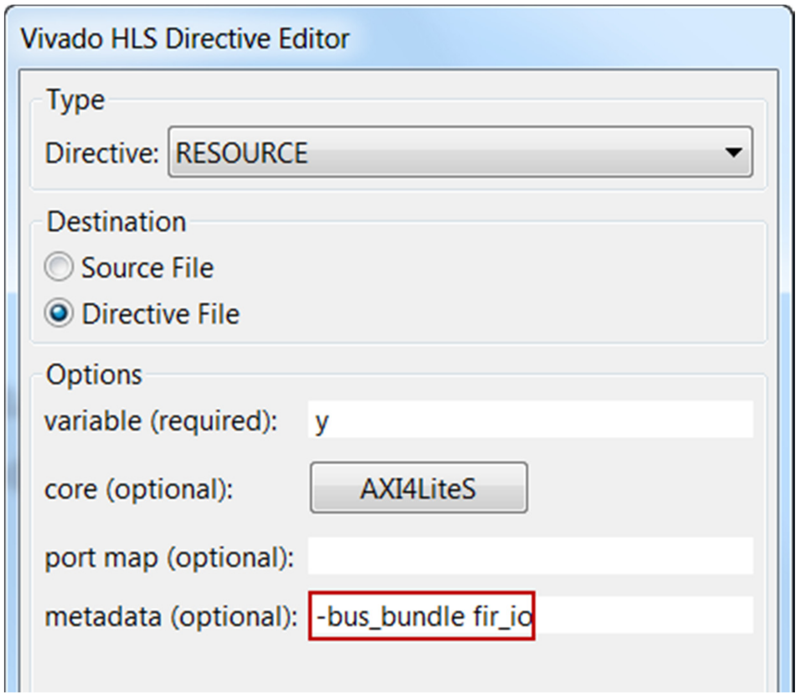
core (optional): AXI4LiteS

port map (optional):

metadata (optional): -bus_bundle fir_io

Figure 15. Applying metadata to assign x input to AXI4Lite adapter

10-1-7. Similarly, apply the **RESOURCE** directive (including metadata) to the **y** output.


Vivado HLS Directive Editor

Type

Directive: RESOURCE

Destination

( ) Source File

(●) Directive File

Options

variable (required): y

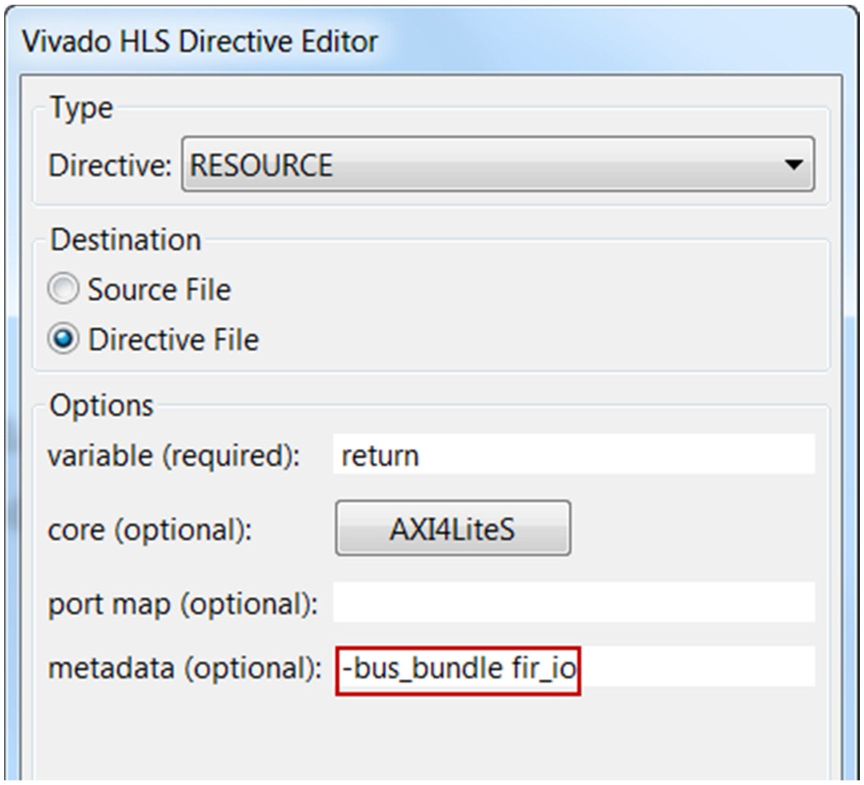
core (optional): AXI4LiteS

port map (optional):

metadata (optional): -bus_bundle fir_io

Figure 16. Applying metadata to assign y output to AXI4Lite adapter

10-1-8. Apply the **RESOURCE** directive to the top-level module **fir.** Doing this will include ap_start, ap_done, and ap_idle signals as part of bus adapter (the variable name shown will be **return**). Include metadata information too.


Vivado HLS Directive Editor

Type

Directive: RESOURCE

Destination

( ) Source File

(●) Directive File

Options

variable (required): return

core (optional): AXI4LiteS

port map (optional):

metadata (optional): -bus_bundle fir_io

Figure 17. Applying metadata to assign function control signals to AXI4Lite adapter

Note that the above steps 8-1-3 through 8-1-8 will create address maps for x, y, ap_start ap_valid, ap_done, and ap_idle, which can be accessed via software. Alternately, ap_start, ap_valid, ap_done, ap_idle signals can be generated as separate ports on the core by not applying RESOURCE directive to the top-level module fir. These ports will then have to be connected in a processor system using available GPIO IP.

Generate the IP-XACT Adapter Step 9

11-1. Re-synthesize the design as directives have been added. Run the RTL Export to generate the IP-XACT adapter.

11-1-1. Since the directives have been added, it is safe to re-synthesize the design. Select **Solution > Run C Synthesis > Active Solution.**

11-1-2. Once the design is synthesized, select **Solution > Export RTL** to open the dialog box so the desired IP can be generated.

An Export RTL Dialog box will open.

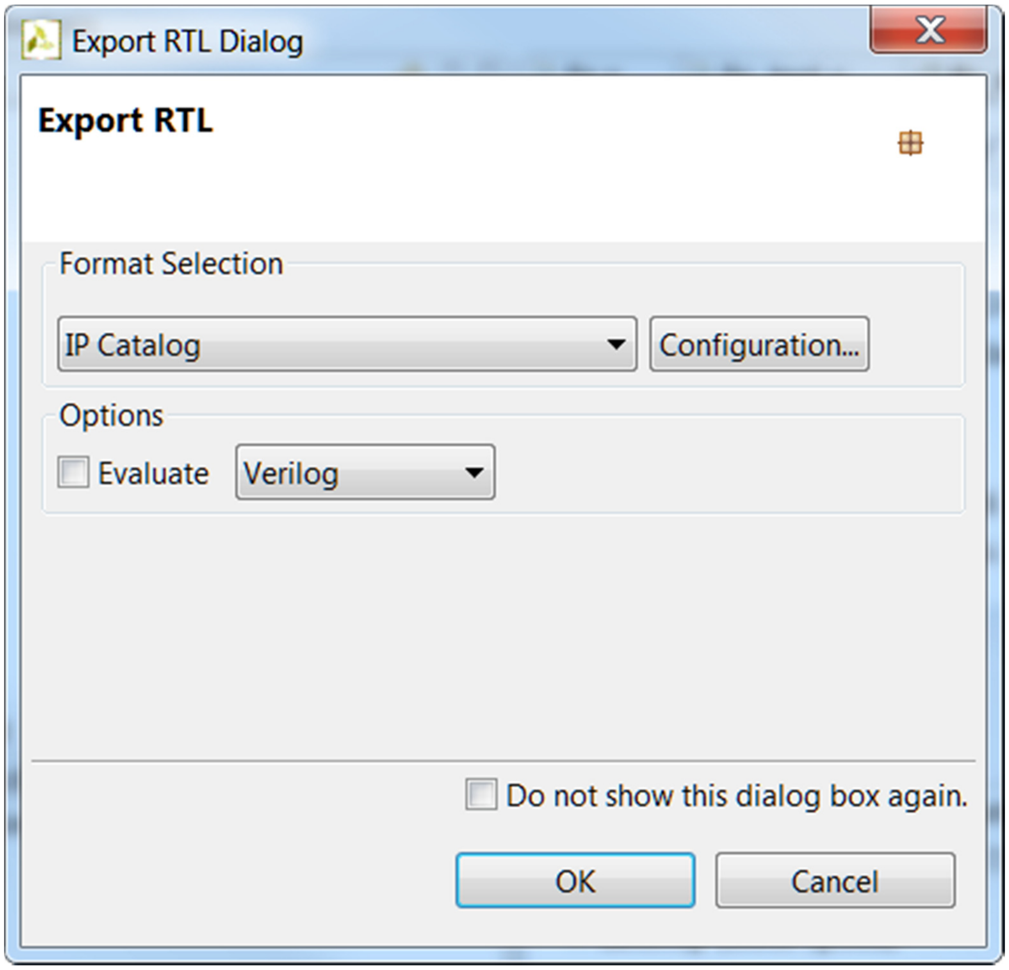


Figure 18. Export RTL Dialog

11-1-3. Click **OK** to generate the IP-XACT adapter.

