Open IC

Open Audio Platform Tutorial on the Zybo Z7

A step-by-step guide on a basic audio platform on the Zybo Z7 FPGA device through Xilinx Vivado and SDK

Kevin Vaca

June 2019

Introduction

This tutorial is the open sourcing of the foundational material needed to create an audio-based project using the Zybo Z7-10 FPGA. The software used throughout the project include Xilinx's 2018.2.1 Vivado and Vivado SDK. This tutorial was made for those who seek to create FPGA projects with audio as a main focus. The tutorial is divided into two main parts, Xilinx Vivado and Vivado SDK, and a third minor part, SD Boot. The main parts each deal with the two main software used in establishing the project that the user will need to use. The third part is included as a supplemental piece in order to finalize the project and run it on its own. In the process of creating the sample project that is presented here, the author found many different sources to pool information from but still spent a lot of time to get everything together and in working order. There existed no easy, step-by-step tutorial for the basic platform of manipulating audio samples on the Zybo FPGA. Thus, as part of the Open-IC team, which seeks to create and sustain platforms for projects that use FPGAs, the author here has created this basic tutorial on building an audio platform on the Zybo Z7.

Though this tutorial is made for novices and beginners with little to no experience using Xilinx Vivado, this tutorial is not meant to teach the Xilinx software. This tutorial's focus is on preparing the audio platform on which the end user may create their own program to run on the Zybo Z7 FPGA. For a more thorough guide on the software itself, the author recommends the Zynq Book tutorials which can be found on their website:
http://www.zynqbook.com/download-tuts.html

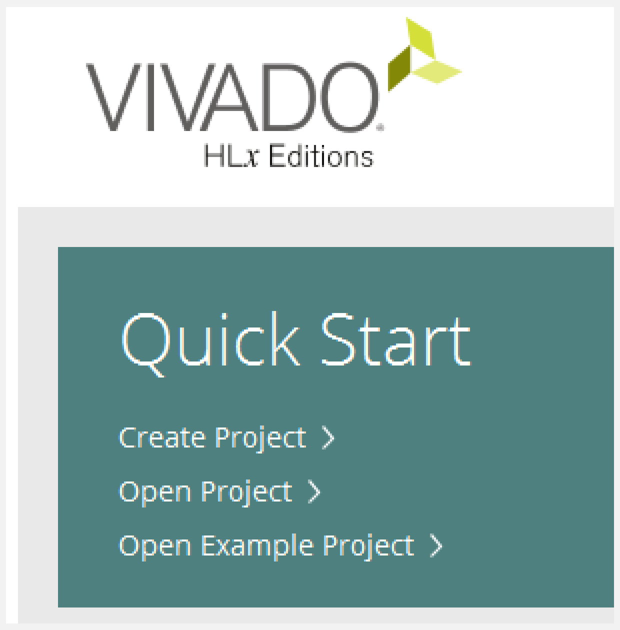
Dependencies

This tutorial has many file dependencies that the user must download onto their computer. The Zybo Audio Control IP used in this project comes from the Zynq Book project website linked previously. The Vivado software also requires Digilent's board files for the Zybo Z7-10 which can be found on their Github:
https://github.com/Digilent/vivado-boards/tree/master/new/board_files/zybo-z7-10/A.0
Pin designations require the use of Digilent's Master Constraint File also located on their Github:
https://github.com/Digilent/digilent-xdc/
Specifics on where to place these files are included in the tutorial, but having the files downloaded prior to beginning will help the user in following along with the tutorial. Of course, this being a tutorial for the Zybo Z7-10, the user should also have the FPGA: which can be bought at the Digilent online store. Other necessary hardware will include a microphone, and a musical instrument for testing the included project reference. The microphone the author used in this project was the PoP Voice Lavalier Lapel Microphone found on Amazon. The author also recommends an instrument that can play complex chords, like a guitar or piano, in order to properly test the chord detection algorithm included in the tutorial.

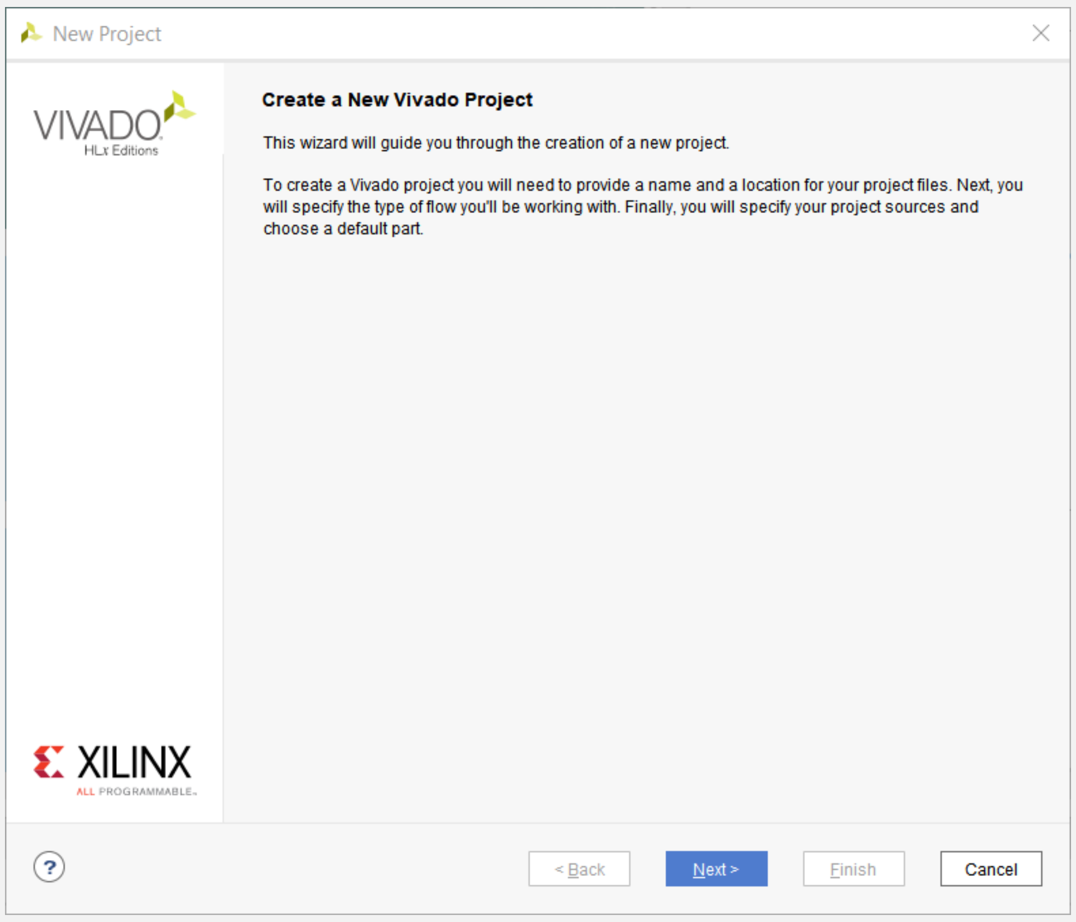
In closing, the author would like to thank the group that supported the full project, of which only the FPGA part is present here. The University of Houston Clear Lake 2019 Senior Project group consisted of Mitchell Jefferies, Rigo De Leon, and Harold Mao, as well as the author. The author would also like to thank Dr. Xiaokun Yang, the mentor for the senior project, as well as the head of the Open-IC project.

Xilinx Vivado

We begin this project by launching Xilinx Vivado 2018.2.
Under Quick Start, we will proceed by clicking on Create Project.



This will open the New Project window. Click on Next.

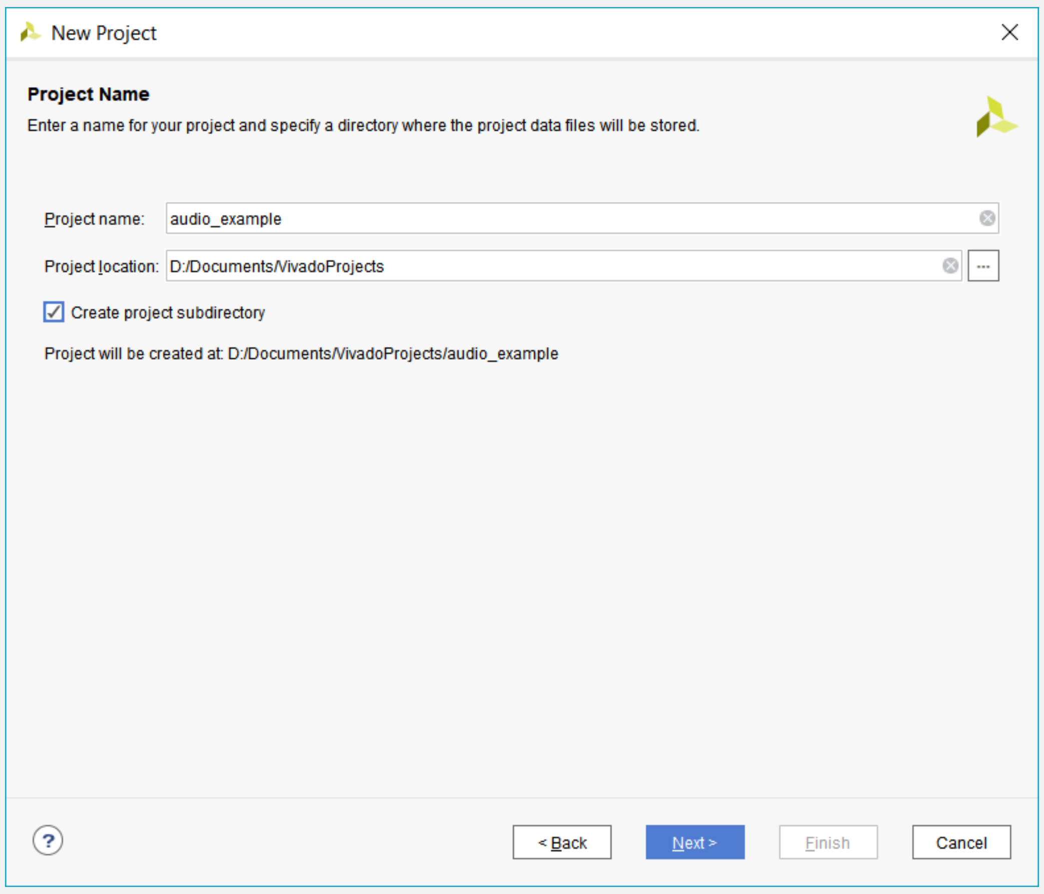

New Project

Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

For this tutorial, we will name the project, "audio example." Be sure to note where the project location is, and ensure that the "Create project subdirectory" option is checked. Click on Next.


New Project

Project Name

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

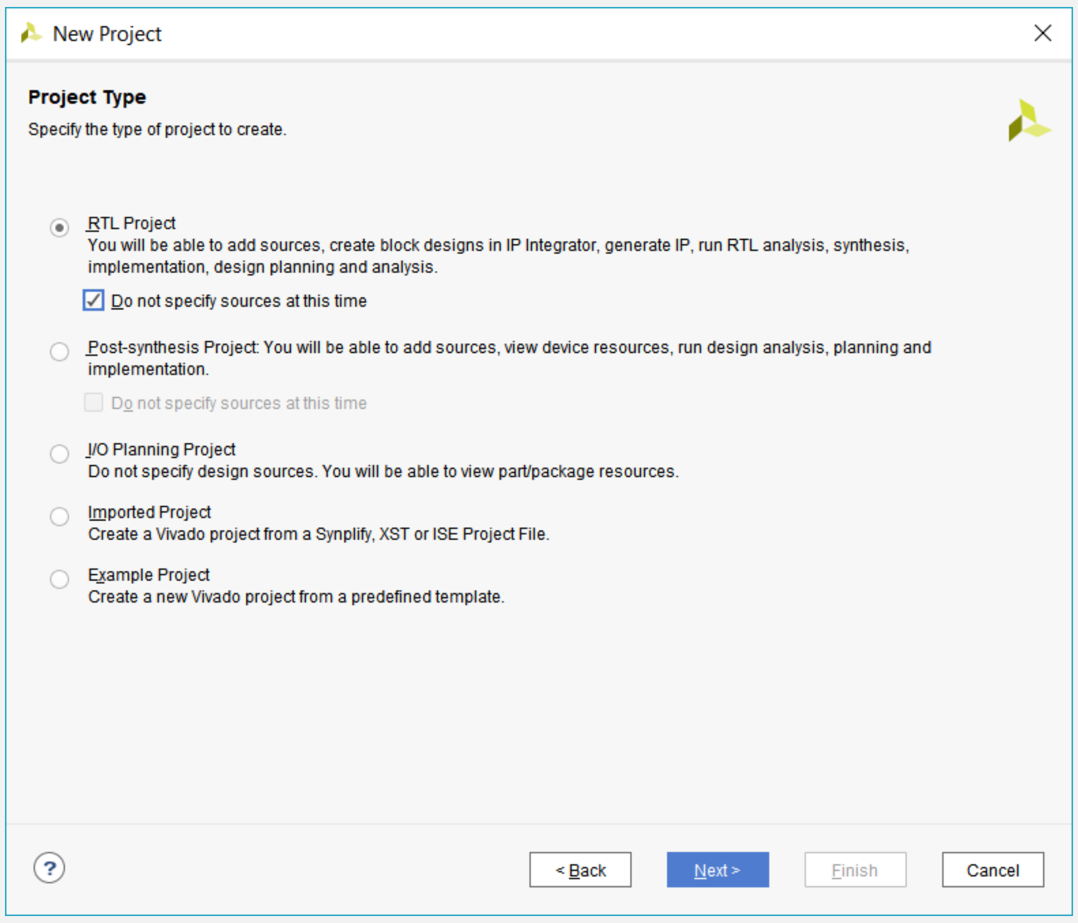
Project name: audio_example

Project location: D:/Documents/VivadoProjects

☑ Create project subdirectory

Project will be created at: D:/Documents/VivadoProjects/audio_example

The project type will be an RTL Project, and no sources will be specified at this time. Proceed with Next.


New Project

Project Type

Specify the type of project to create.

● RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☑ Do not specify sources at this time

○ Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

○ I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

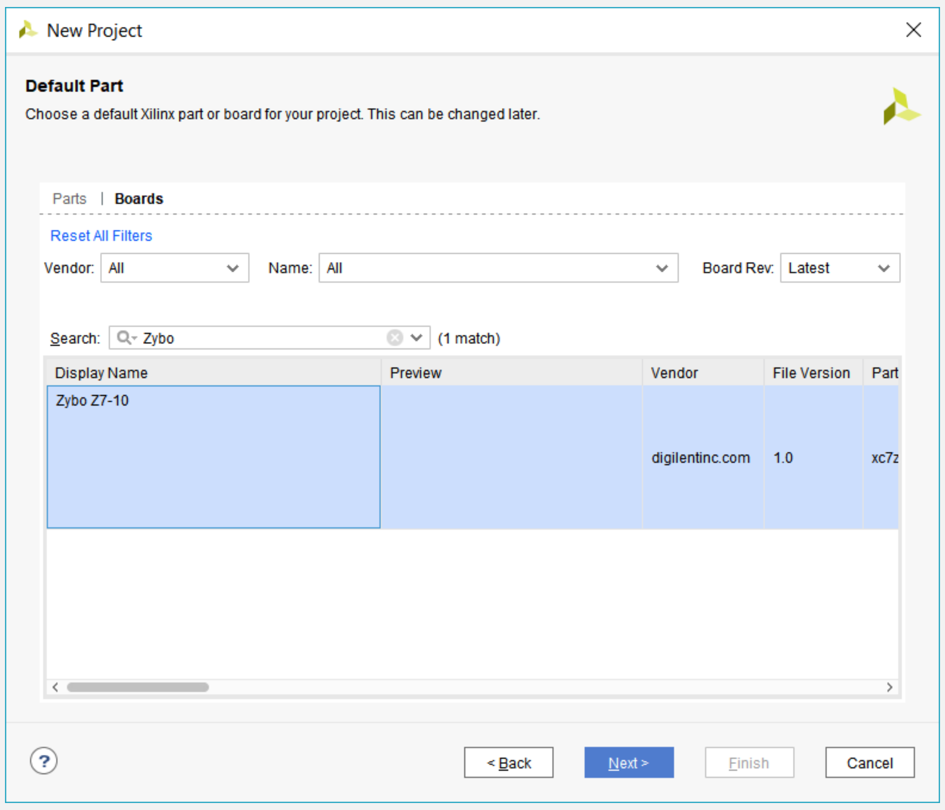
○ Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

○ Example Project
Create a new Vivado project from a predefined template.