11-1-4. When the run is completed, expand the **impl** folder in the *Explorer* view and observe various generated directories; ip, verilog and vhdl.

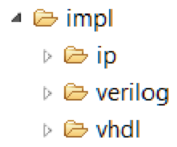

```
impl
  ip
  verilog
  vhdl
```

Figure 19. IP-XACT adapter generated

Expand the *ip* directory and observe several files and sub-directories. One of the sub-directory of interest is the *drivers* directory which consists of header, c, tcl, mdd, and makefile files. Another file of interest is the *zip* file, which is the ip repository file that can be imported in an IP Integrator design

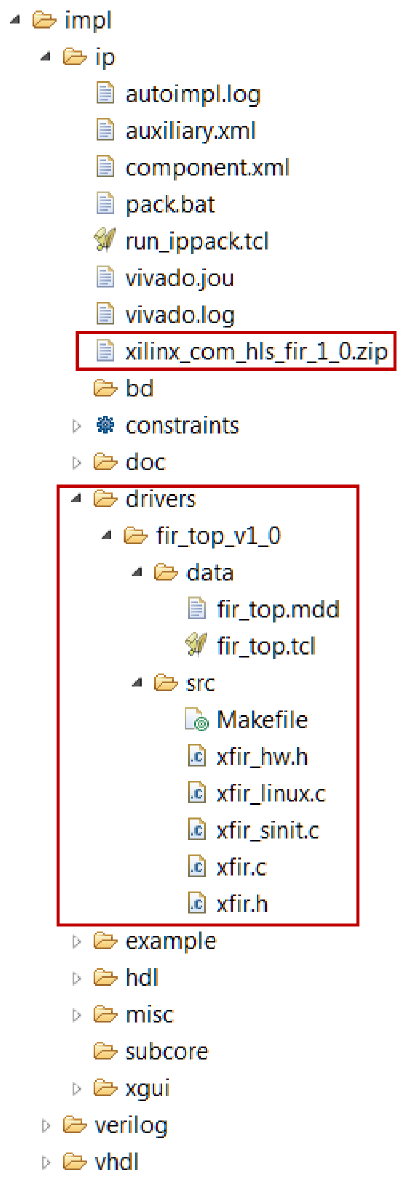

```
impl
  ip
    autoimpl.log
    auxiliary.xml
    component.xml
    pack.bat
    run_ippack.tcl
    vivado.jou
    vivado.log
    xilinx_com_hls_fir_1_0.zip
    bd
    constraints
    doc
    drivers
      fir_top_v1_0
        data
          fir_top.mdd
          fir_top.tcl
        src
          Makefile
          xfir_hw.h
          xfir_linux.c
          xfir_sinit.c
          xfir.c
          xfir.h
    example
    hdl
    misc
    subcore
    xgui
  verilog
  vhdl
```

Figure 20. Adapter's drivers directory

11-1-5. Close Vivado HLS by selecting **File > Exit.**

Update the Vivado Project with the FIR IP Step 10

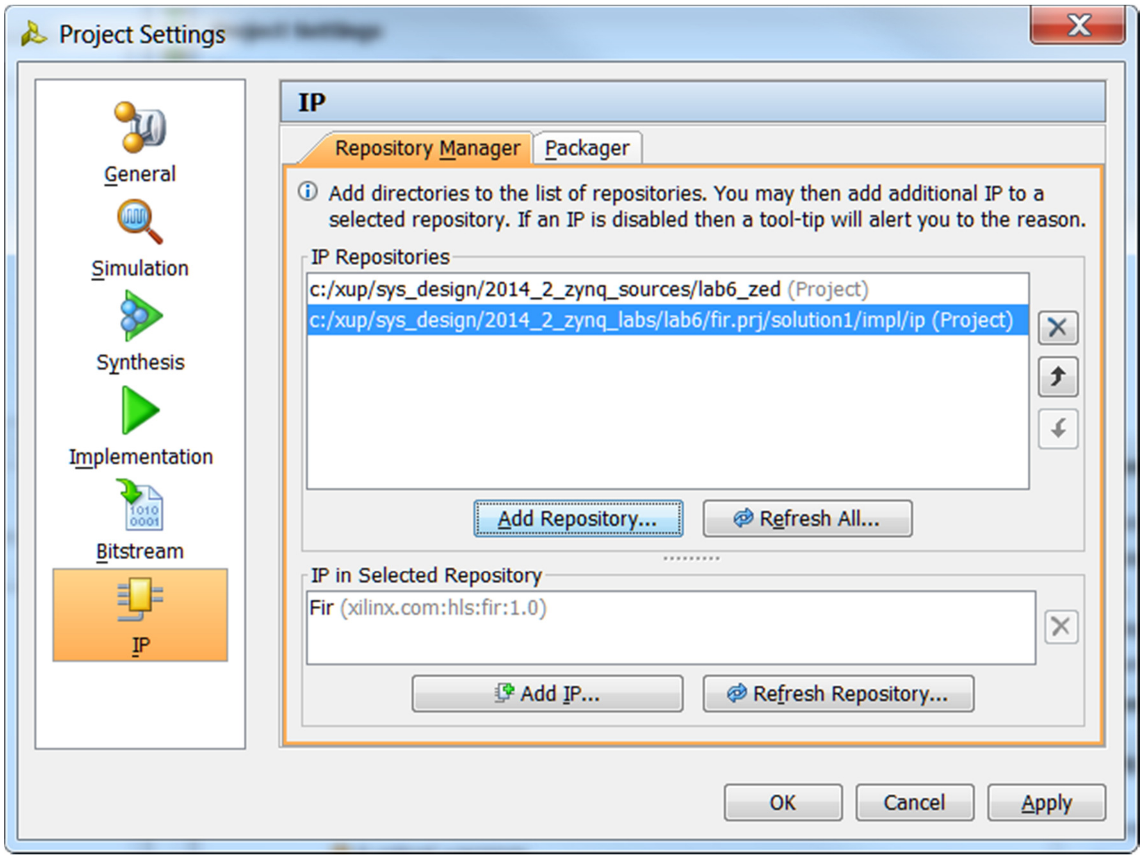
12-1. In Vivado, set the HLS IP to the IP Catalog settings.

12-1-1. In the *Flow Navigator* pane, click **Project Settings** under *Project Manager.*

12-1-2. Click the **IP** icon.

12-1-3. Click the **Add Repository…** button. Browse to **<2014_2_zynq_labs>\fir.prj\solution1\impl\ip** and click **Select**.

The directory will be scanned and added in the *IP Repositories* window, and *Fir* IP entry will be displayed in the *IP in Selected Repository* window.



IP

Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

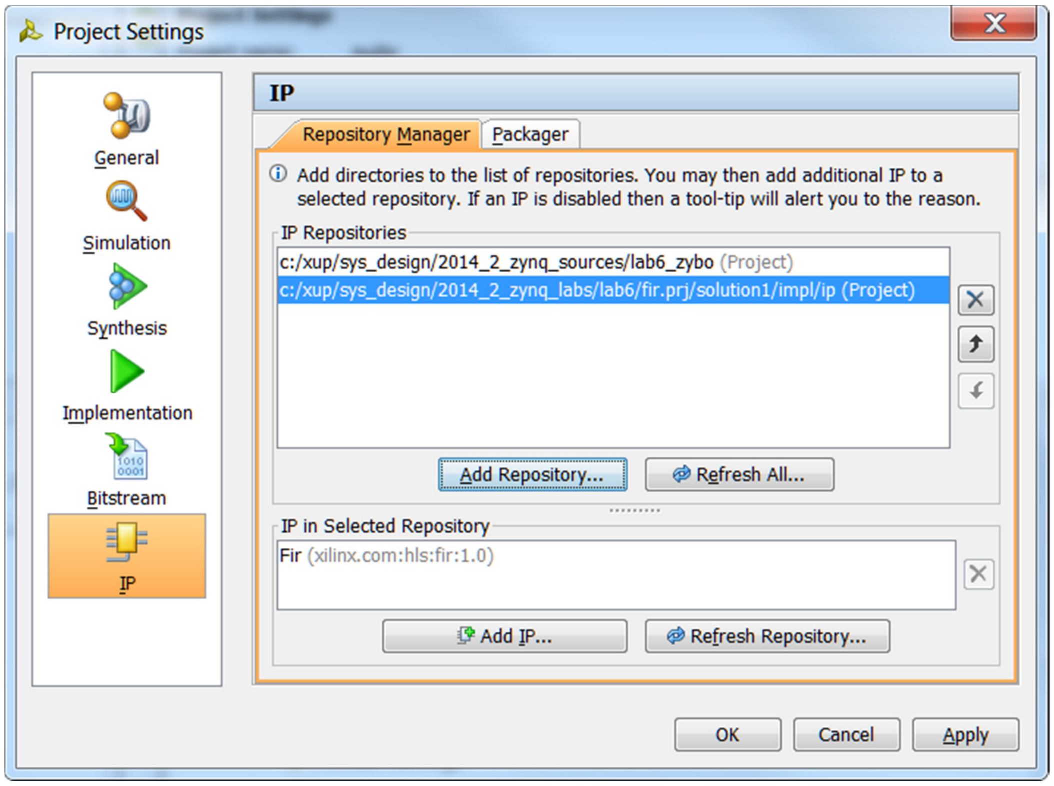
IP Repositories

c:/xup/sys_design/2014_2_zynq_sources/lab6_zed (Project)
c:/xup/sys_design/2014_2_zynq_labs/lab6/fir.prj/solution1/impl/ip (Project)

IP in Selected Repository

Fir (xilinx.com:hls:fir:1.0)

Figure 21. Setting path to IP Repositories for ZedBoard



IP

Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

IP Repositories

c:/xup/sys_design/2014_2_zynq_sources/lab6_zybo (Project)
c:/xup/sys_design/2014_2_zynq_labs/lab6/fir.prj/solution1/impl/ip (Project)

IP in Selected Repository

Fir (xilinx.com:hls:fir:1.0)

Figure 21. Setting path to IP Repositories for Zybo

12-1-4. Click **OK** to accept the settings, and **No,** if prompted to create a new synthesis run.

12-2. Open the block design. Instantiate fir_top core twice, one for each side channel, into the processing system naming the instances as fir_left and fir_right.

12-2-1. Select **Open Block Design > system.bd** from the *Flow Navigator* pane.

12-2-2. Click the Add IP icon and search for **FIR** in the catalog by typing **FIR** and double-click on the *FIR* entry to add an instance.

12-2-3. Click on the **Add IP to Block Design** button if presented.

Notice that the added IP has HLS logo in it indicating that this was created by Vivado HLS.

12-2-4. Select the added instance in the diagram, and change its instance name to **fir_left** by typing it in the *Name* field of the *Block Properties* form in the left.

12-2-5. Similarly, add another instance of the HLS IP, and name it **fir_right**.

12-2-6. Click on **Run Connection Automation**, and select /**fir_left**/**S_AXI_FIR_IO** and click **OK.**

12-2-7. Similarly, click on **Run Connection Automation** again, and select /**fir_right**/**S_AXI_FIR_IO** and click **OK.**

At this s e the design should look like shown below (you may have to click the regenerate button [ ]).Note that the interrupt signals for the *fir* IP blocks are not required and will remain unconnected.

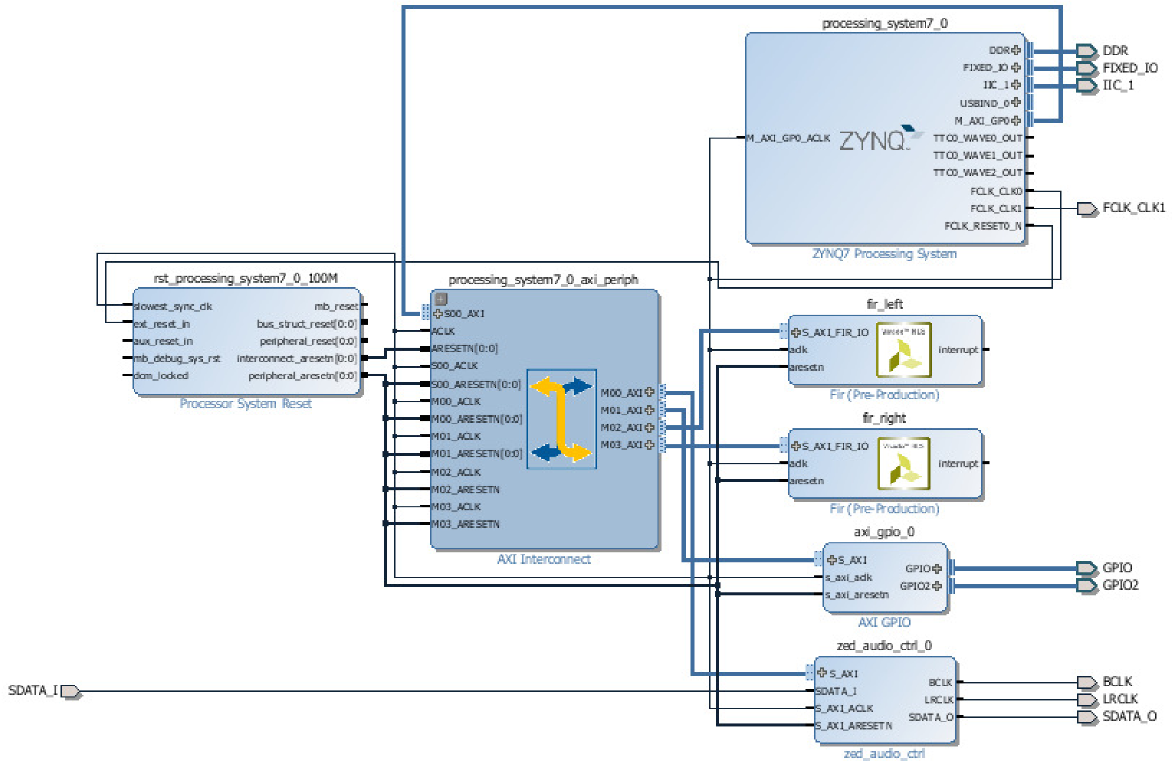


Figure 22. The complete hardware design for ZedeBoard

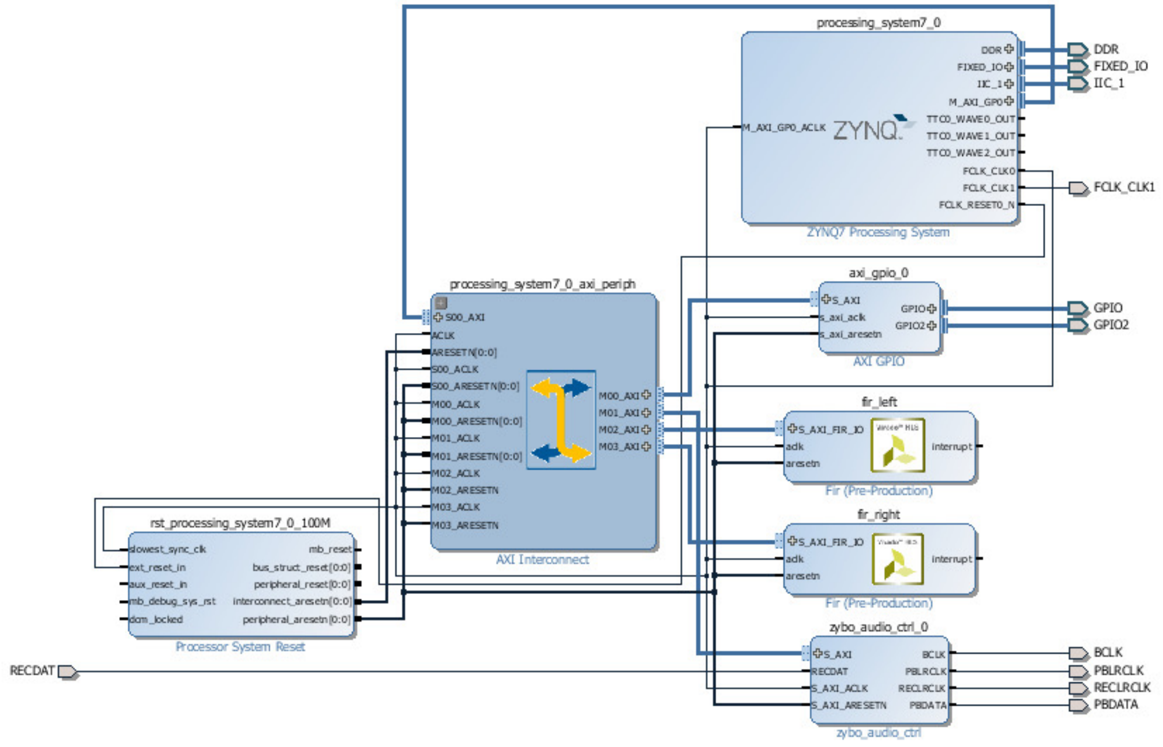
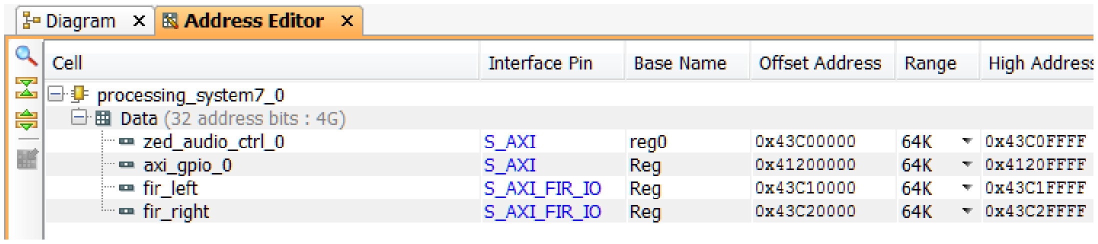


Figure 22. The complete hardware design for Zybo

12-3. Verify addresses and validate the design. Generate the system_wrapper file.