The next step will require the board files for the Zybo Z7-10 FPGA.

NAVIGATE TO THE DIRECTORY WHERE XILINX VIVADO WAS INSTALLED. BY DEFAULT, IT INSTALLS DIRECTLY INTO THE C DRIVE. THE "ZYBO-Z7-10" FOLDER WHICH CONTAINS THE BOARD FILES FOR THE ZYBO FPGA SHOULD BE COPIED INTO: "\XILINX\VIVADO\2018.2\DATA\BOARDS\BOARD_FILES"

On the window in Vivado, select the “Boards” tab, and search for the Zybo-Z7-10 board files you have downloaded. Proceed with Next.


New Project

Default Part

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

Parts | **Boards**

Reset All Filters

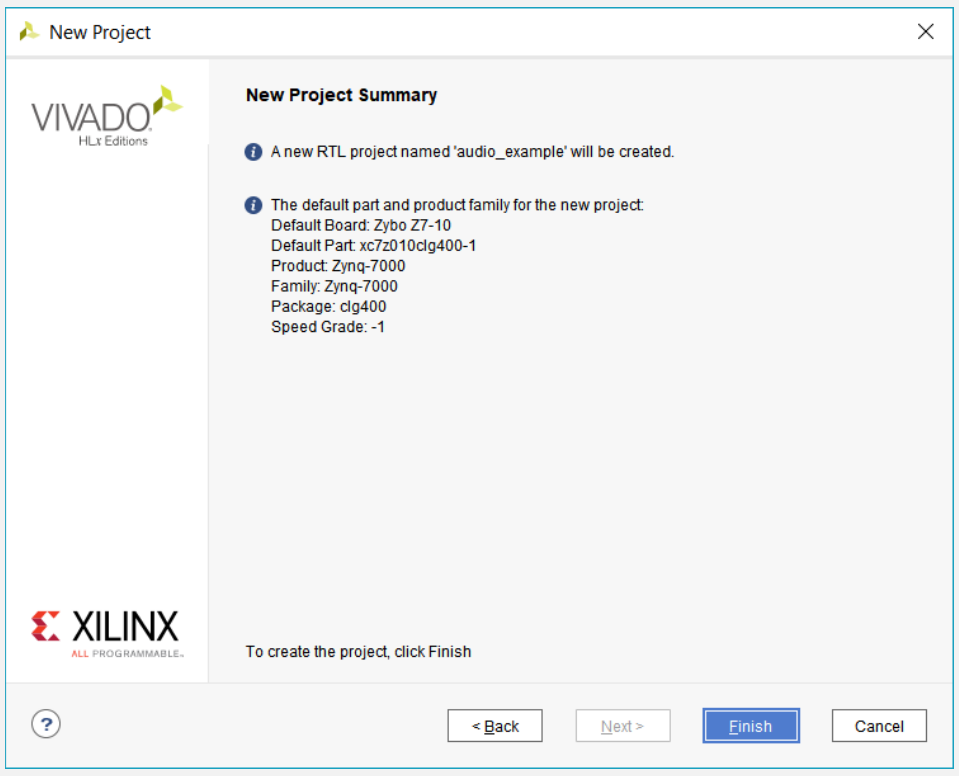
Vendor: All | Name: All | Board Rev: Latest

Search: Zybo (1 match)

Display Name	Preview	Vendor	File Version	Part
Zybo Z7-10		digilentinc.com	1.0	xc7z

< Back | Next > | Finish | Cancel

The last page on the window will show a summary of all of the project settings chosen. Compare with the included image.


New Project

New Project Summary

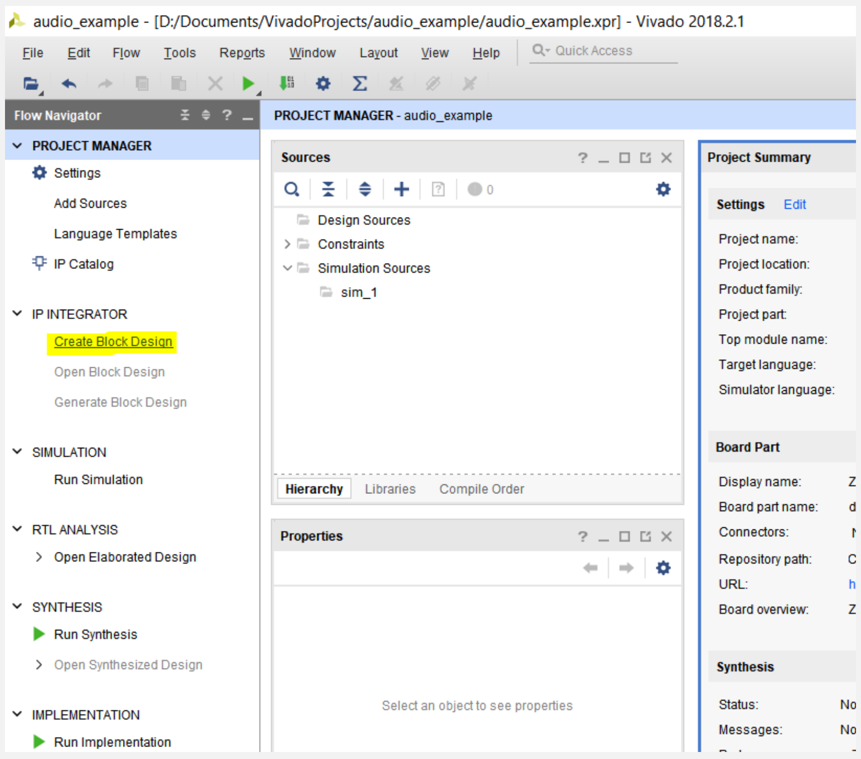
A new RTL project named 'audio_example' will be created.

The default part and product family for the new project:
Default Board: Zybo Z7-10
Default Part: xc7z010clg400-1
Product: Zynq-7000
Family: Zynq-7000
Package: clg400
Speed Grade: -1

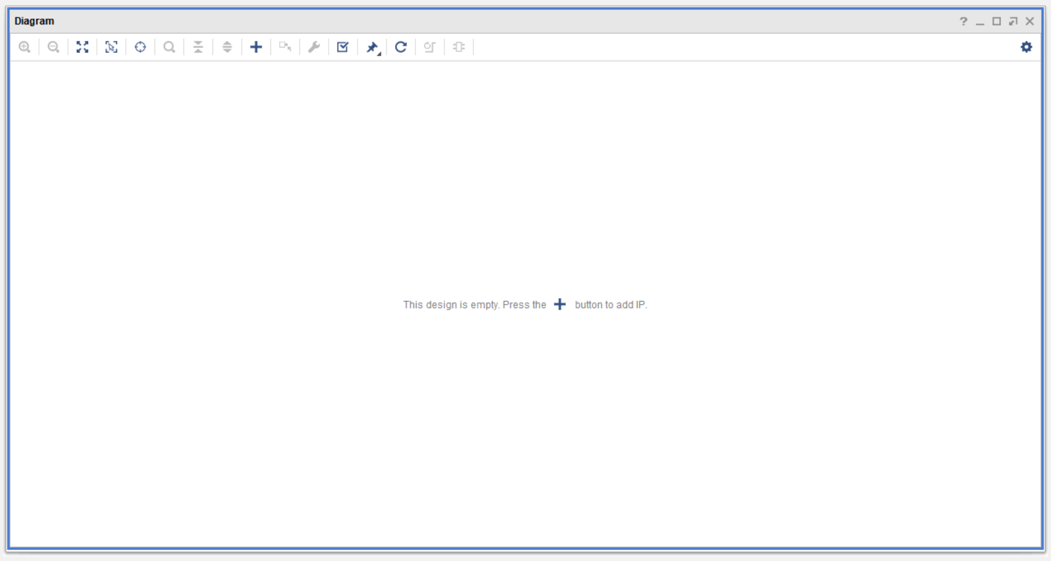
To create the project, click Finish

< Back | Next > | Finish | Cancel

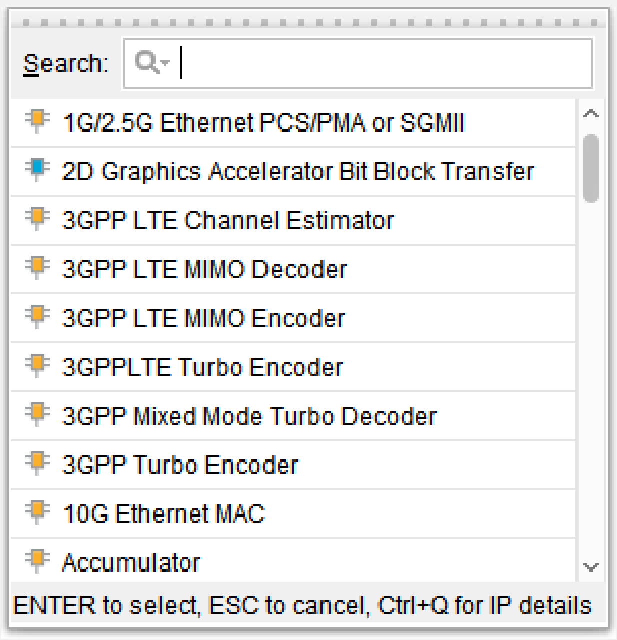
We will begin by creating a new Block Design. You may leave the default Block Design name. Press OK.


audio_example - [D:/Documents/VivadoProjects/audio_example/audio_example.xpr] - Vivado 2018.2.1

Vivado should present you with this window:



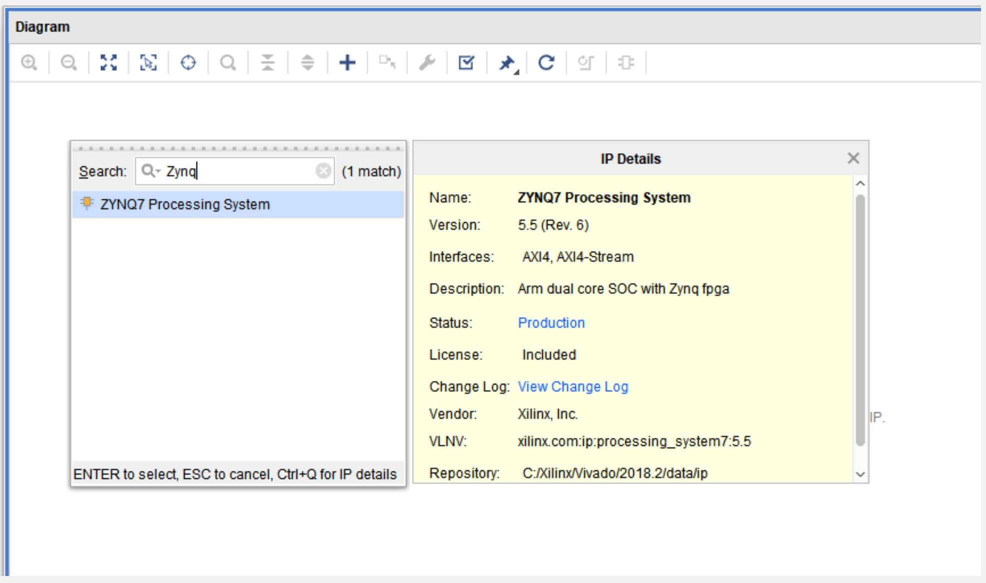
Click on the large + to begin adding IP blocks.


Search:

1G/2.5G Ethernet PCS/PMA or SGMII
2D Graphics Accelerator Bit Block Transfer
3GPP LTE Channel Estimator
3GPP LTE MIMO Decoder
3GPP LTE MIMO Encoder
3GPPLTE Turbo Encoder
3GPP Mixed Mode Turbo Decoder
3GPP Turbo Encoder
10G Ethernet MAC
Accumulator

ENTER to select, ESC to cancel, Ctrl+Q for IP details

The first block we will be adding is the Zynq Processing System.



Diagram

Search: Zynq (1 match)

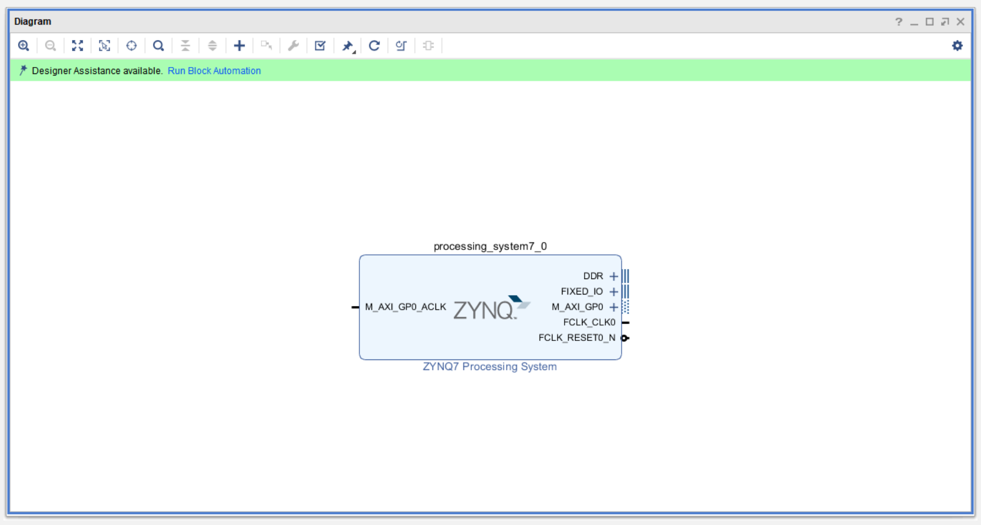
ZYNQ7 Processing System

ENTER to select, ESC to cancel, Ctrl+Q for IP details

IP Details

Name:	**ZYNQ7 Processing System**
Version:	5.5 (Rev. 6)
Interfaces:	AXI4, AXI4-Stream
Description:	Arm dual core SOC with Zynq fpga
Status:	Production
License:	Included
Change Log:	View Change Log
Vendor:	Xilinx, Inc.
VLNV:	xilinx.com:ip:processing_system7:5.5
Repository:	C:/Xilinx/Vivado/2018.2/data/ip

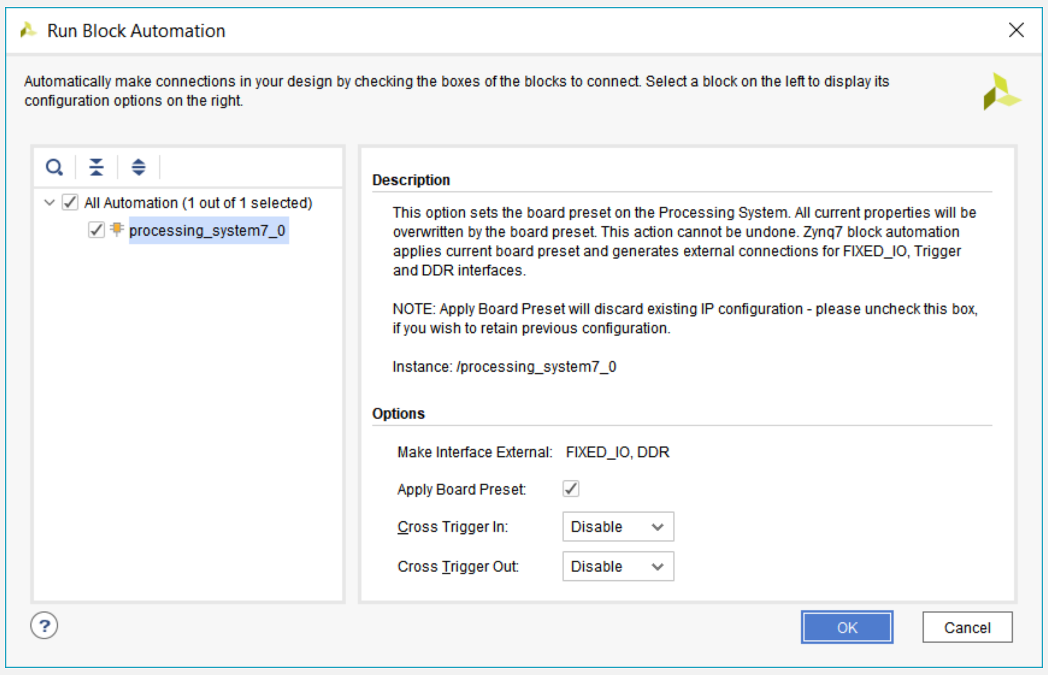
Double click on the IP to add it to your workspace.
A banner should appear stating, “Designer Assistance available. Run Block Automation.”
Click on “Run Block Automation.”