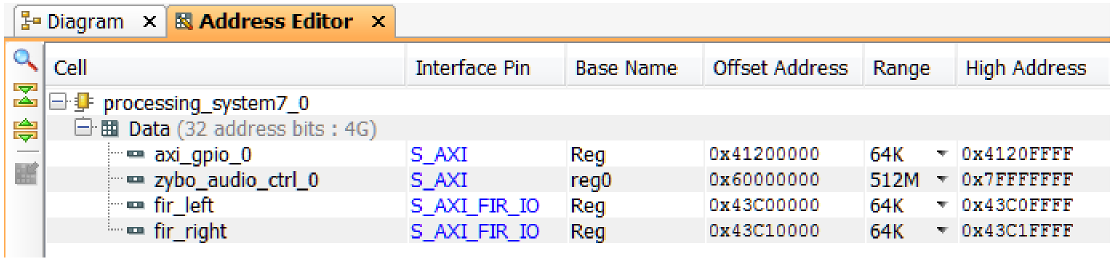
12-3-1. Click on the *Address Editor*, and expand the **processing_system7_0 > Data** if necessary.

The generated address map should look like as shown below.



Cell	Interface Pin	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 4G)					
zed_audio_ctrl_0	S_AXI	reg0	0x43C00000	64K	0x43C0FFFF
axi_gpio_0	S_AXI	Reg	0x41200000	64K	0x4120FFFF
fir_left	S_AXI_FIR_IO	Reg	0x43C10000	64K	0x43C1FFFF
fir_right	S_AXI_FIR_IO	Reg	0x43C20000	64K	0x43C2FFFF

Figure 23. Generated address map for ZedBoard



Cell	Interface Pin	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 4G)					
axi_gpio_0	S_AXI	Reg	0x41200000	64K	0x4120FFFF
zybo_audio_ctrl_0	S_AXI	reg0	0x60000000	512M	0x7FFFFFFF
fir_left	S_AXI_FIR_IO	Reg	0x43C00000	64K	0x43C0FFFF
fir_right	S_AXI_FIR_IO	Reg	0x43C10000	64K	0x43C1FFFF

Figure 23. Generated address map for Zybo

12-3-2. Run *Design Validation* (**Tools > Validate Design**) and verify there are no errors

12-3-3. In the *sources* view, right-click on the block diagram file, **system.bd**, and select **Create HDL Wrapper** to update the HDL wrapper file. When prompted, click **OK** with the *Let Vivado manage wrapper and auto-update* option.

12-3-4. Click on the **Generate Bitstream** in the Flow Navigator to run the synthesis, implementation, and bitstream generation processes.

12-3-5. Click **Save** and **Yes** if prompted.

12-3-6. When the bit generation is completed, click **OK** to open the implemented design.

Export to SDK and Profile the Application with Hardware Step 11

13-1. Export the hardware along with the generated bitstream to SDK.

To Export the hardware, the block diagram must be open and the Implemented design must be open.

13-1-1. If it is not already open, click **Open Block Design** (under IP Integrator in the Flow Navigator)

13-1-2. If it is not already open, click **Open Implemented Design** (under Implementation)

13-1-3. If SDK is already open then skip the next two steps.

13-1-4. Select **File > Launch SDK**

13-1-5. Click **OK**.

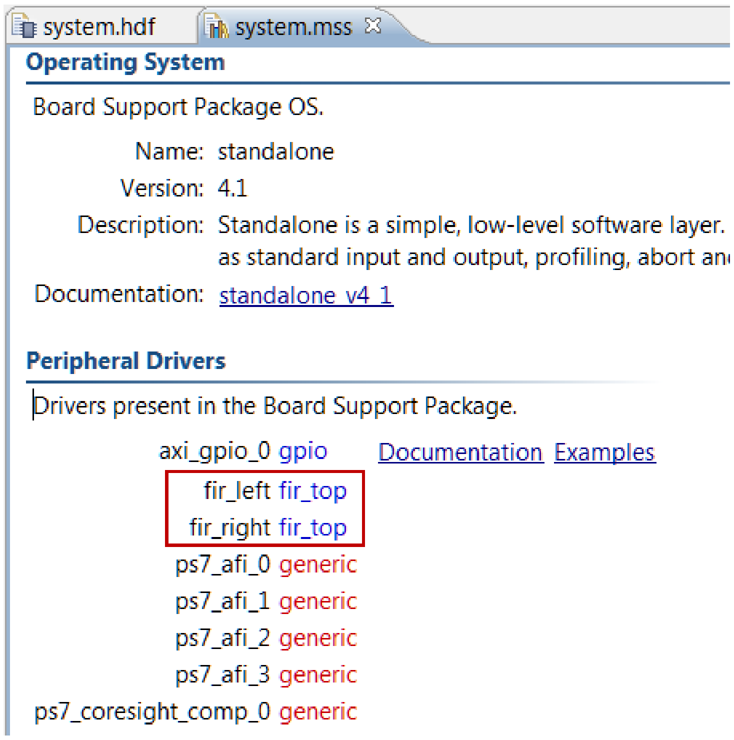
13-1-6. Select **File > Export > Export Hardware**

13-1-7. Make sure that **Include Bitstream** option is selected and click **OK**, leaving the target directory set to local project directory.

The SDK will detect the change in the hardware and bitstream and may pop-up a warning box asking you if it is OK to update.

13-1-8. Click **Yes.**

The BSP will be re-compiled and the system.mss tab will be regenerated showing fir_top driver assigned to fir_left and fir_right instances.



Operating System

Board Support Package OS.

Name:	standalone
Version:	4.1
Description:	Standalone is a simple, low-level software layer. as standard input and output, profiling, abort an
Documentation:	standalone_v4_1

Peripheral Drivers

Drivers present in the Board Support Package.

axi_gpio_0	gpio	Documentation Examples
fir_left	fir_top	
fir_right	fir_top	
ps7_afi_0	generic	
ps7_afi_1	generic	
ps7_afi_2	generic	
ps7_afi_3	generic	
ps7_coresight_comp_0	generic	

Figure 24. Updated mss file with driver assigned to the added IP

13-1-9. Right-click on the *standalone_bsp_0* and select **Re-generate BSP Sources**. Click **Yes** to re-generate the BSP.

13-2. Remove the user defined SW_PROFILE symbol and add the HW_PROFILE symbol.

13-2-1. Select the *lab6* application, right-click, and select **C/C++ Build Settings**.

13-2-2. Under the **ARM gcc compiler** group, select the **Symbols** sub-group**,** select **SW_PROFILE**, and delete it by clicking on the delete button.

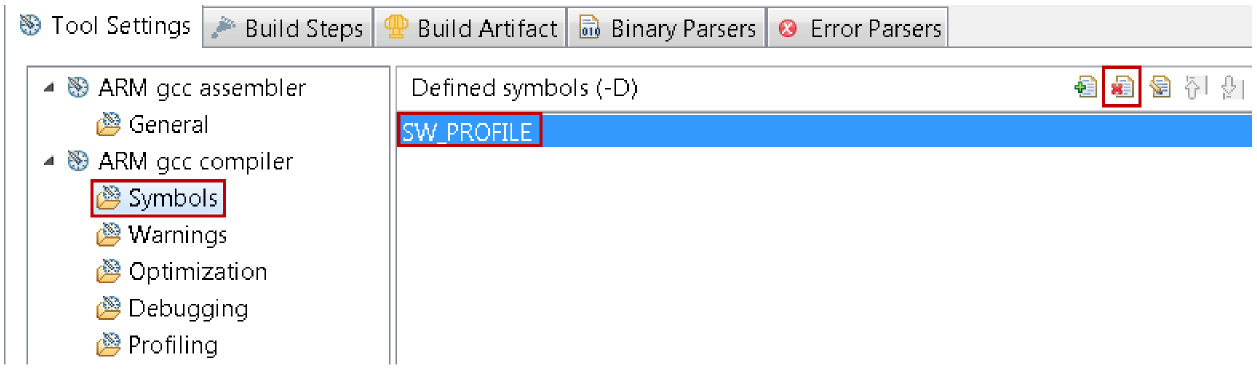


Figure 25. Deleting the user-defined symbol

13-2-3. Add the **HW_PROFILE** symbol and click **OK**.

This will allow us to profile the hardware IP of the FIR application. The program should compile without error.

13-3. Make sure that the SW[1] is in ON position. Power ON the board. Program the FPGA. Profile the application using the hardware FIR filter IP.

13-3-1. Make sure that the SW[1] is in the ON (up) position.

13-3-2. Power ON the board.

13-3-3. Select **Xilinx Tools** > **Program FPGA**

13-3-4. Click the **Program** button.

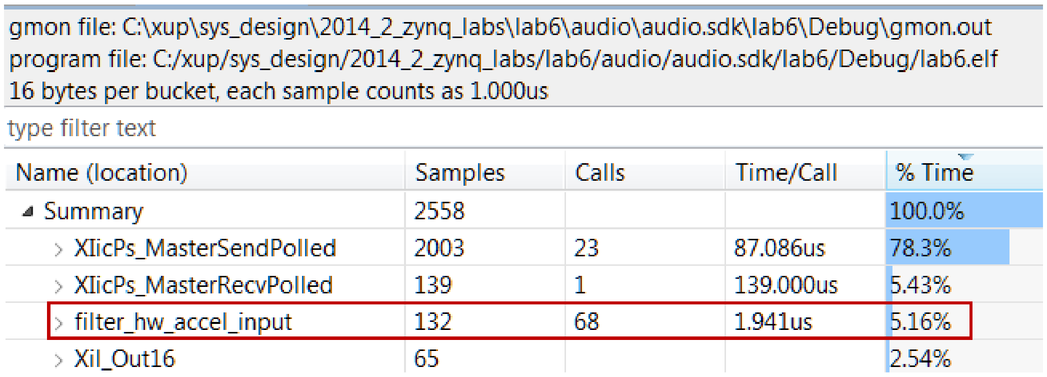
13-3-5. From the menu bar, select **Run > Run Configurations** and click the **Run** button to profile the application.

13-3-6. Click **OK** when prompted.

13-3-7. Invoke *gprof* by double-clicking gmon.out entry under *lab6* > *Debug* folder in the Project Explorer view of SDK, select the **Sorts samples per function** output, and sort the %Time column.

Notice that the output now shows filter_hw_accel_input function call instead of the fir_software function call. Note that the number of calls to the filter function has not changed but the average time spent per call is 1.941 us for ZedBoard or 1.779 us for Zybo as the filtering is done in the hardware instead of the software.

Also notice that the amount of time spent in the filtering function reduced from about 16.63% to 5.16% for ZedBoard or 12.45% to 3.5% for Zybo.

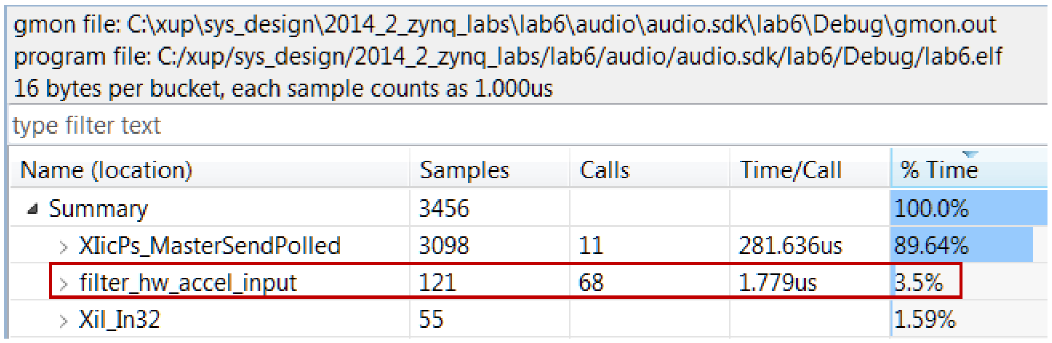


gmon file: C:\xup\sys_design\2014_2_zynq_labs\lab6\audio\audio.sdk\lab6\Debug\gmon.out
program file: C:/xup/sys_design/2014_2_zynq_labs/lab6/audio/audio.sdk/lab6/Debug/lab6.elf
16 bytes per bucket, each sample counts as 1.000us

type filter text

Name (location)	Samples	Calls	Time/Call	% Time
Summary	2558			100.0%
XIicPs_MasterSendPolled	2003	23	87.086us	78.3%
XIicPs_MasterRecvPolled	139	1	139.000us	5.43%
filter_hw_accel_input	132	68	1.941us	5.16%
Xil_Out16	65			2.54%

Figure 26. Profiling the application with the hardware IP for ZedBoard



gmon file: C:\xup\sys_design\2014_2_zynq_labs\lab6\audio\audio.sdk\lab6\Debug\gmon.out
program file: C:/xup/sys_design/2014_2_zynq_labs/lab6/audio/audio.sdk/lab6/Debug/lab6.elf
16 bytes per bucket, each sample counts as 1.000us

type filter text

Name (location)	Samples	Calls	Time/Call	% Time
Summary	3456			100.0%
XIicPs_MasterSendPolled	3098	11	281.636us	89.64%
filter_hw_accel_input	121	68	1.779us	3.5%
Xil_In32	55			1.59%

Figure 26. Profiling the application with the hardware IP for Zybo

Verify the Design in Hardware Step 12

14-1. Turn-OFF the profiling in the Board Support Package setting, lab6's C/C++ Build Settings, and Run Configuration settings.

14-1-1. Right-click on the *standalone_bsp_0* project in the Project Explorer and select **Board Support Package Settings**.

14-1-2. Select the **Overview > standalone** entry in the left pane, click on the drop-down arrow of the *enable_sw_intrusive_profiling Value* field and select **false**.

14-1-3. Select the **Overview > drivers > cpu_cortexa9** and remove **–pg** from the *extra_compiler_flags Value* field.

14-1-4. Click **OK** to accept the settings and update the BSP.

14-1-5. Right-click on the *lab6* project and select *C/C++ Build Settings*. Select the *Profiling* settings and **uncheck** the *Enable Profiling* check box.

14-1-6. Select *Run > Run Configurations*. Select the *Profiling* tab, and **uncheck** the *Enable Profiling* option.

14-1-7. Click **Apply** and **Close**.

14-2. Connect an audio patch cable between the Line In jack and the Speaker (header) out jack of a PC. Connect a headphone to the Line Out jack of the ZedBoard or HPH Out of the Zybo. Set the SW[1] in the OFF position. Play the provided corrupted_music_4kHz.wav file.

14-2-1. Connect an audio patch cable between the Line In jack and the Speaker (header) out jack of a PC

14-2-2. Connect a headphone to the Line Out jack on the ZedBoard or HPH Out on the Zybo.

14-2-3. Set the SW[1] in the OFF position.

14-2-4. Double-click **corrupted_music_4KHz.wav** or some other wave file of interest to play it using the installed media player. Place it in the continuous play mode.

14-2-5. Right-click on the *lab6* in the Project Explorer pane and select **Run As > Launch On Hardware (GDB)**.

The program will be downloaded and run. If you want to listen to corrupted signal then set the SW[0] OFF. To listened the filtered signal set the SW[0] ON.

14-2-6. When done, power OFF the board, and exit SDK and Vivado using **File > Exit**.

Conclusion

In this lab, you profiled a software application after creating a processor system using IP Integrator. Then you created a Vivado HLS project and added RESOURCE directive to create an IP-XACT adapter. You generated the IP-XACT adapter during the export phase. You then updated the processor system using the generated IP-XACT adapter, and profiled the system with the provided application.

Answers

1. Answer the following questions:

ZedBoard:

Estimated clock period:	7.95 ns
Worst case latency:	175 clock cycles
Number of DSP48E used:	3
Number of BRAMs used:	0
Number of FFs used:	138
Number of LUTs used:	393

Zybo:

Estimated clock period:	6.38 ns
Worst case latency:	175 clock cycles
Number of DSP48E used:	3
Number of BRAMs used:	0
Number of FFs used:	163
Number of LUTs used:	396

Appendix

Create a Project using Vivado GUI Step 13

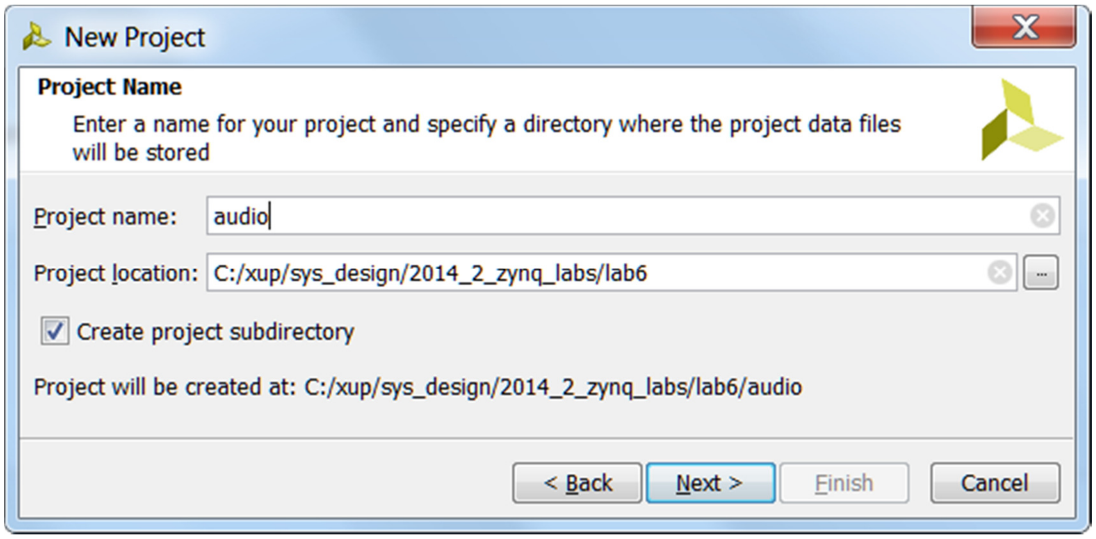
15-1. Launch Vivado and create an empty project either targeting the ZedBoard (having xc7z020clg484-1 device) or the Zybo (having xc7z010clg400-1 device) and using the Verilog language.