Diagram

Designer Assistance available. Run Block Automation

Leave all options as default. Press OK.



Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

All Automation (1 out of 1 selected)
processing_system7_0

Description

This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone. Zynq7 block automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces.

NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

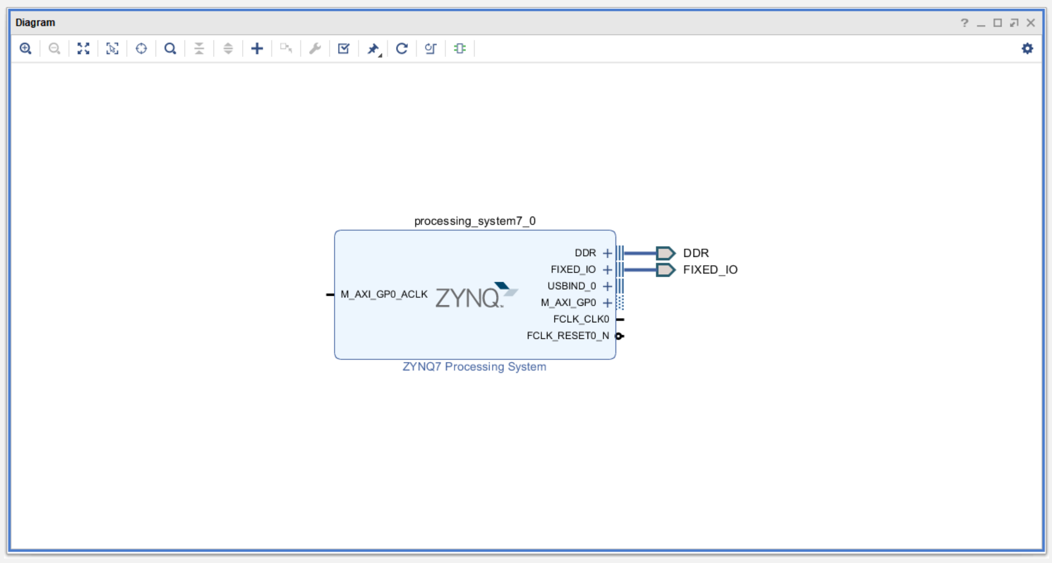
Instance: /processing_system7_0

Options

Make Interface External: FIXED_IO, DDR
Apply Board Preset: ✓
Cross Trigger In: Disable
Cross Trigger Out: Disable

OK Cancel

The results should be two external pin connections to the Zynq Processor IP block.

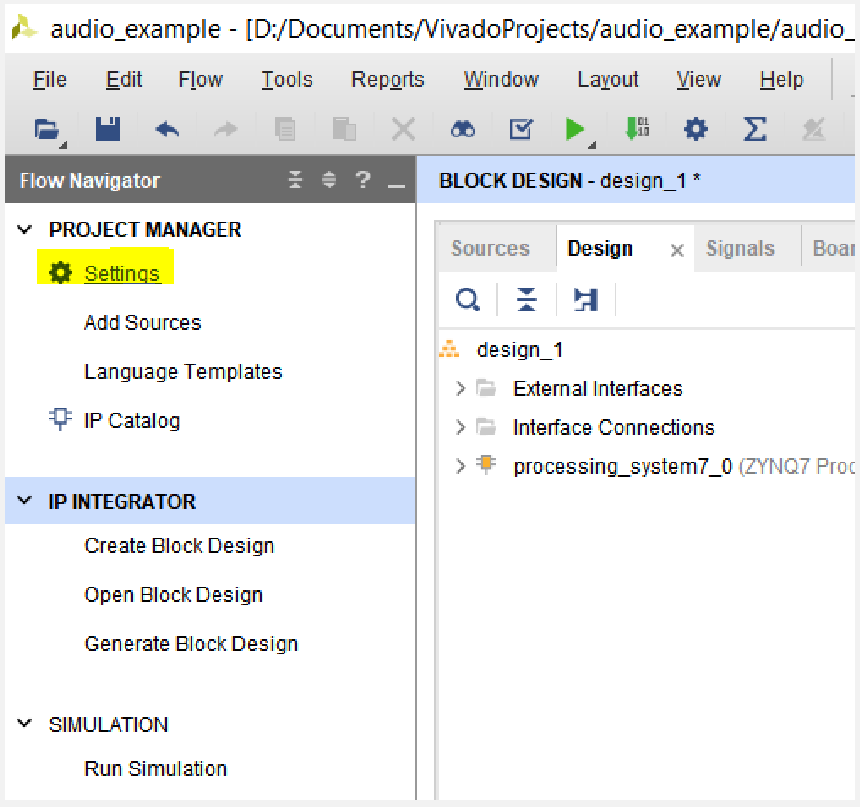


The next step requires the IP block for the Zybo Audio Control IP.

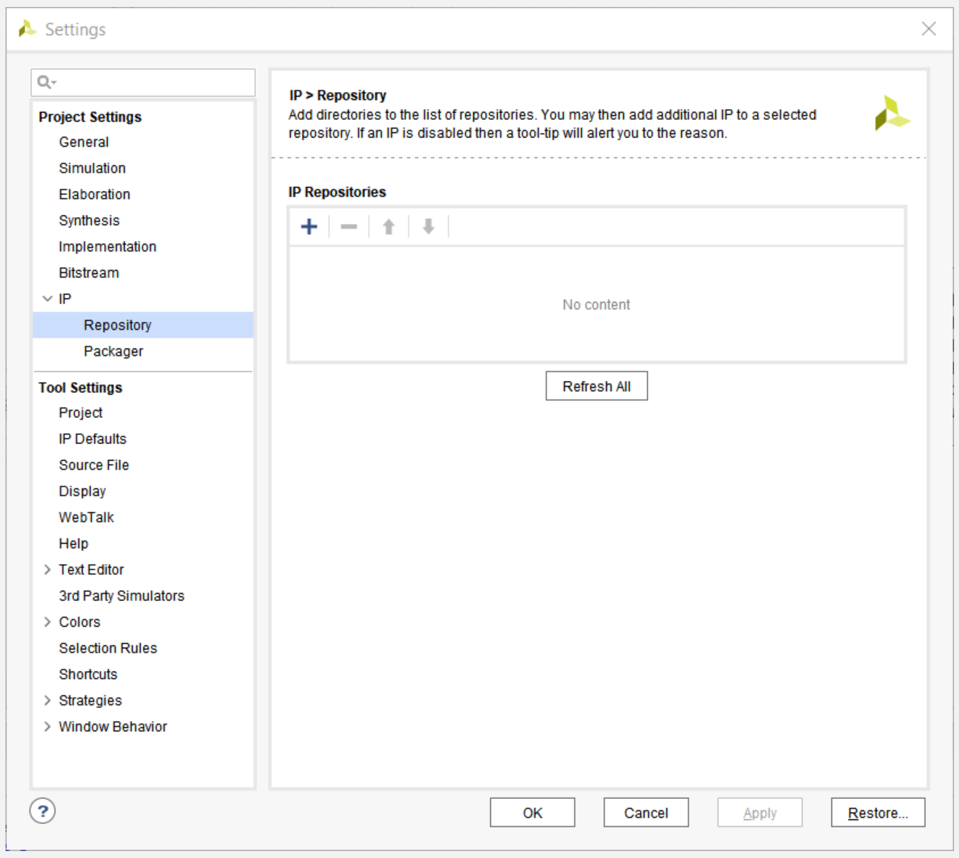
OPEN THE SOURCE DOWNLOAD ZIP FILE "THE_ZYNQ_BOOK_TUTORIAL_SOURCES_AUG_15.ZIP" DOWNLOADED FROM THE ZYNQ BOOK WEBSITE. NAVIGATE TO "SOURCES\ZYBO\ADVENTURES_WITH_IP_INTEGRATOR\IP\" AND EXTRACT THE ZIP FILE "XILINX_COM_ZYBO_AUDIO_CTRL_1.0.ZIP" TO YOUR DESKTOP. NAVIGATE TO YOUR VIVADO INSTALL DIRECTORY AND CREATE THE FOLDER "\DROPIN" AT "\XILINX\VIVADO\2018.2\DATA\IP\"

EXTRACT THE SOURCE IP FOLDER INTO THE NEWLY CREATED "DROPIN" FOLDER WITH A FINAL PATH OF: "\XILINX\VIVADO\2018.2\DATA\IP\DROPIN\XILINX_COM_ZYBO_AUDIO_CTRL_1.0"

We will now add an IP repository to our project to point Vivado towards our included IP files. On the Flow Navigator to the left, under PROJECT MANAGER, click on Settings.



Navigate to Project Settings > IP > Repository



IP > Repository

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.

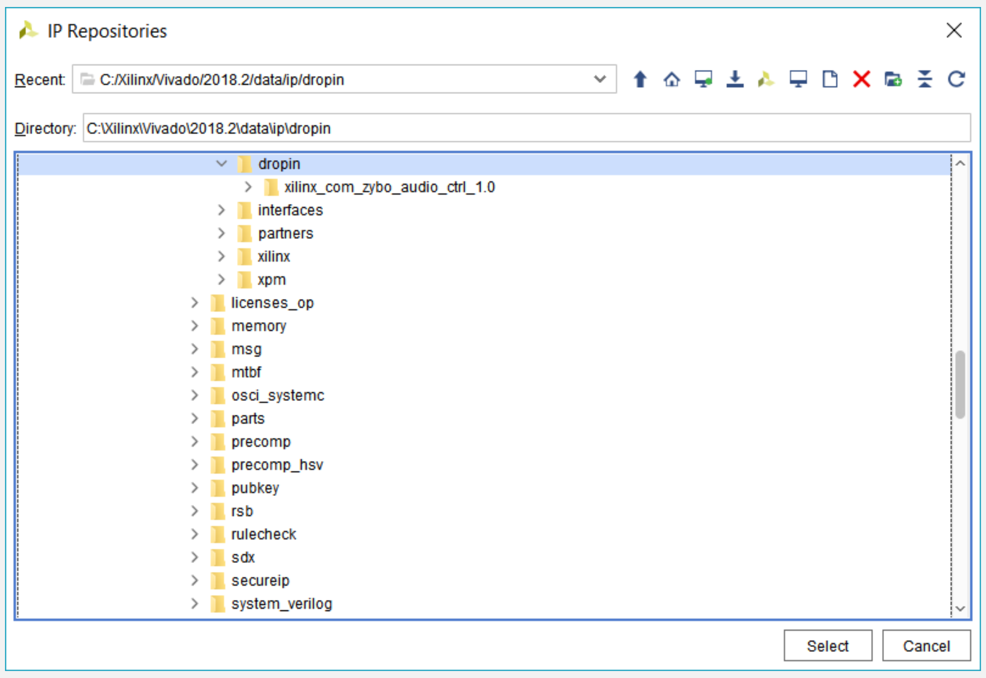
IP Repositories

No content

Refresh All

OK Cancel Apply Restore...

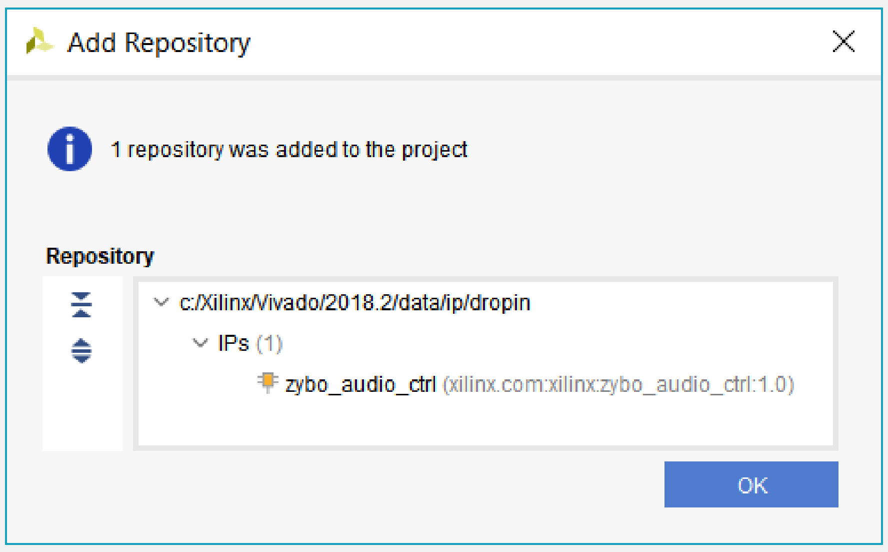
Click on the + sign. This will open a browsing window. Search for the folder in which we saved the Zybo Audio Control IP. In our case, we saved it to "\Xilinx\Vivado\2018.2\data\ip\dropin"


IP Repositories

Recent: C:/Xilinx/Vivado/2018.2/data/ip/dropin

Directory: C:\Xilinx\Vivado\2018.2\data\ip\dropin

Click on the “dropin” folder and click Select. This should add the repository to the project which includes the “zybo_audio_ctrl” IP.


Add Repository

1 repository was added to the project

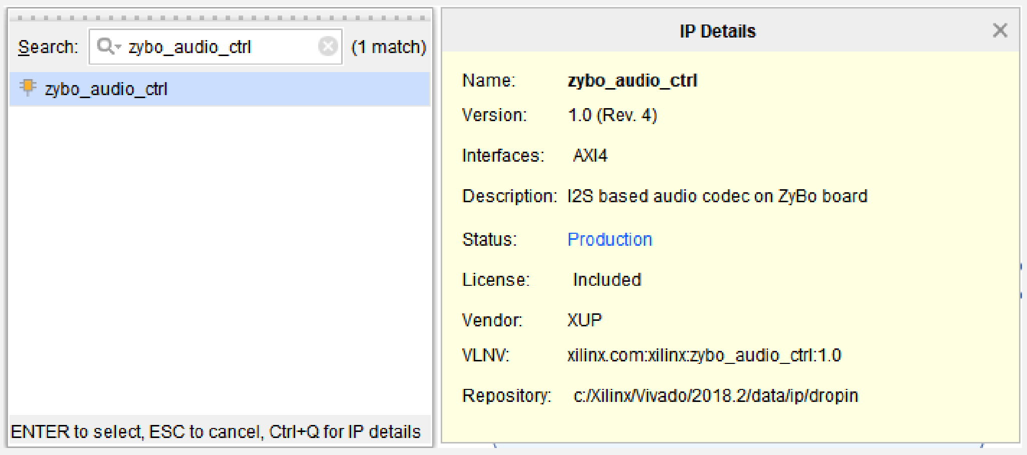
Repository

c:/Xilinx/Vivado/2018.2/data/ip/dropin
IPs (1)
zybo_audio_ctrl (xilinx.com:xilinx:zybo_audio_ctrl:1.0)

OK

The IP should be added to the repository. Click on OK, click on Apply, and close the settings window.

We should now be able to search for and find the “zybo_audio_ctrl” IP block.


Search: zybo_audio_ctrl (1 match)

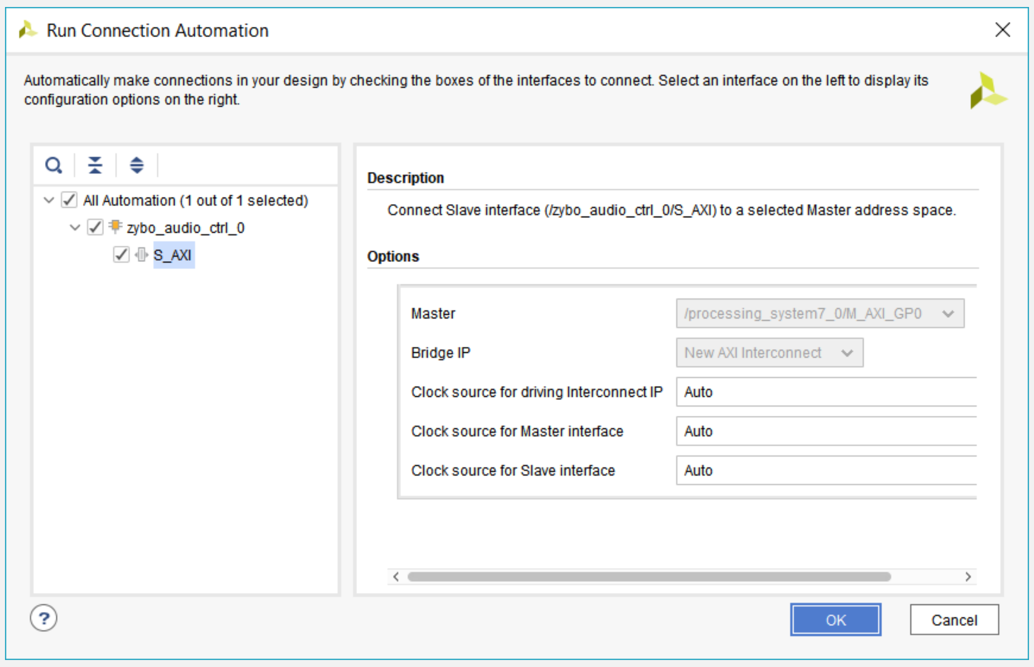
zybo_audio_ctrl

ENTER to select, ESC to cancel, Ctrl+Q for IP details

IP Details

Name:	zybo_audio_ctrl
Version:	1.0 (Rev. 4)
Interfaces:	AXI4
Description:	I2S based audio codec on ZyBo board
Status:	Production
License:	Included
Vendor:	XUP
VLNV:	xilinx.com:xilinx:zybo_audio_ctrl:1.0
Repository:	c:/Xilinx/Vivado/2018.2/data/ip/dropin

Double click on the IP, and run the connection automation again. This should connect the IP block S_AXI connection to an AXI Interconnect which is then connected to the ZYNQ7 Processing System.


Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

- [✓] All Automation (1 out of 1 selected)
  - [✓] zybo_audio_ctrl_0
    - [✓] S_AXI

Description

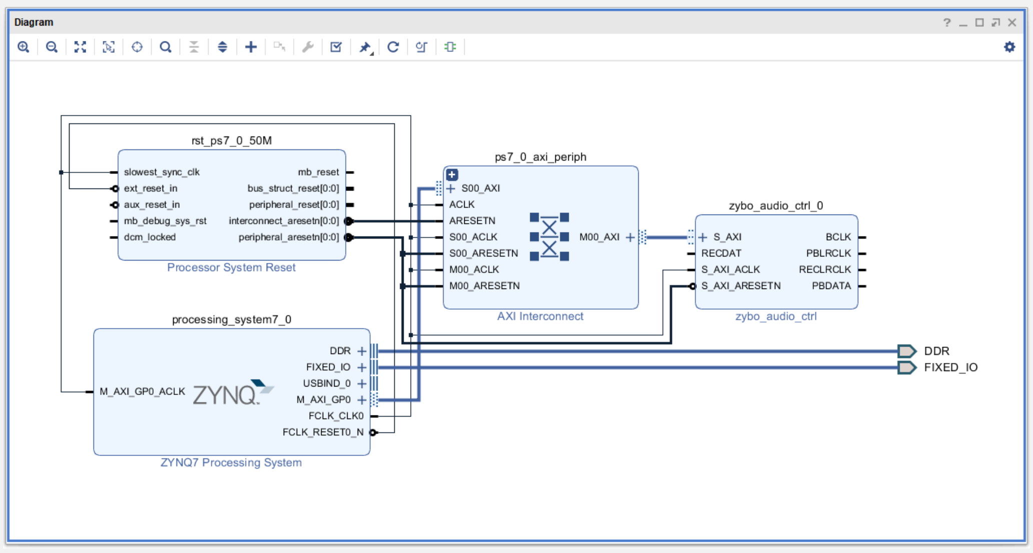
Connect Slave interface (/zybo_audio_ctrl_0/S_AXI) to a selected Master address space.

Options

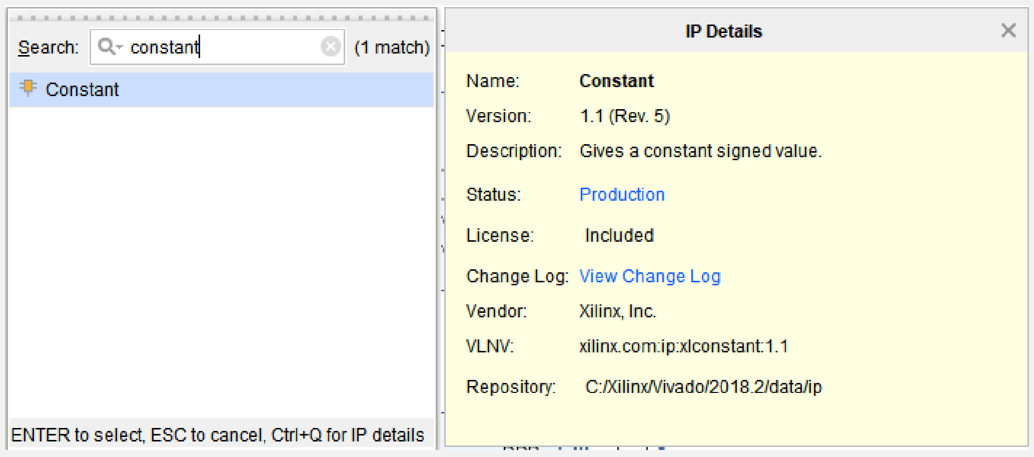
Master	/processing_system7_0/M_AXI_GP0
Bridge IP	New AXI Interconnect
Clock source for driving Interconnect IP	Auto
Clock source for Master interface	Auto
Clock source for Slave interface	Auto

OK Cancel

After the connection automation is complete, click on the “Regenerate Layout” button to clean up the window. The results should be similar to this:



The Zybo FPGA’s SSM2603 Audio Codec requires an enable signal to function. Our project will need the codec always on, so we will add a ‘constant’ IP block to drive the signal high.


Search: constant (1 match)

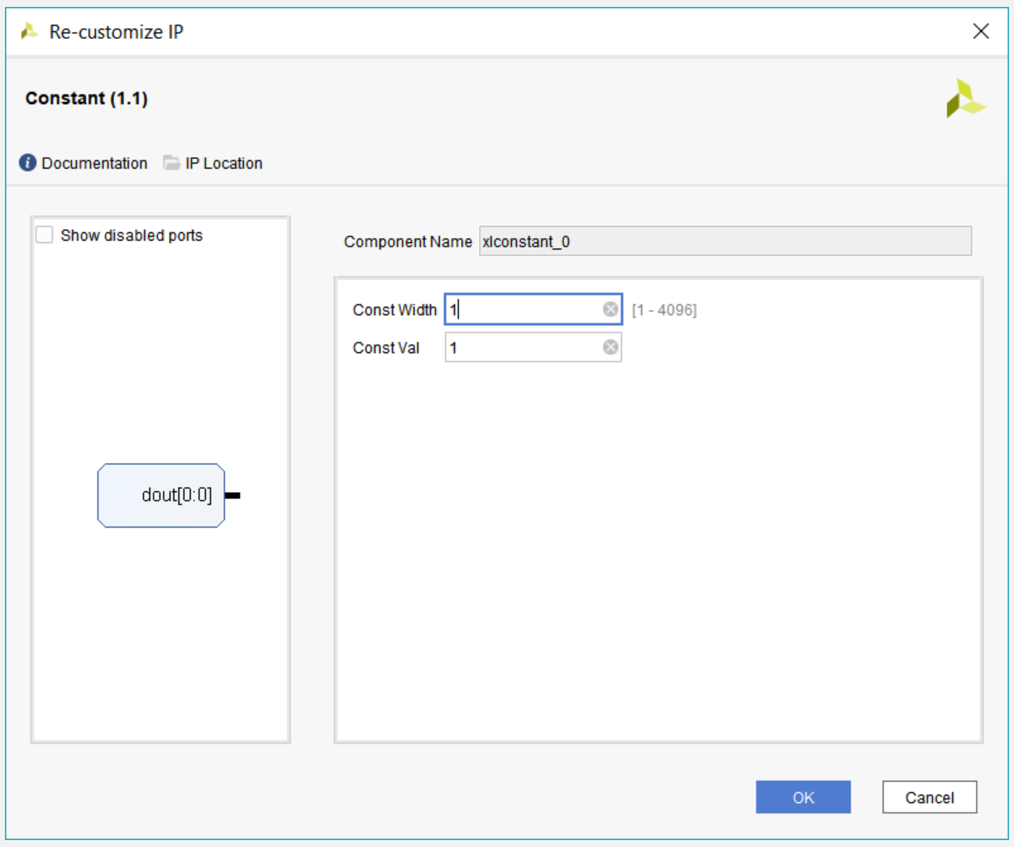
Constant

ENTER to select, ESC to cancel, Ctrl+Q for IP details

IP Details

Name:	Constant
Version:	1.1 (Rev. 5)
Description:	Gives a constant signed value.
Status:	Production
License:	Included
Change Log:	View Change Log
Vendor:	Xilinx, Inc.
VLNV:	xilinx.com:ip:xlconstant:1.1
Repository:	C:/Xilinx/Vivado/2018.2/data/ip

Double clicking on the ‘Constant’ IP block will allow us to adjust its settings.


Re-customize IP

Constant (1.1)

Documentation IP Location

Show disabled ports

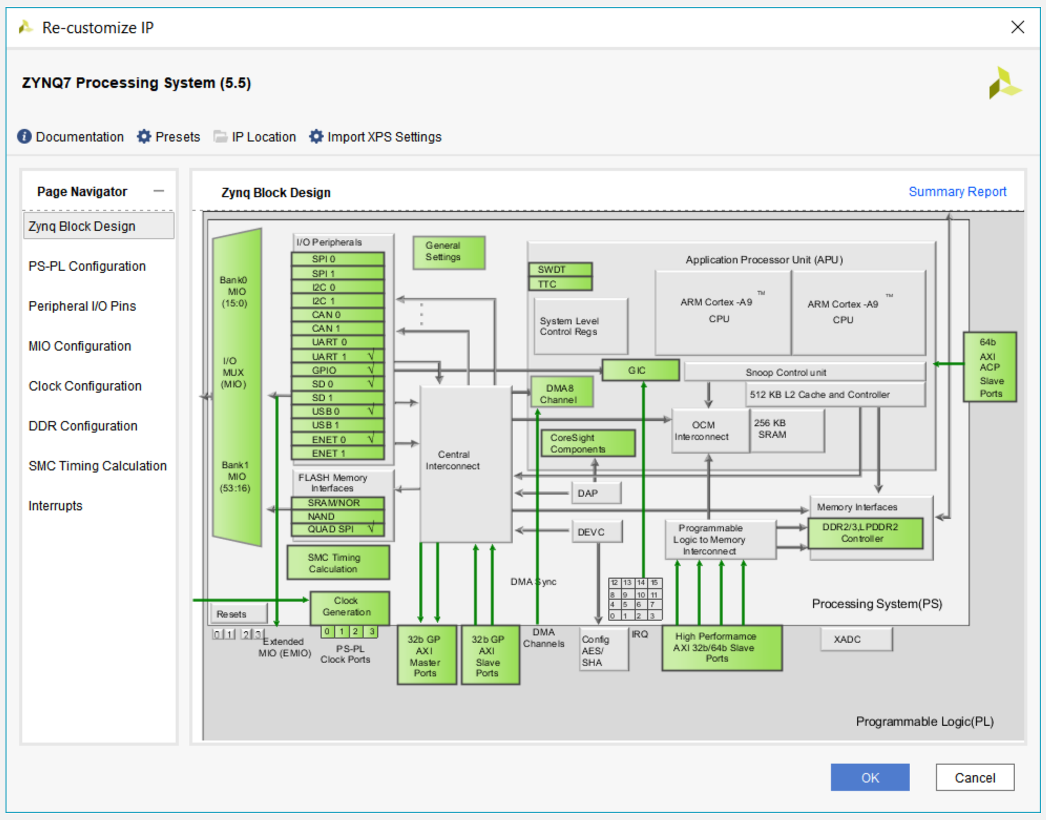
dout[0:0]

Component Name	xlconstant_0	
Const Width	1	[1 - 4096]
Const Val	1	

OK Cancel

Ensure both settings are set to ‘1’, and click on OK.