15-1-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2014.2 > Vivado 2014.2**

15-1-2. Click **Create New Project** to start the wizard. You will see the *Create a New Vivado Project* dialog box. Click **Next**.

15-1-3. Click the Browse button of the *Project Location* field of the **New Project** form, browse to **<2014_2_zynq_labs>\lab6**, and click **Select**.

15-1-4. Enter **audio** in the *Project Name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.



Enter a name for your project and specify a directory where the project data files will be stored

Project name: audio

Project location: C:/xup/sys_design/2014_2_zynq_labs/lab6

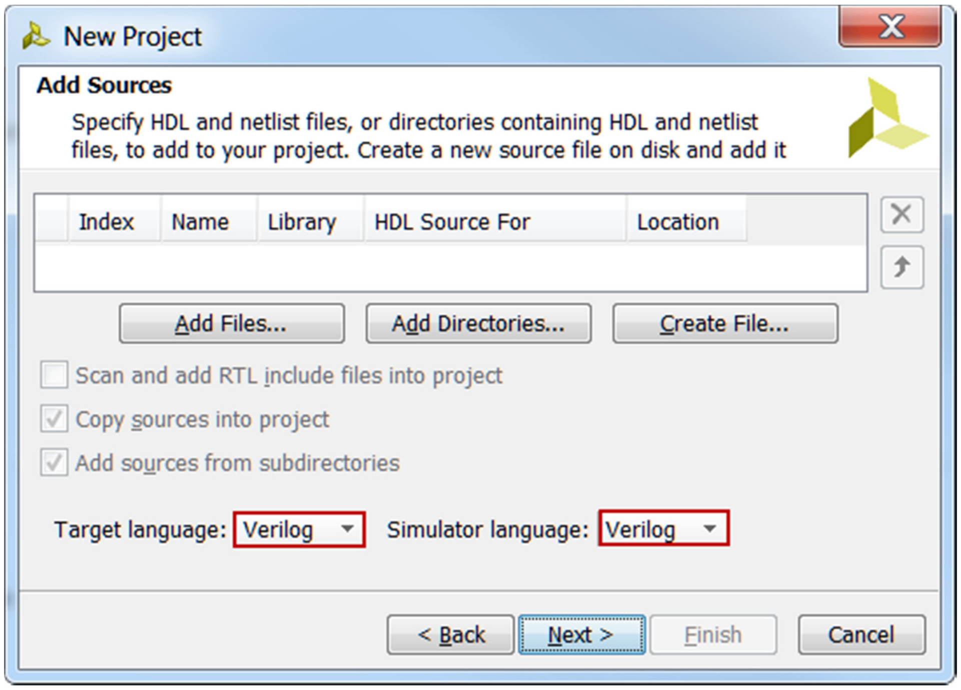
✓ Create project subdirectory

Project will be created at: C:/xup/sys_design/2014_2_zynq_labs/lab6/audio

Figure A-1. Project Name entry

15-1-5. Select **RTL Project** in the *Project Type* form, and click **Next**.

15-1-6. Select **Verilog** as the *Target language* and *Simulator Language* in the *Add Sources* form, and click **Next**.



New Project

Add Sources

Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it

Index	Name	Library	HDL Source For	Location

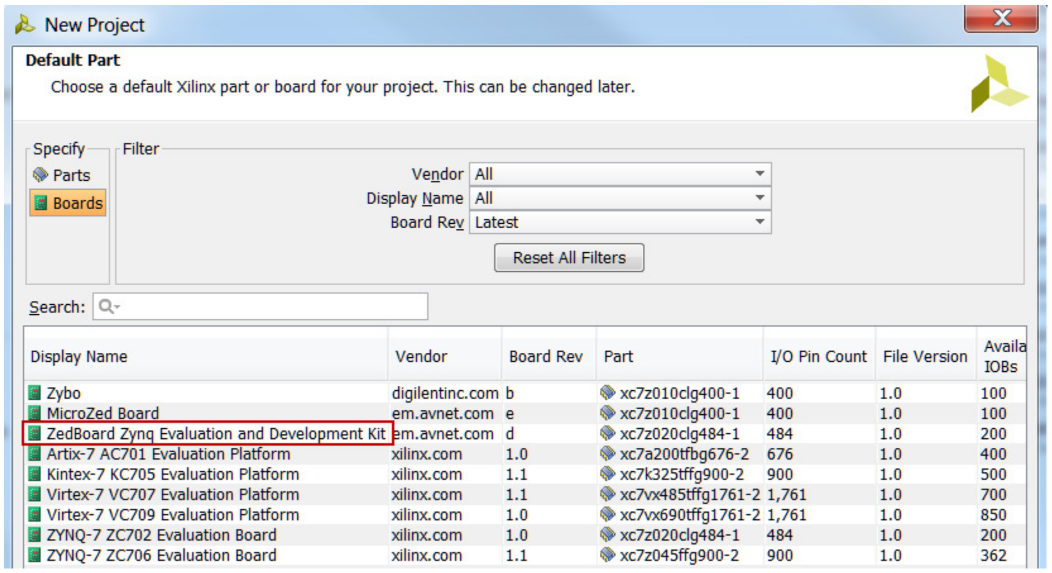
- [ ] Scan and add RTL include files into project
- [x] Copy sources into project
- [x] Add sources from subdirectories

Target language: Verilog Simulator language: Verilog

Figure A-2. Add sources to new project

15-1-7. Click **Next** <u>two times</u> to skip *Adding Existing IP* and *Add Constraints* dialog boxes

15-1-8. In the *Default Part* form, select *Boards*, and select **ZedBoard** or **Zybo**. Click **Next**.



New Project

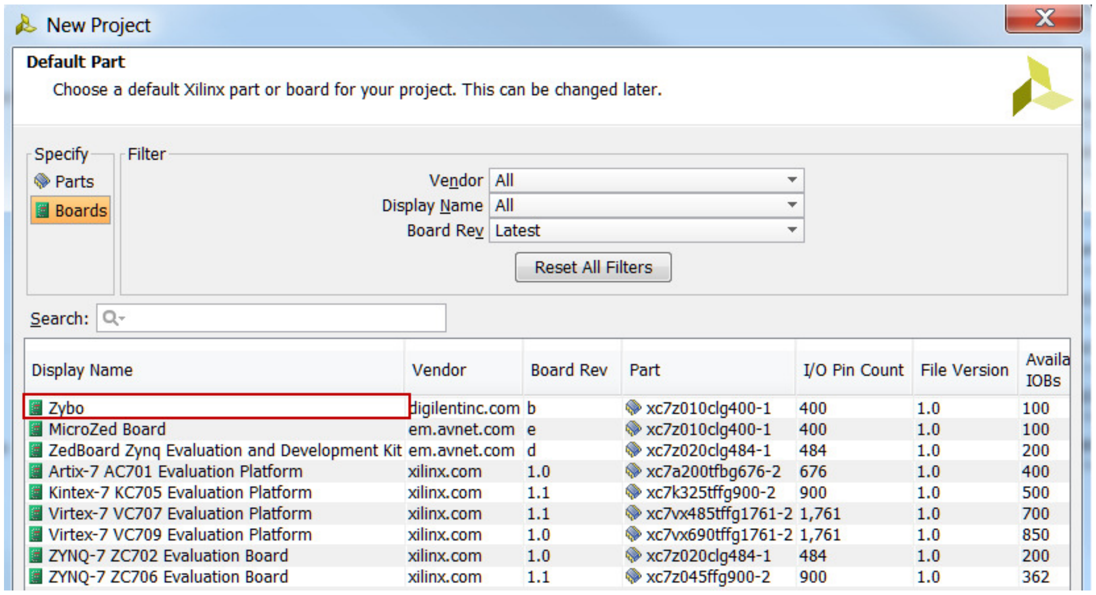
Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Vendor: All
Display Name: All
Board Rev: Latest