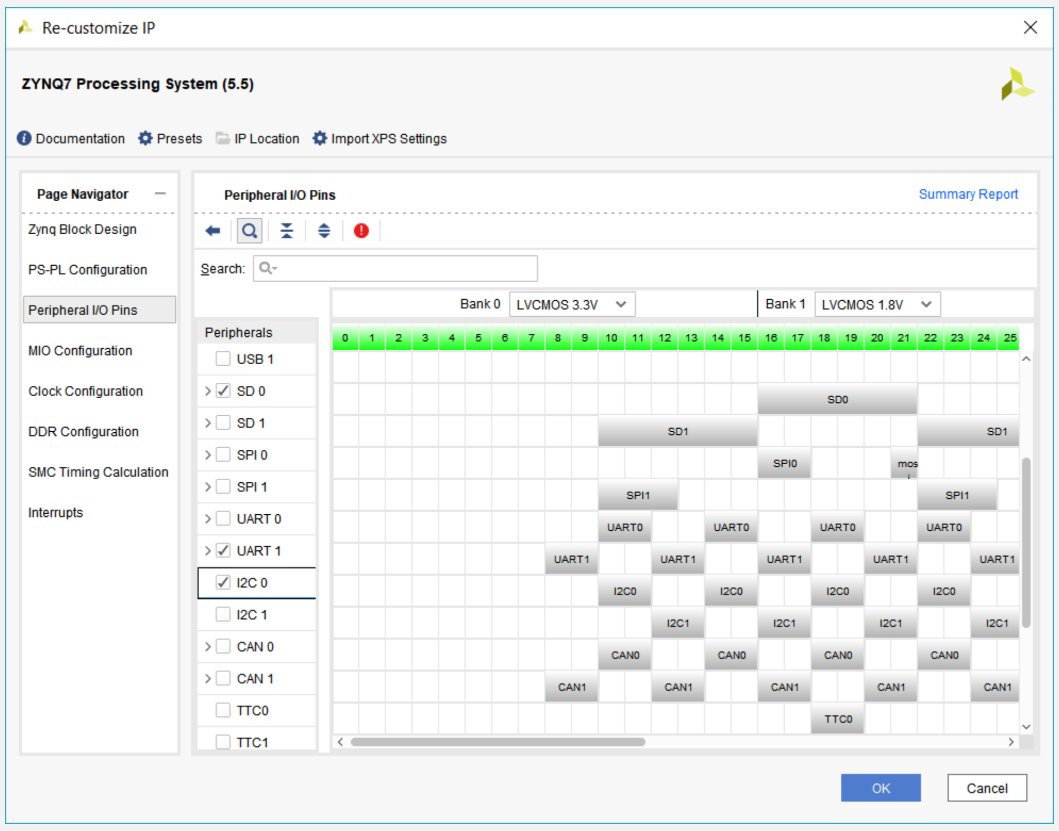
We will now configure the ZYNQ7 Processing System. Double-click on the IP block for the PS.


Re-customize IP

ZYNQ7 Processing System (5.5)

There are two things we must configure for this project. First, we must activate an I2C connection on the processor in order to write to the audio codec's registers. Secondly, we need to create a new clock to drive the audio codec.

To create the I2C connection, we will navigate to the "Peripheral I/O Pins" page on the left-hand Page Navigator. Scroll down to find the 'I2C 0' peripheral and ensure that the box is checked.


Re-customize IP

ZYNQ7 Processing System (5.5)

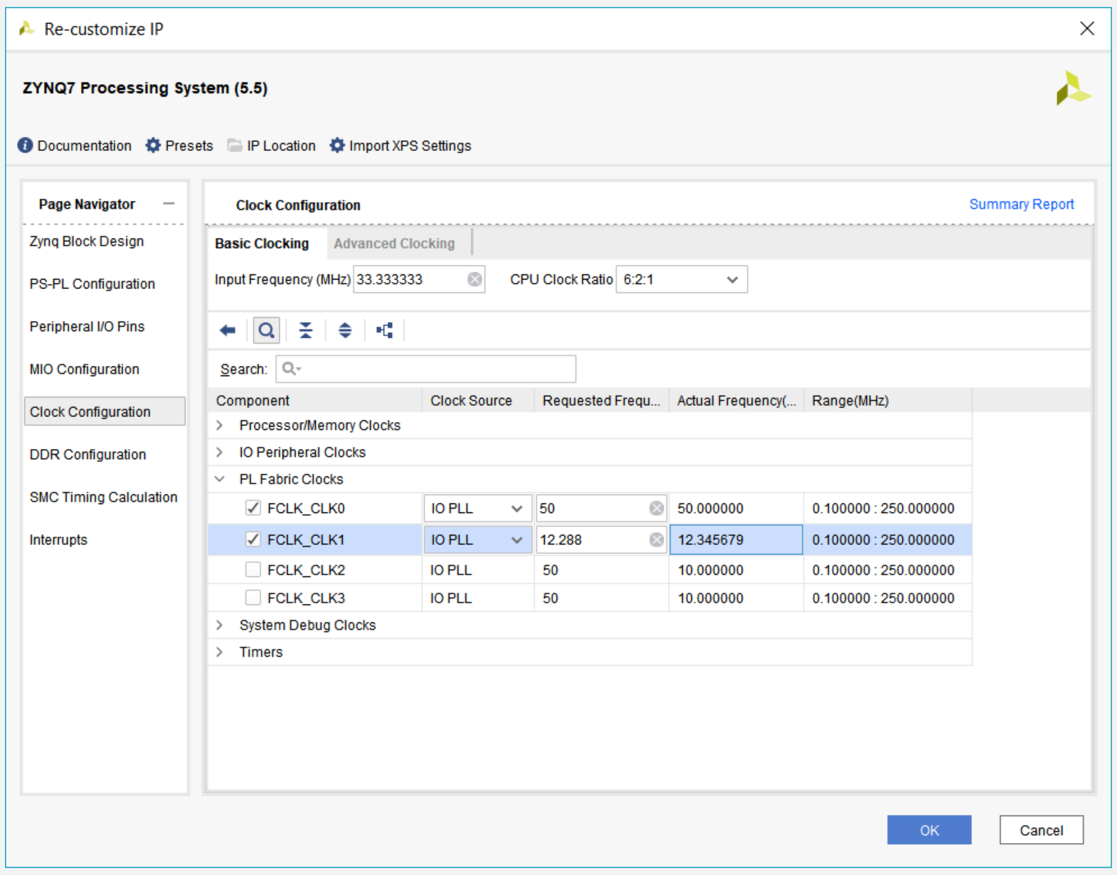
Peripheral I/O Pins

Bank 0 LVCMOS 3.3V — Bank 1 LVCMOS 1.8V

Peripherals
☐ USB 1
✓ SD 0
☐ SD 1
☐ SPI 0
☐ SPI 1
☐ UART 0
✓ UART 1
✓ I2C 0
☐ I2C 1
☐ CAN 0
☐ CAN 1
☐ TTC0
☐ TTC1

Scrolling all the way to the right should show EMIO highlighted in green for the I2C connection.

For the clock configuration, click on "Clock Configuration" on the Page Navigator. Click on the drop-down arrow for 'PL Fabric Clocks.' Check the box next to 'FCLK_CLK1' and type "12.288" under 'Requested Frequency.'


Re-customize IP

ZYNQ7 Processing System (5.5)

Clock Configuration

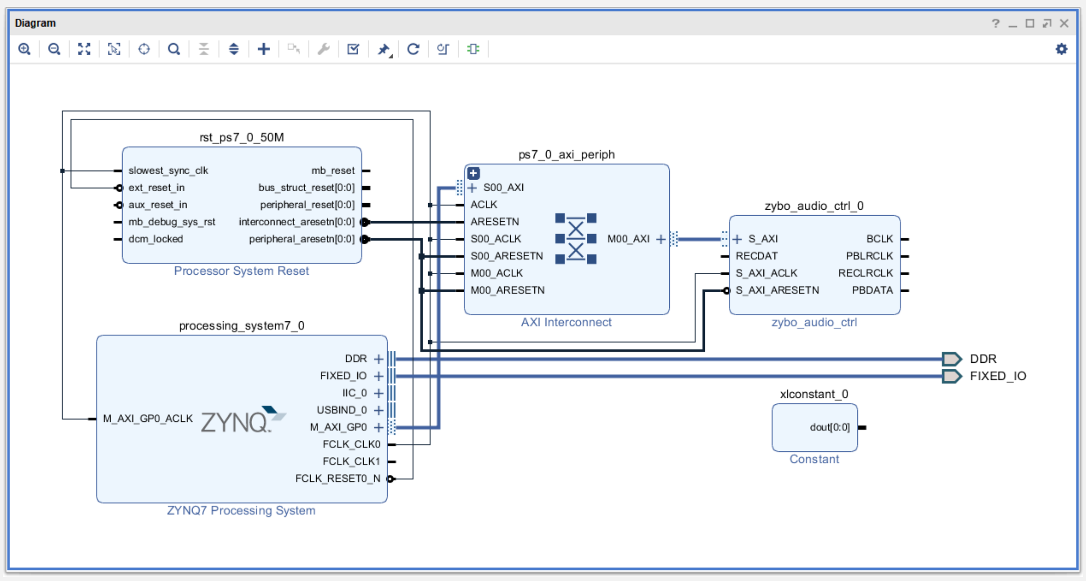
Basic Clocking | Advanced Clocking

Input Frequency (MHz) 33.333333 CPU Clock Ratio 6:2:1

Component	Clock Source	Requested Frequ...	Actual Frequency(...	Range(MHz)
Processor/Memory Clocks				
IO Peripheral Clocks				
PL Fabric Clocks				
✓ FCLK_CLK0	IO PLL	50	50.000000	0.100000 : 250.000000
✓ FCLK_CLK1	IO PLL	12.288	12.345679	0.100000 : 250.000000
☐ FCLK_CLK2	IO PLL	50	10.000000	0.100000 : 250.000000
☐ FCLK_CLK3	IO PLL	50	10.000000	0.100000 : 250.000000
System Debug Clocks				
Timers				

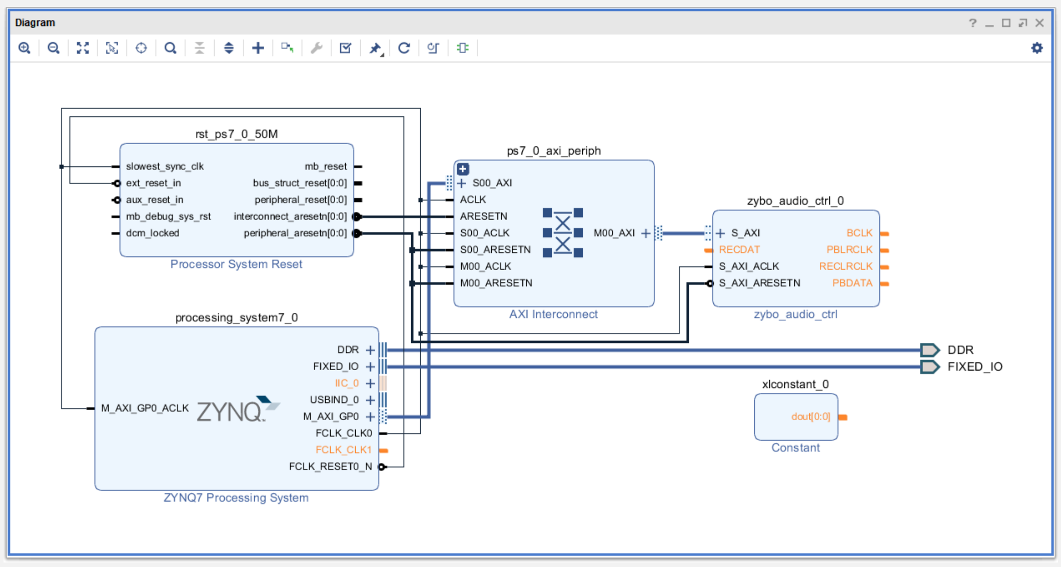
Do note that the 'Actual Frequency' will be different than the requested frequency.

Press OK to close the ZYNQ7 Processing System IP customization window. Two new connections should appear on the ZYNQ7 IP Block: 'IIC_0' and 'FCLK_CLK1'.

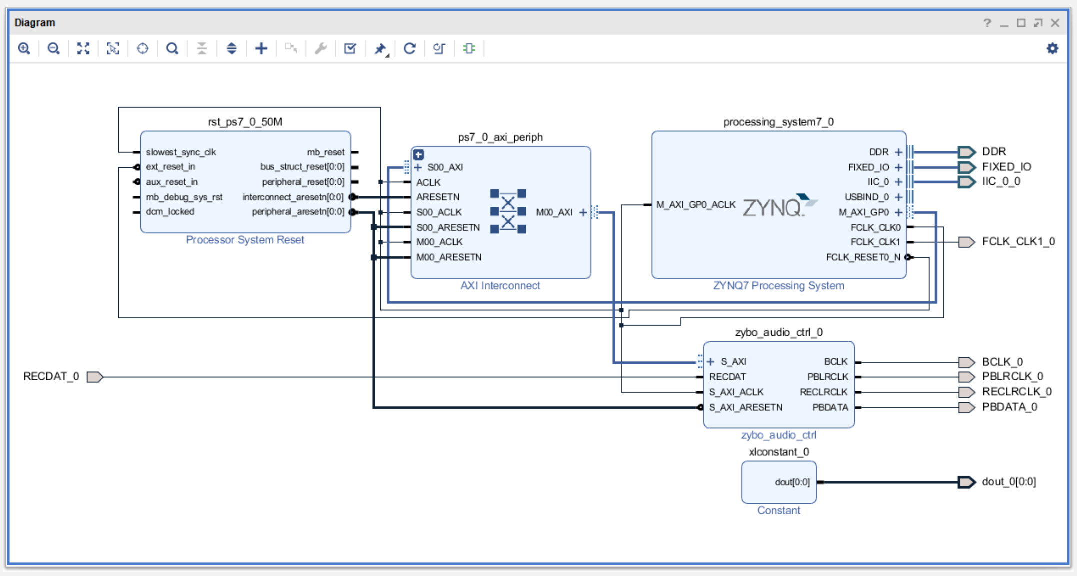


We will now make all necessary "External Connections" in order to 'wire' our components internally.

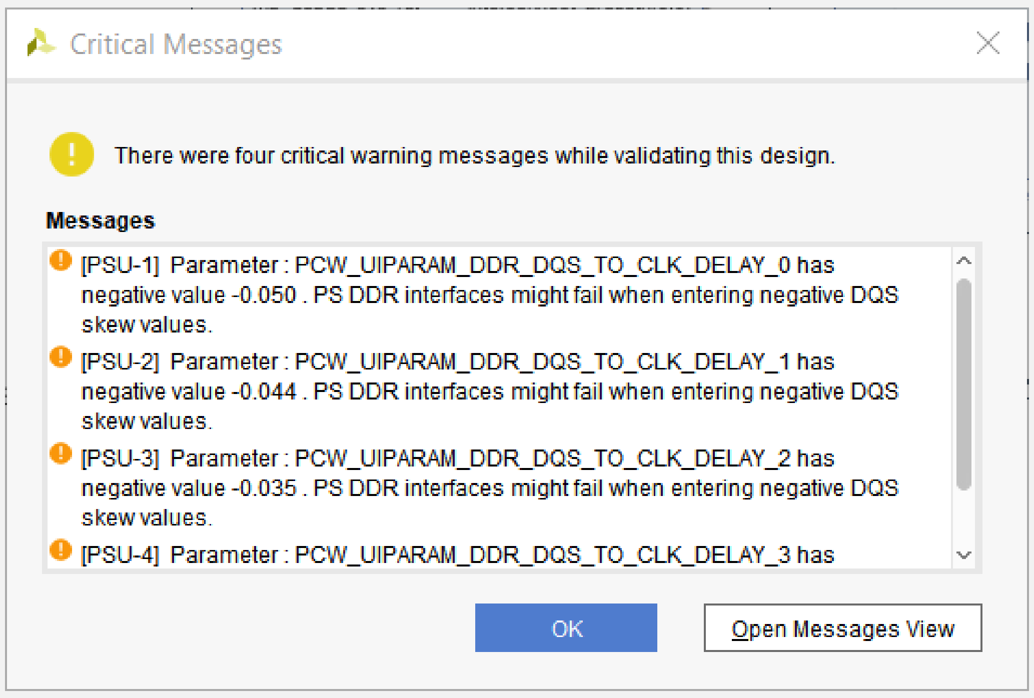
The connections in question are: the IIC_0 and FCLK_CLK1 connections on the ZYNQ7 IP Block: RECDAT, BCLK, PBLRCLK, RECLRCLK, and PBDATA on the zybo_audio_ctrl IP block: and lastly, the dout[0:0] pin on the Constant IP Block.



You may select each pin and right-click to select “Make External.” Or you may Ctrl+Click each pin and press Ctrl+T. After another layout regeneration, the block diagram should look something like this:



That should complete the block diagram. Press F6 on the keyboard to validate the design. You can ignore any messages pertaining to negative DQS skew values.


Critical Messages

There were four critical warning messages while validating this design.

Messages

[PSU-1] Parameter : PCW_UIPARAM_DDR_DQS_TO_CLK_DELAY_0 has negative value -0.050 . PS DDR interfaces might fail when entering negative DQS skew values.

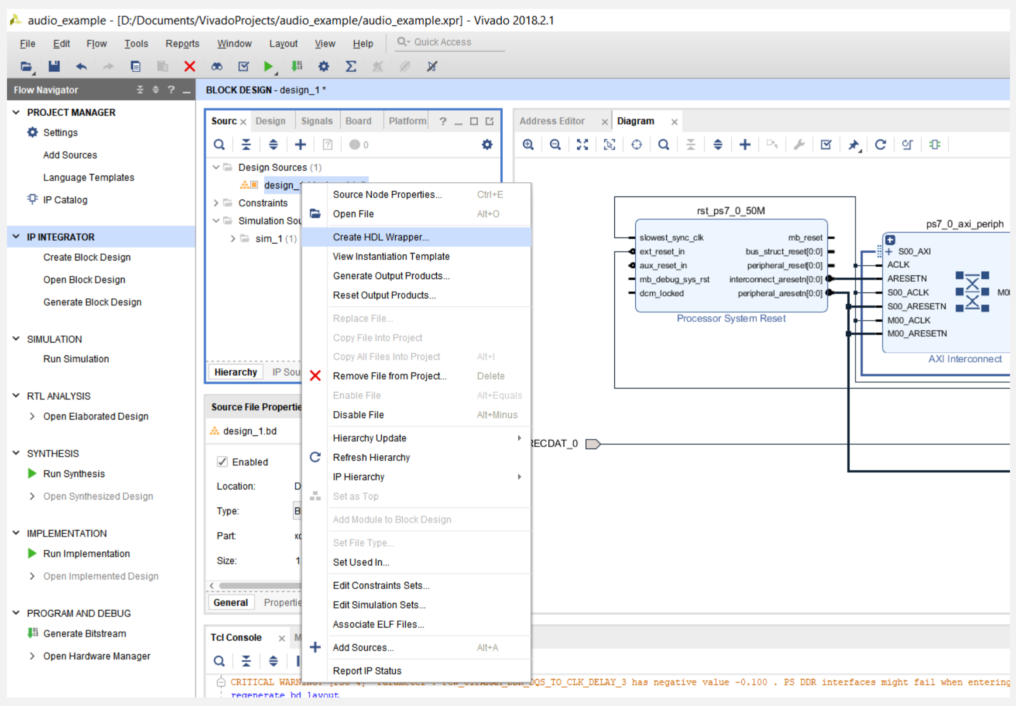
[PSU-2] Parameter : PCW_UIPARAM_DDR_DQS_TO_CLK_DELAY_1 has negative value -0.044 . PS DDR interfaces might fail when entering negative DQS skew values.

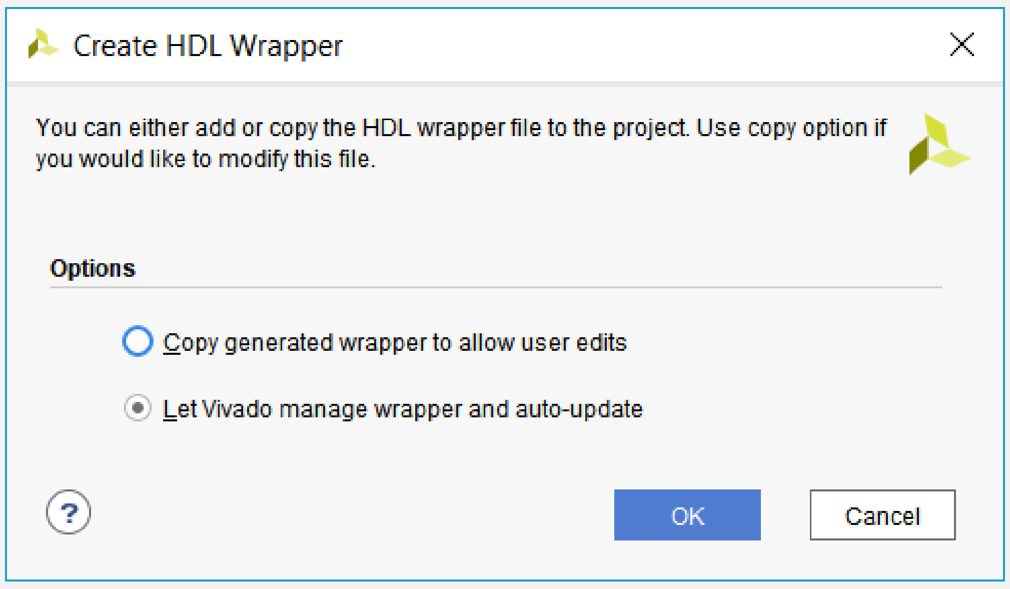
[PSU-3] Parameter : PCW_UIPARAM_DDR_DQS_TO_CLK_DELAY_2 has negative value -0.035 . PS DDR interfaces might fail when entering negative DQS skew values.

[PSU-4] Parameter : PCW_UIPARAM_DDR_DQS_TO_CLK_DELAY_3 has

If this ‘Critical Messages’ window appears, click on OK to close it.

We will now create the Block Diagram wrapper for our design. In the Hierarchy view under BLOCK DESIGN, right-click on the design_1.bd Design Source, and click on “Create HDL Wrapper…”




Create HDL Wrapper

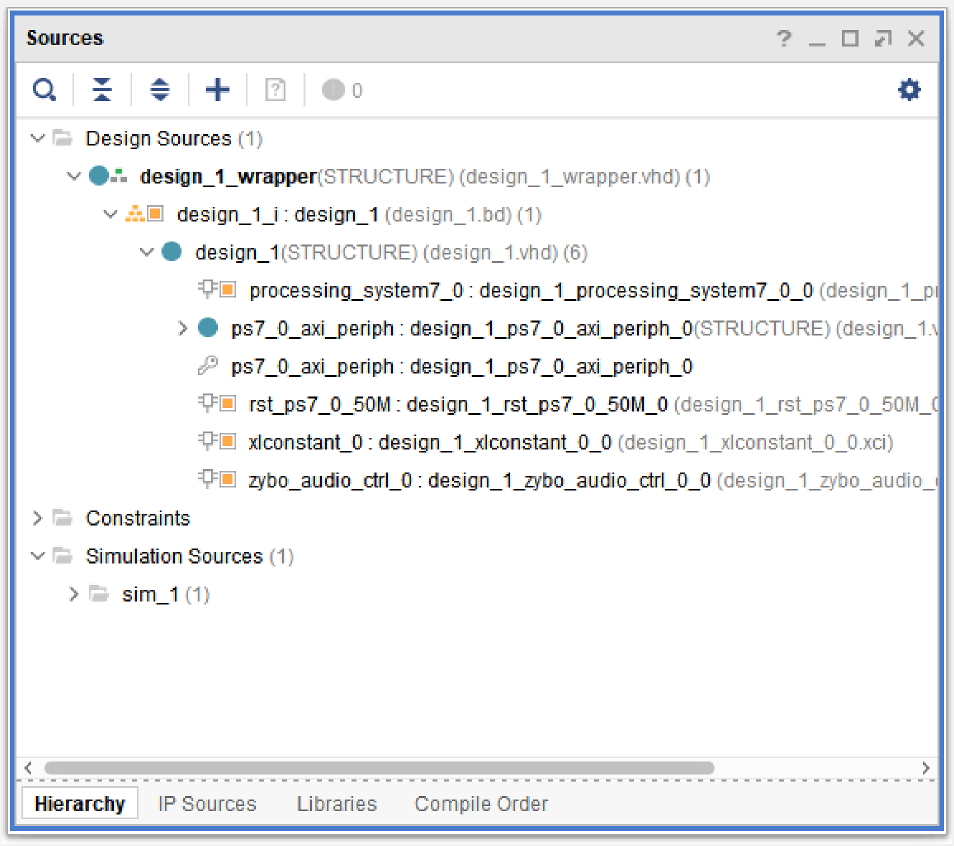
You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.

Options

Copy generated wrapper to allow user edits [✓ Let Vivado manage wrapper and auto-update]

Let Vivado manage the wrapper and press OK.

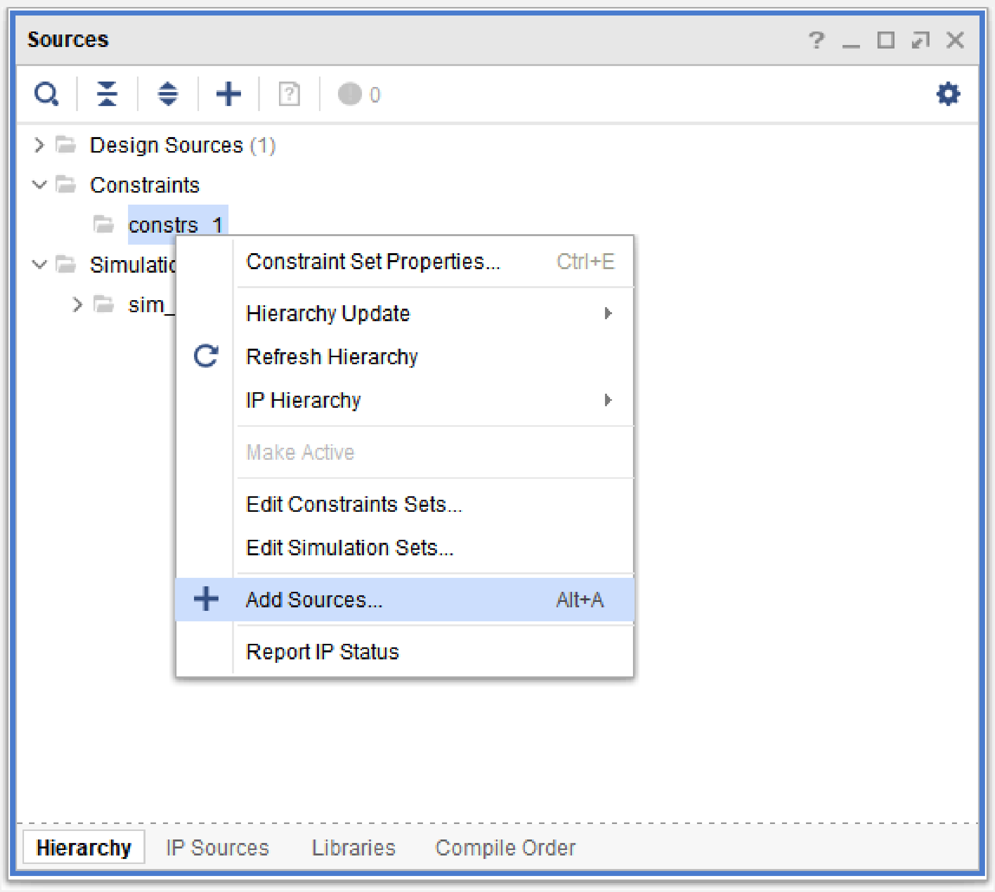
After some time, the wrapper will generate and the hierarchy drop down for your design sources will update.


Sources

Design Sources (1)
  design_1_wrapper(STRUCTURE) (design_1_wrapper.vhd) (1)
    design_1_i : design_1 (design_1.bd) (1)
      design_1(STRUCTURE) (design_1.vhd) (6)
        processing_system7_0 : design_1_processing_system7_0_0 (design_1_p
        ps7_0_axi_periph : design_1_ps7_0_axi_periph_0(STRUCTURE) (design_1.
        ps7_0_axi_periph : design_1_ps7_0_axi_periph_0
        rst_ps7_0_50M : design_1_rst_ps7_0_50M_0 (design_1_rst_ps7_0_50M_
        xlconstant_0 : design_1_xlconstant_0_0 (design_1_xlconstant_0_0.xci)
        zybo_audio_ctrl_0 : design_1_zybo_audio_ctrl_0_0 (design_1_zybo_audio_
Constraints
Simulation Sources (1)
  sim_1 (1)

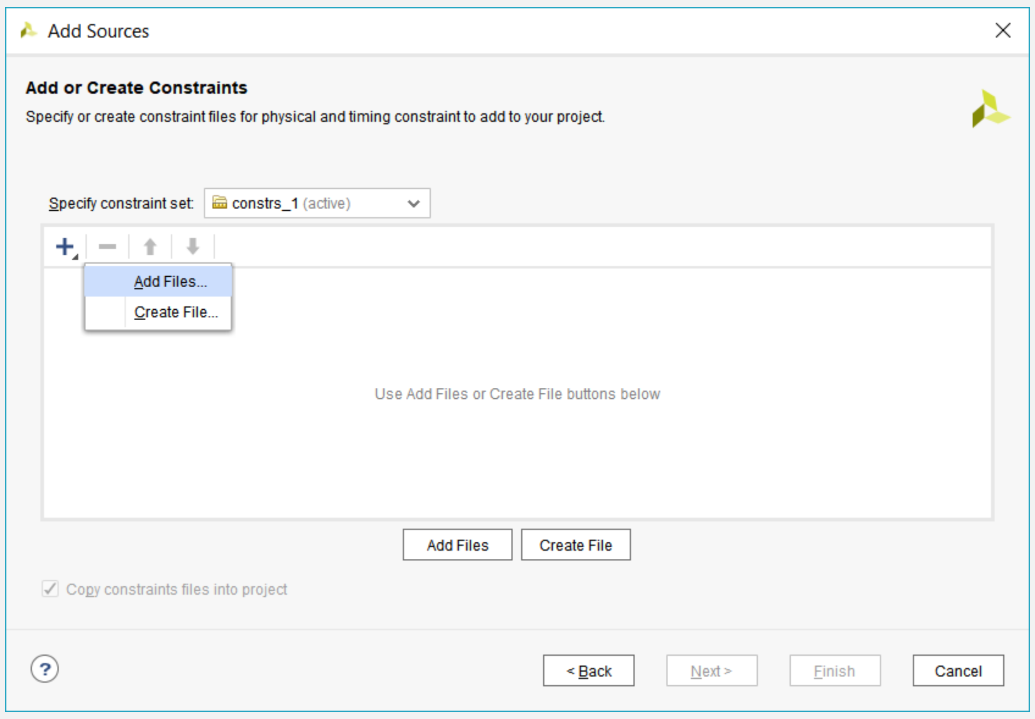
We will now need to configure our Constraints file.

Under the same Sources window, click on the drop down for Constraints, and right-click on constrs_1. Click on “Add Sources…”



Click on “Add or create constraints” and click on Next.

Clicking on the + Symbol will allow us to Add or Create constraint files. We will be adding the Master XDC file we downloaded earlier.