Display Name	Vendor	Board Rev	Part	I/O Pin Count	File Version	Availa IOBs
Zybo	digilentinc.com	b	xc7z010clg400-1	400	1.0	100
MicroZed Board	em.avnet.com	e	xc7z010clg400-1	400	1.0	100
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484	1.0	200
Artix-7 AC701 Evaluation Platform	xilinx.com	1.0	xc7a200tfbg676-2	676	1.0	400
Kintex-7 KC705 Evaluation Platform	xilinx.com	1.1	xc7k325tffg900-2	900	1.0	500
Virtex-7 VC707 Evaluation Platform	xilinx.com	1.1	xc7vx485tffg1761-2	1,761	1.0	700
Virtex-7 VC709 Evaluation Platform	xilinx.com	1.0	xc7vx690tffg1761-2	1,761	1.0	850
ZYNQ-7 ZC702 Evaluation Board	xilinx.com	1.0	xc7z020clg484-1	484	1.0	200
ZYNQ-7 ZC706 Evaluation Board	xilinx.com	1.1	xc7z045ffg900-2	900	1.0	362

Figure A-3. Selecting the ZedBoard as the target board


New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Specify: Parts, Boards

Filter: Vendor: All; Display Name: All; Board Rev: Latest; Reset All Filters

Search:

Display Name	Vendor	Board Rev	Part	I/O Pin Count	File Version	Availa IOBs
Zybo	digilentinc.com	b	xc7z010clg400-1	400	1.0	100
MicroZed Board	em.avnet.com	e	xc7z010clg400-1	400	1.0	100
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484	1.0	200
Artix-7 AC701 Evaluation Platform	xilinx.com	1.0	xc7a200tfbg676-2	676	1.0	400
Kintex-7 KC705 Evaluation Platform	xilinx.com	1.1	xc7k325tffg900-2	900	1.0	500
Virtex-7 VC707 Evaluation Platform	xilinx.com	1.1	xc7vx485tffg1761-2	1,761	1.0	700
Virtex-7 VC709 Evaluation Platform	xilinx.com	1.0	xc7vx690tffg1761-2	1,761	1.0	850
ZYNQ-7 ZC702 Evaluation Board	xilinx.com	1.0	xc7z020clg484-1	484	1.0	200
ZYNQ-7 ZC706 Evaluation Board	xilinx.com	1.1	xc7z045ffg900-2	900	1.0	362

Figure A-3. Selecting the Zybo as the target board

15-1-9. Check the *Project Summary* and click **Finish** to create an empty Vivado project.

Creating the System Using the IP Integrator Step 14

16-1. Use the IP Integrator to create a new Block Design, and generate the ARM Cortex-A9 processor based hardware system.

16-1-1. In the Flow Navigator, click **Create Block Design** under IP Integrator

16-1-2. Enter **system** for the design name and click **OK**

16-1-3. IP from the catalog can be added in different ways. Click on *Add IP* in the message at the top of the *Diagram* panel, or click the *Add IP icon* in the block diagram side bar, press Ctrl + I, or right-click anywhere in the Diagram workspace and select Add IP.

16-1-4. Once the IP Catalog is open, type "zy" into the Search bar, find and double click on **ZYNQ7 Processing System** entry, or click on the entry and hit the Enter key to add it to the design.

The Zynq block will be added.

16-1-5. Notice the message at the top of the Diagram window that Designer Assistance available. Click on **Run Block Automation** and select /**processing_system7_0**

16-1-6. Click **OK** when prompted to run automation with the default settings.

Notice that external ports have been automatically added for the DDR and Fixed IO once Block Automation has been complete, Some of the other default ports are also added to the block.

16-1-7. In the block diagram, double click on the *Zynq* block to open the *Customization* window for the Zynq processing system.

A block diagram of the Zynq should now be open, showing various configurable blocks of the Processing System.

At this stage, the designer can click on various configurable blocks (highlighted in green) and change the system configuration.

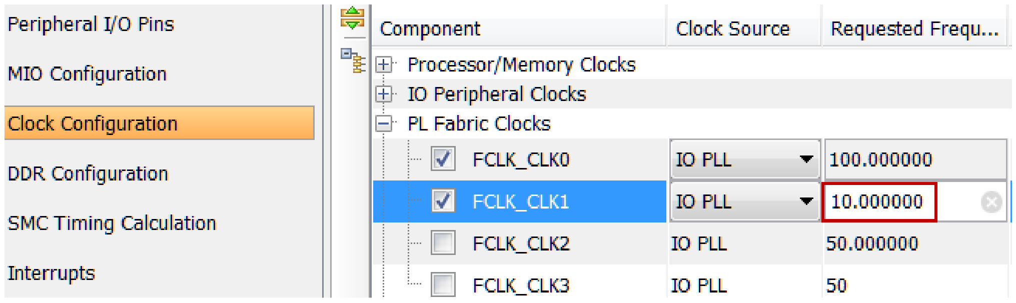
16-2. Configure I/O Peripherals block to use UART 1 and I2C 1 peripherals, disabling other unwanted peripherals. Enable FCLK_CLK1, the PL fabric clock and set its frequency either to 10.000 MHz for the ZedBoard or to 12.288 MHz for the Zybo.

16-2-1. Select the *MIO Configuration* tab on the left to open the configuration form and expand *I/O Peripheral* in the right pane.

16-2-2. Click on the check box of the *I2C 1* peripheral. Uncheck *USB0, SD 0, ENET 0, GPIO > GPIO MIO* as we don't need them.

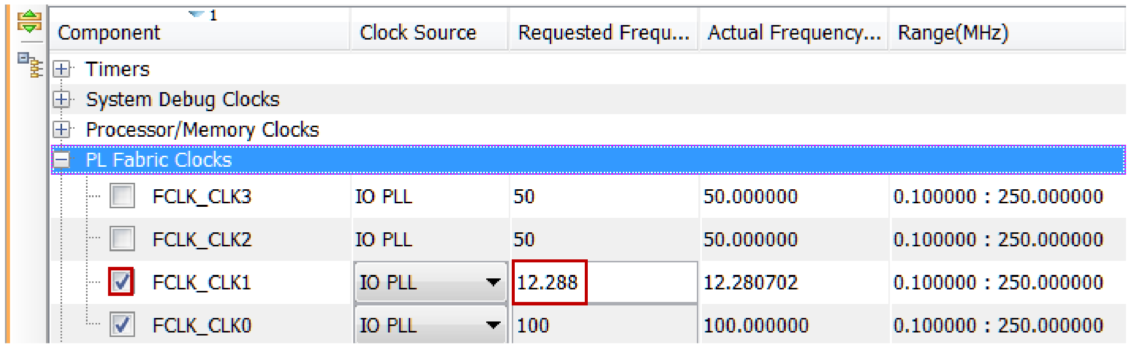
16-2-3. Select the *Clock Configuration* in the left pane, expand *the PL Fabric Clocks* entry in the right, and click the check-box of *FCLK_CLK1*.

16-2-4. Change the *Requested Frequency* value of *FCLK_CLK1* to **10.000** MHz for the ZedBoard or **12.288** MHz for the Zybo.


Peripheral I/O Pins
MIO Configuration
Clock Configuration
DDR Configuration
SMC Timing Calculation
Interrupts

Component	Clock Source	Requested Frequ...
Processor/Memory Clocks		
IO Peripheral Clocks		
PL Fabric Clocks		
✓ FCLK_CLK0	IO PLL	100.000000
✓ FCLK_CLK1	IO PLL	10.000000
FCLK_CLK2	IO PLL	50.000000
FCLK_CLK3	IO PLL	50

Figure A-4. Enabling and setting the frequency of FCLK_CLK1 for the ZedBoard



Component	Clock Source	Requested Frequ...	Actual Frequency...	Range(MHz)
Timers				
System Debug Clocks				
Processor/Memory Clocks				
PL Fabric Clocks				
FCLK_CLK3	IO PLL	50	50.000000	0.100000 : 250.000000
FCLK_CLK2	IO PLL	50	50.000000	0.100000 : 250.000000
✓ FCLK_CLK1	IO PLL	12.288	12.280702	0.100000 : 250.000000
✓ FCLK_CLK0	IO PLL	100	100.000000	0.100000 : 250.000000

Figure A-4. Enabling and setting the frequency of FCLK_CLK1 for the Zybo

16-2-5. Click **OK**.

Notice that the Zynq block only shows the necessary ports.

16-3. Add the provided I2C-based either zed_audio_ctrl IP for the ZedBoard or zybo_audio_ctrl IP for the Zybo to the IP Catalog

16-3-1. In the *Flow Navigator* pane, click **IP Catalog** under *Project Manager.*

The IP Catalog will open.

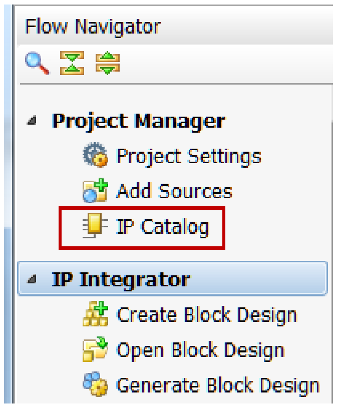


Figure A-5. Invoking IP Catalog

16-3-2. Click on the *IP Settings* button in the IP Catalog.