Add Sources

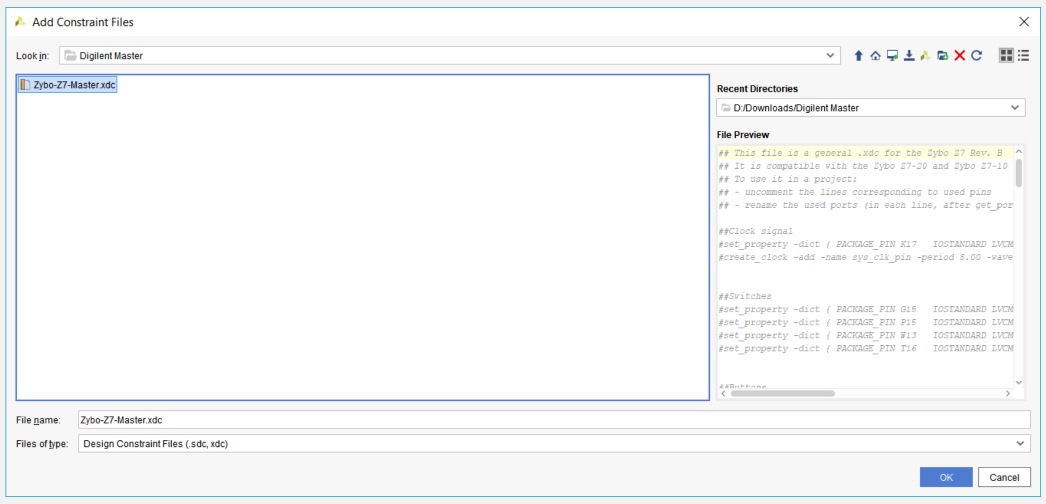
Add or Create Constraints

Specify or create constraint files for physical and timing constraint to add to your project.

Specify constraint set: constrs_1 (active)

Use Add Files or Create File buttons below

Copy constraints files into project


Add Constraint Files

Look in: Digilent Master

Recent Directories: D:/Downloads/Digilent Master

File Preview

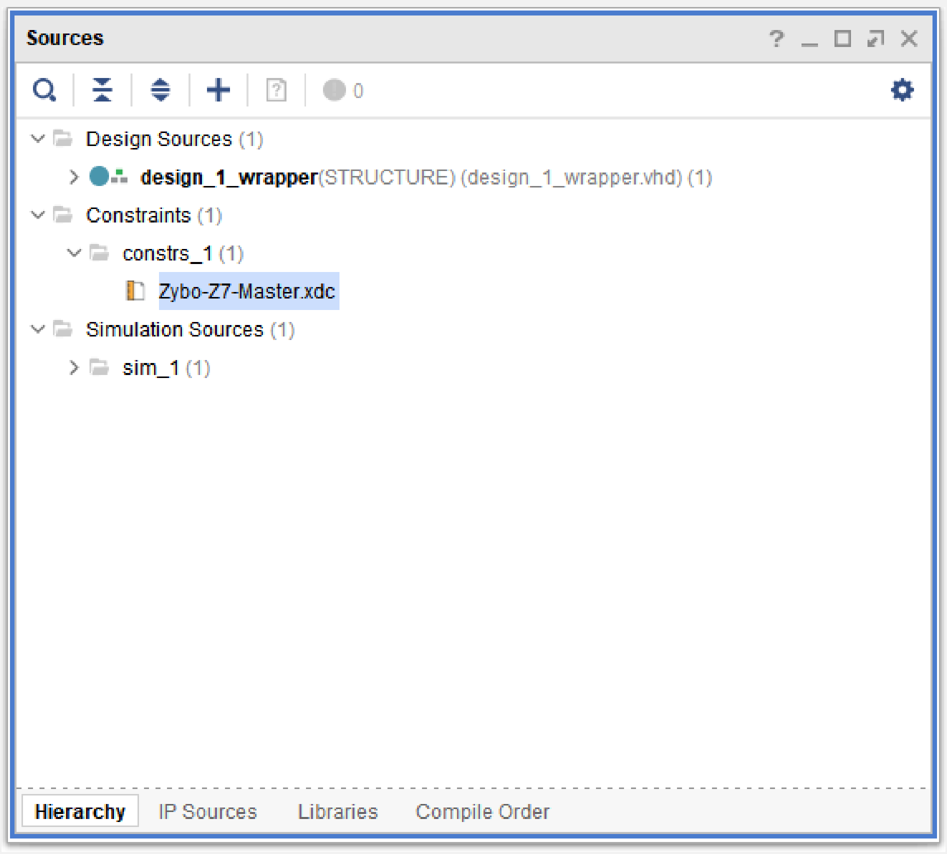
```
## This file is a general .xdc for the Zybo Z7 Rev. B
## It is compatible with the Zybo Z7-20 and Zybo Z7-10
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_por

##Clock signal
#set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCM
#create_clock -add -name sys_clk_pin -period 8.00 -wave


##Switches
#set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCM
#set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCM
#set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCM
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCM
```

File name: Zybo-Z7-Master.xdc

Files of type: Design Constraint Files (.sdc, xdc)

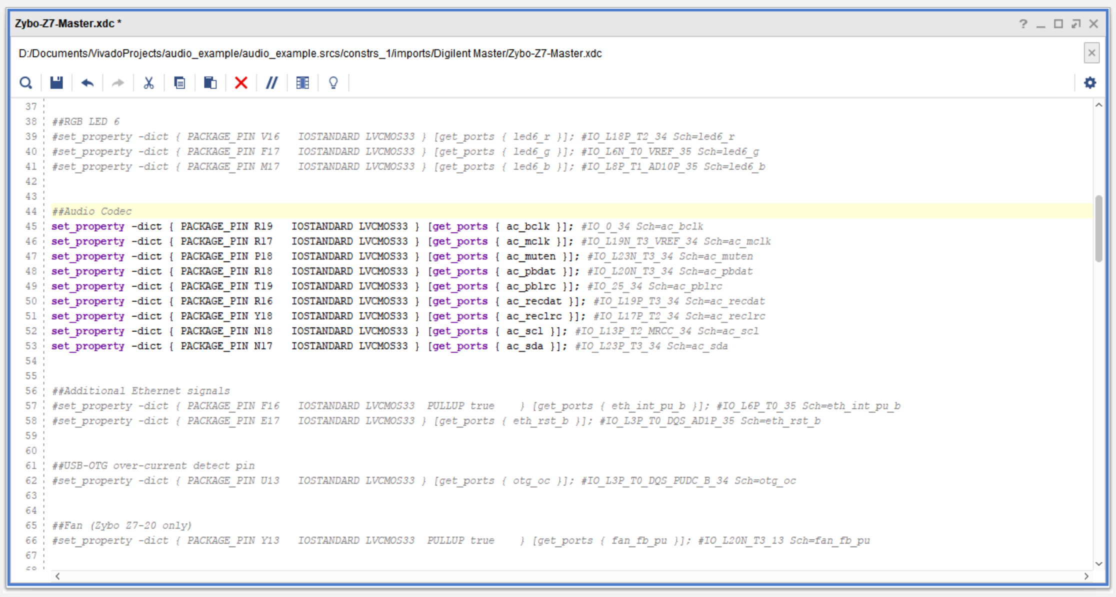
Navigate to where you downloaded the Master Constraints file, click on the .xdc file, ensure that “Copy constraints files into project” is checked, and click on Finish.

This will add a copy of the constraints file to your local project.



We will now need to edit the constraints file. Double click on the “Zybo-Z7-Master.xdc” in the Sources window to edit it in Vivado.

The part we are interested in is the audio section. Scroll down to line 44 and uncomment all the pin declarations for the audio codec.

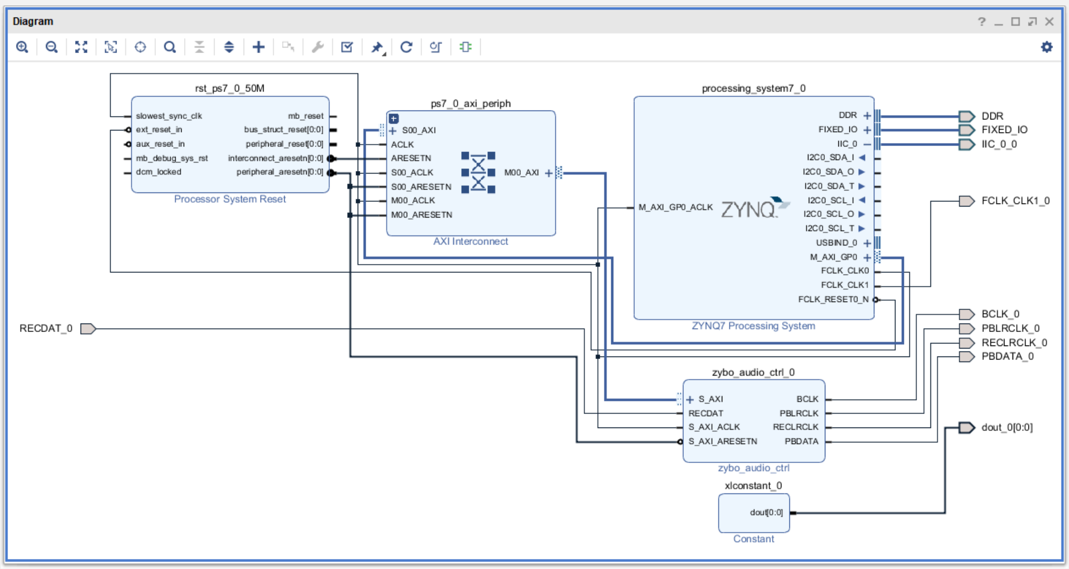

Zybo-Z7-Master.xdc *

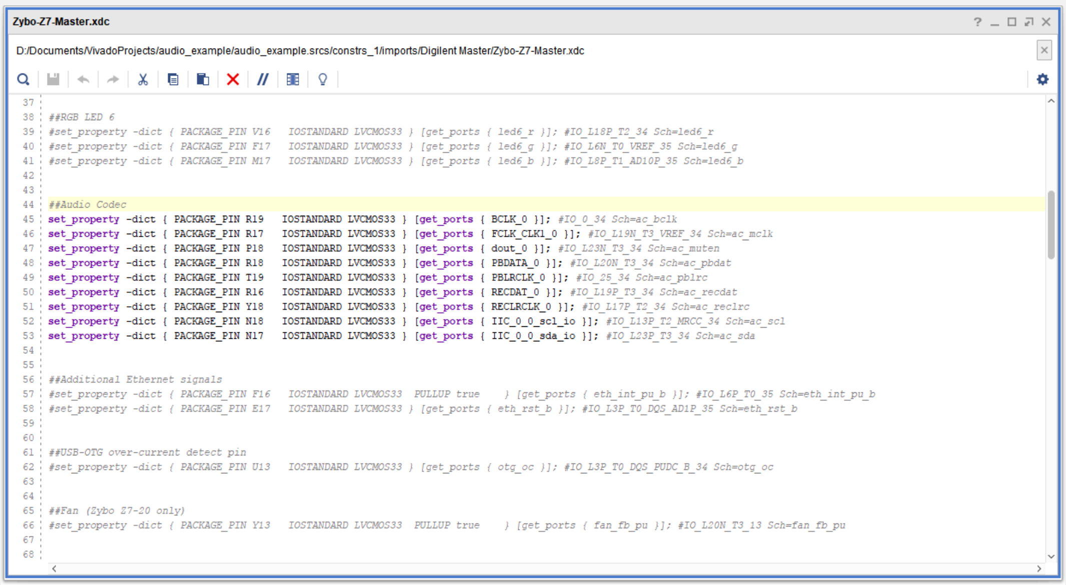
D:/Documents/VivadoProjects/audio_example/audio_example.srcs/constrs_1/imports/Digilent Master/Zybo-Z7-Master.xdc

```
37
38 ##RGB LED 6
39 #set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { led6_r }]; #IO_L18P_T2_34 Sch=led6_r
40 #set_property -dict { PACKAGE_PIN F17   IOSTANDARD LVCMOS33 } [get_ports { led6_g }]; #IO_L6N_T0_VREF_35 Sch=led6_g
41 #set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { led6_b }]; #IO_L8P_T1_AD10P_35 Sch=led6_b
42
43
44 ##Audio Codec
45 set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports { ac_bclk }]; #IO_0_34 Sch=ac_bclk
46 set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { ac_mclk }]; #IO_L19N_T3_VREF_34 Sch=ac_mclk
47 set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { ac_muten }]; #IO_L23N_T3_34 Sch=ac_muten
48 set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { ac_pbdat }]; #IO_L20N_T3_34 Sch=ac_pbdat
49 set_property -dict { PACKAGE_PIN T19   IOSTANDARD LVCMOS33 } [get_ports { ac_pblrc }]; #IO_25_34 Sch=ac_pblrc
50 set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { ac_recdat }]; #IO_L19P_T3_34 Sch=ac_recdat
51 set_property -dict { PACKAGE_PIN Y18   IOSTANDARD LVCMOS33 } [get_ports { ac_reclrc }]; #IO_L17P_T2_34 Sch=ac_reclrc
52 set_property -dict { PACKAGE_PIN N18   IOSTANDARD LVCMOS33 } [get_ports { ac_scl }]; #IO_L13P_T2_MRCC_34 Sch=ac_scl
53 set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { ac_sda }]; #IO_L23P_T3_34 Sch=ac_sda
54
55
56 ##Additional Ethernet signals
57 #set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33  PULLUP true    } [get_ports { eth_int_pu_b }]; #IO_L6P_T0_35 Sch=eth_int_pu_b
58 #set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports { eth_rst_b }]; #IO_L3P_T0_DQS_AD1P_35 Sch=eth_rst_b
59
60
61 ##USB-OTG over-current detect pin
62 #set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { otg_oc }]; #IO_L3P_T0_DQS_PUDC_B_34 Sch=otg_oc
63
64
65 ##Fan (Zybo Z7-20 only)
66 #set_property -dict { PACKAGE_PIN Y13   IOSTANDARD LVCMOS33  PULLUP true    } [get_ports { fan_fb_pu }]; #IO_L20N_T3_13 Sch=fan_fb_pu
67
```

You may choose to rename all the constraint files pin designations to match the external pins present in our block diagram, or you can rename the external pins in the block diagram to match the pin names in the constraints file. For this tutorial, we will rename the pin designations in the constraints file.