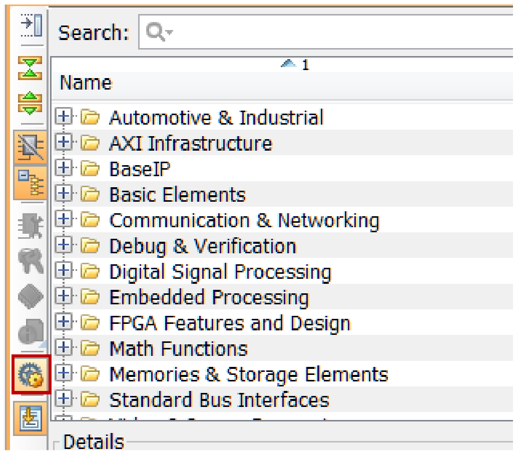


Figure A-6. Invoking IP Settings

16-3-3. Click on the **Add Repository…** button. Browse to **<2014_2_zynq_sources>\lab6_zed for the ZedBoard** or **<2014_2_zynq_sources>\lab6_zybo for the Zybo** directory, and click **Select**.

Notice that either the *zed_audio_ctrl* or the *zybo_audio_ctrl* entry is displayed in the *IP in Selected Repository* field.

16-3-4. Click **OK** to accept the settings.

16-4. ZedBoard: Instantiate zed_audio_ctrl and GPIO with width of 2 bits on channel 1 and width of 2 bits on channel 2.

Zybo: Instantiate zybo_audio_ctrl and GPIO with width of 1 bit output only on channel 1 and width of 2 bits input only on channel 2.

Run connection automation to connect them.

16-4-1. Click the Add IP button if the IP Catalog is not open and search for **AXI GPIO** in the catalog by typing **gpi** and double-click on the AXI GPIO entry to add an instance.

16-4-2. Click on the **Add IP to Block Design** button.

16-4-3. Double-click on the added instance and the **Re-Customize IP** GUI will be displayed.

16-4-4. Change the *Channel 1* width to **2** for the ZedBoard or width of **1 output only** for the Zybo.

16-4-5. Check the *Enable Dual Channel* box, set the width to **2** for the ZedBoard or width of **2 input only** for the Zybo, and click **OK**.

16-4-6. Similarly add an instance of either the *zed_audio_ctrl* for the ZedBoard or the the *zybo_audio_ctrl* IP for the Zybo.

16-4-7. Notice that *Design assistance* is available. Click on **Run Connection Automation**, and select /**axi_gpio_0/S_AXI**

16-4-8. Click **OK** to connect it to the M_AXI_GP0 interface.

Notice two additional blocks, *Proc Sys Reset*, and *AXI Interconnect* have automatically been added to the design.

16-4-9. Similarly, click on **Run Connection Automation**, and select either **/zed_audio_ctrl_0/S_AXI** for the ZedBoard or the **/zybo_audio_ctrl_0/S_AXI** for the Zybo and click **OK**.

16-5. Make IIC_1, GPIO, FCLK_CLK1, and either zed_audio_ctrl or zybo_audio_ctrl ports external.

16-5-1. Select the **GPIO** interface of the *axi_gpio_0* instance, right-click on it and select **Make External** to create an external port. This will create the external port named *GPIO* and connect it to the peripheral.

16-5-2. Select the **GPIO2** interface of the *axi_gpio_0* instance, right-click on it and select **Make External** to create the external port.

16-5-3. Similarly, selecting one port at a time either of the *zed_audio_ctrl_0* instance or the *zybo_audio_ctrl_0* instance, and make them external.

16-5-4. Similarly, make the **IIC_1** interface and **FCLK_CLK1** port of the *processing_system7_0* instance external.

At this stage the design should look like shown below (you may have to click the regenerate [ ] button).

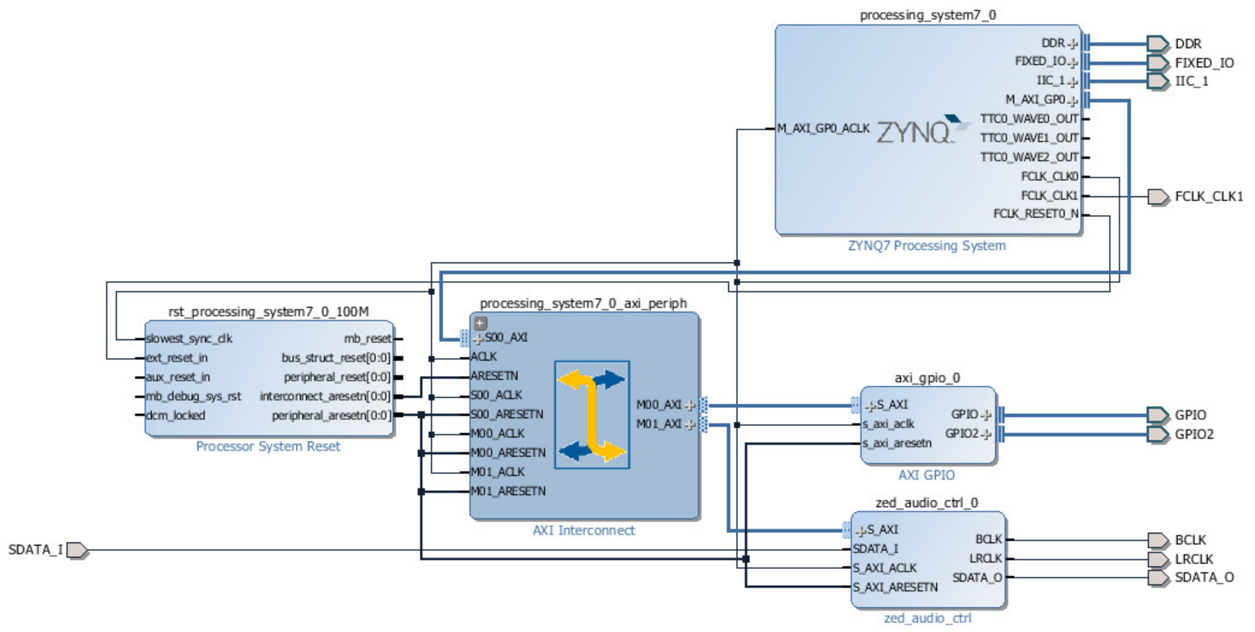


Figure A-7. Block design after I2C based zed_audio_ctrl core added and connections made for the ZedBoard

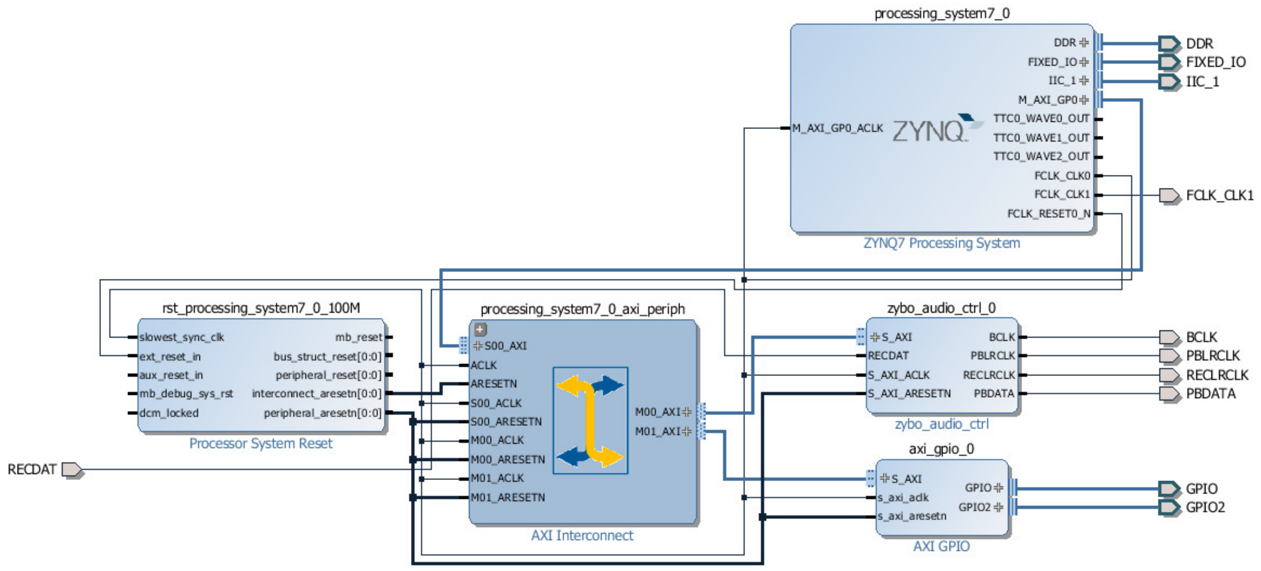


Figure A-7. Block design after I2C based zybo_audio_ctrl core added and connections made for the Zybo