Here is our block diagram once more, and the corresponding pins mapped in the constraints file:




Zybo-Z7-Master.xdc

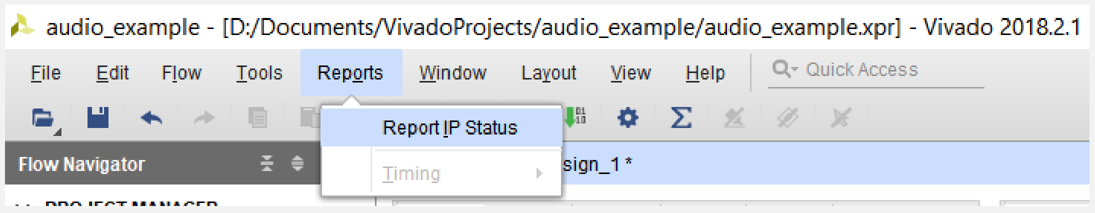
D:/Documents/VivadoProjects/audio_example/audio_example.srcs/constrs_1/imports/Digilent Master/Zybo-Z7-Master.xdc

```

##RGB LED 6
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { led6_r }]; #IO_L18P_T2_34 Sch=led6_r
#set_property -dict { PACKAGE_PIN F17   IOSTANDARD LVCMOS33 } [get_ports { led6_g }]; #IO_L6N_T0_VREF_35 Sch=led6_g
#set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { led6_b }]; #IO_L8P_T1_AD10P_35 Sch=led6_b


##Audio Codec
set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports { BCLK_0 }]; #IO_0_34 Sch=ac_bclk
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { FCLK_CLK1_0 }]; #IO_L19N_T3_VREF_34 Sch=ac_mclk
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { dout_0 }]; #IO_L23N_T3_34 Sch=ac_muten
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { PBDATA_0 }]; #IO_L20N_T3_34 Sch=ac_pbdat
set_property -dict { PACKAGE_PIN T19   IOSTANDARD LVCMOS33 } [get_ports { PBLRCLK_0 }]; #IO_25_34 Sch=ac_pblrc
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { RECDAT_0 }]; #IO_L19P_T3_34 Sch=ac_recdat
set_property -dict { PACKAGE_PIN Y18   IOSTANDARD LVCMOS33 } [get_ports { RECLRCLK_0 }]; #IO_L17P_T2_34 Sch=ac_reclrc
set_property -dict { PACKAGE_PIN N18   IOSTANDARD LVCMOS33 } [get_ports { IIC_0_0_scl_io }]; #IO_L13P_T2_MRCC_34 Sch=ac_scl
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { IIC_0_0_sda_io }]; #IO_L23P_T3_34 Sch=ac_sda


##Additional Ethernet signals
#set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33  PULLUP true    } [get_ports { eth_int_pu_b }]; #IO_L6P_T0_35 Sch=eth_int_pu_b
#set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports { eth_rst_b }]; #IO_L3P_T0_DQS_AD1P_35 Sch=eth_rst_b


##USB-OTG over-current detect pin
#set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { otg_oc }]; #IO_L3P_T0_DQS_PUDC_B_34 Sch=otg_oc


##Fan (Zybo Z7-20 only)
#set_property -dict { PACKAGE_PIN Y13   IOSTANDARD LVCMOS33  PULLUP true    } [get_ports { fan_fb_pu }]; #IO_L20N_T3_13 Sch=fan_fb_pu


```

Note the format for the names in the constraints file for the IIC connections (N18 and N17) as well as the name for the mute enable pin (P18).

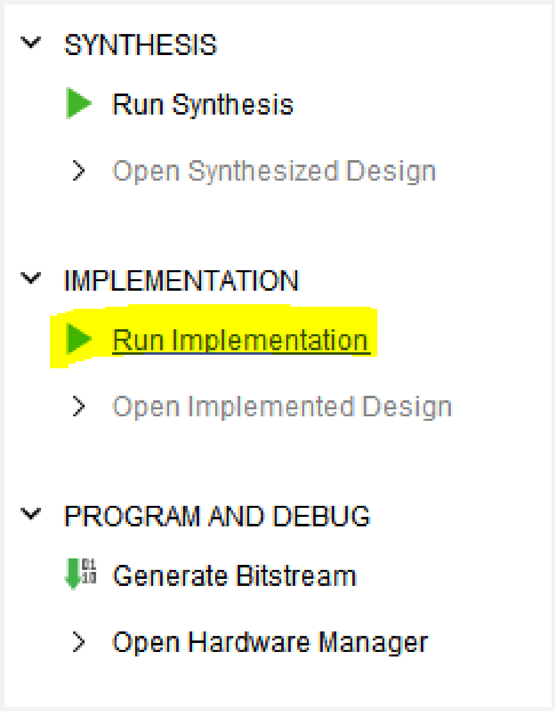
Save your block design and constraints file.

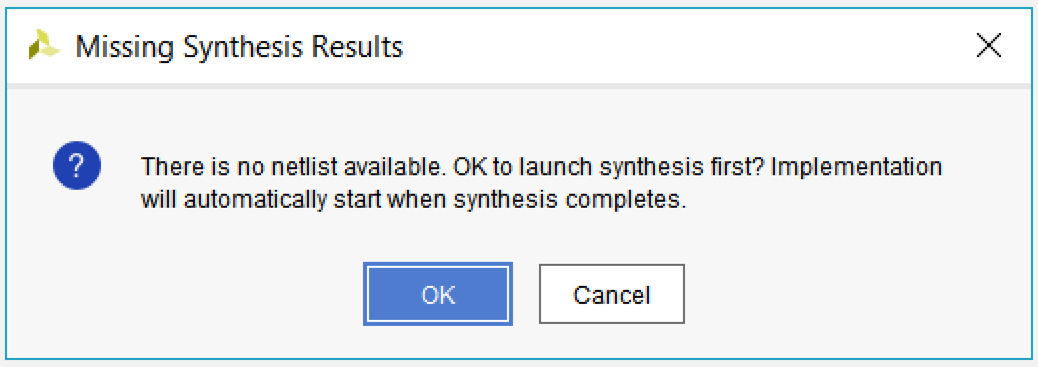
Before we move on, Vivado will want us to ensure all IPs are up-to-date. We can do this with the “Report IP Status” under “Reports” on the top of the navigation toolbar.



Upgrade any IPs that need upgrading before continuing.

We will now run both the Synthesis and Implementation of our design.
The Synthesis will auto-run prior to the Implementation, so we can go ahead and click on “Run Implementation” in the flow navigator to the left.

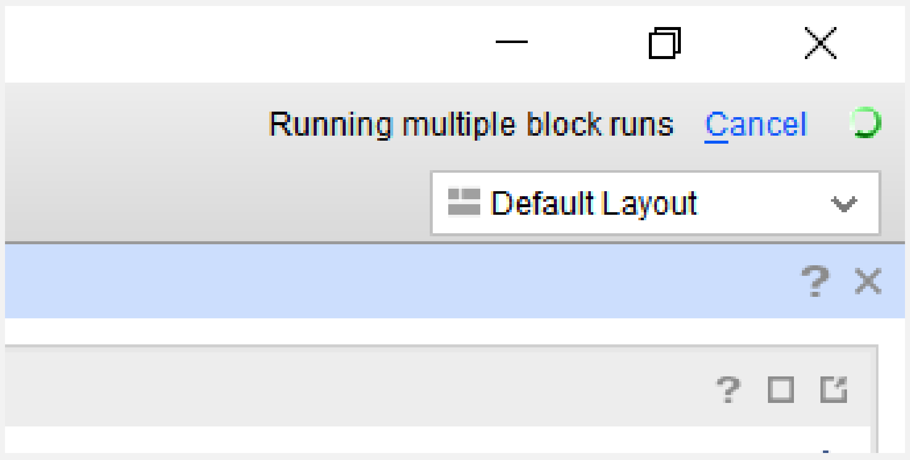



Missing Synthesis Results

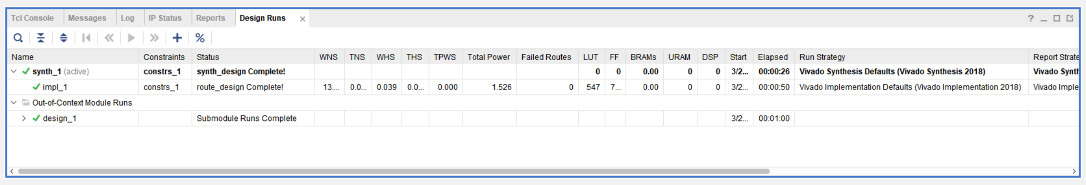
There is no netlist available. OK to launch synthesis first? Implementation will automatically start when synthesis completes.

Click OK to give Vivado permission to run the Synthesis first.

Vivado will begin to run both the Synthesis, and then the Implementation. This may take a while depending on your computer hardware. You can keep track of the progress on this process at the top right corner.

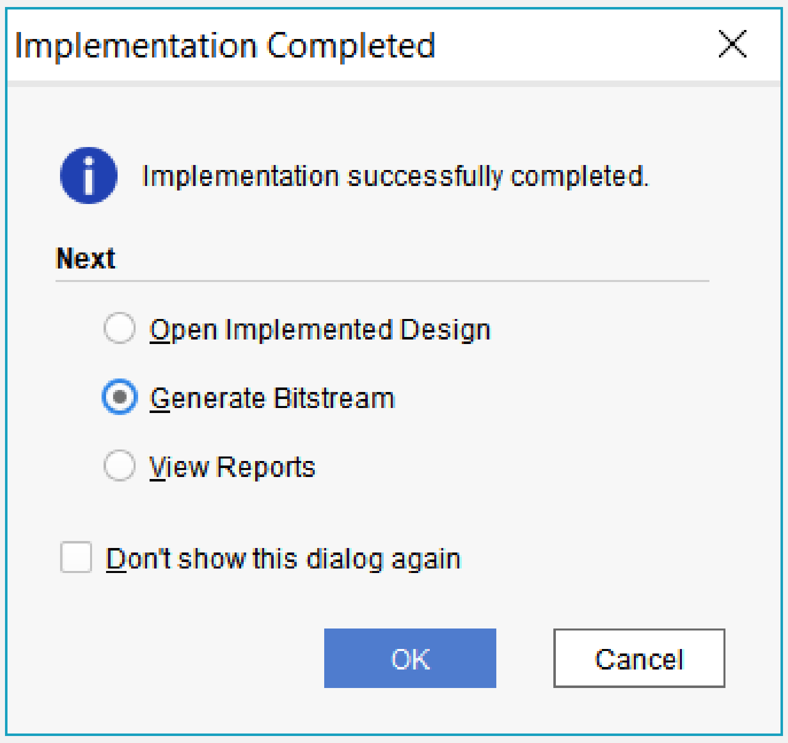


You may also check the “Design Runs” window at the bottom of Vivado.



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy	Report Strate
✓ synth_1 (active)	constrs_1	synth_design Complete!								0	0	0.00	0	0	3/2...	00:00:26	Vivado Synthesis Defaults (Vivado Synthesis 2018)	Vivado Synth
✓ impl_1	constrs_1	route_design Complete!	13....	0.0...	0.039	0.0...	0.000	1.526	0	547	7...	0.00	0	0	3/2...	00:00:50	Vivado Implementation Defaults (Vivado Implementation 2018)	Vivado Imple
Out-of-Context Module Runs																		
✓ design_1		Submodule Runs Complete													3/2...	00:01:00		

Upon successful Implementation completion, a pop-up window will appear. Click on "Generate Bitstream" and click OK.


Implementation Completed

Implementation successfully completed.

Next

- Open Implemented Design
- [✓ Generate Bitstream]
- View Reports

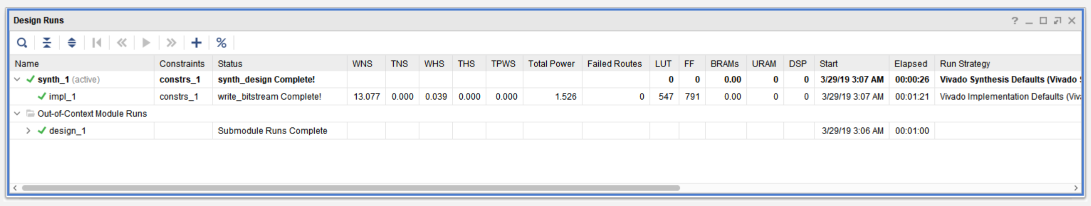
Don't show this dialog again

OK | Cancel

My Implementation/Synthesis failed!

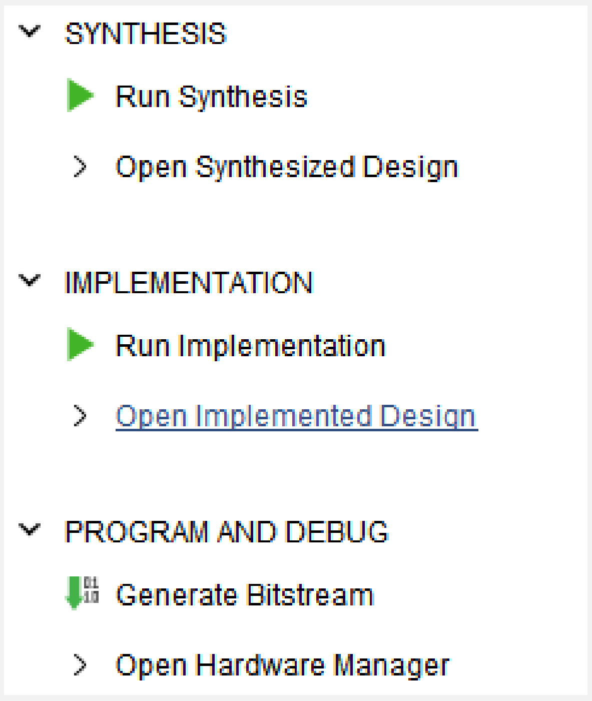
THE LEADING CAUSE OF THE AUTHOR'S MISTAKES THAT LED TO THE IMPLEMENTATION OR SYNTHESIS FAILURE WAS IMPROPER PIN DESIGNATIONS IN THE CONSTRAINTS FILE. DOUBLE CHECK YOUR CONSTRAINT FILES FOR ANY MISSPELLINGS OR INCONSISTENCIES. CAPITALIZATION IS IMPORTANT!

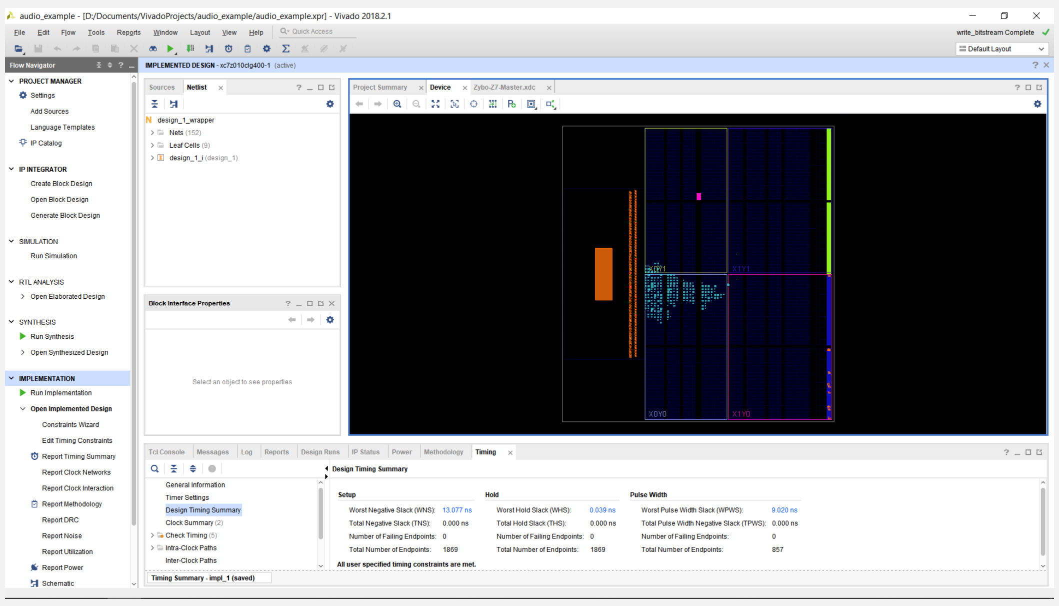
The Design Runs window will notify you when the bitstream is complete.


Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
synth_1 (active)	constrs_1	synth_design Complete!								0	0	0.00	0	0	3/29/19 3:07 AM	00:00:26	Vivado Synthesis Defaults (Vivado
impl_1	constrs_1	write_bitstream Complete!	13.077	0.000	0.039	0.000	0.000	1.526	0	547	791	0.00	0	0	3/29/19 3:07 AM	00:01:21	Vivado Implementation Defaults (Viv
Out-of-Context Module Runs																	
design_1		Submodule Runs Complete													3/29/19 3:06 AM	00:01:00	

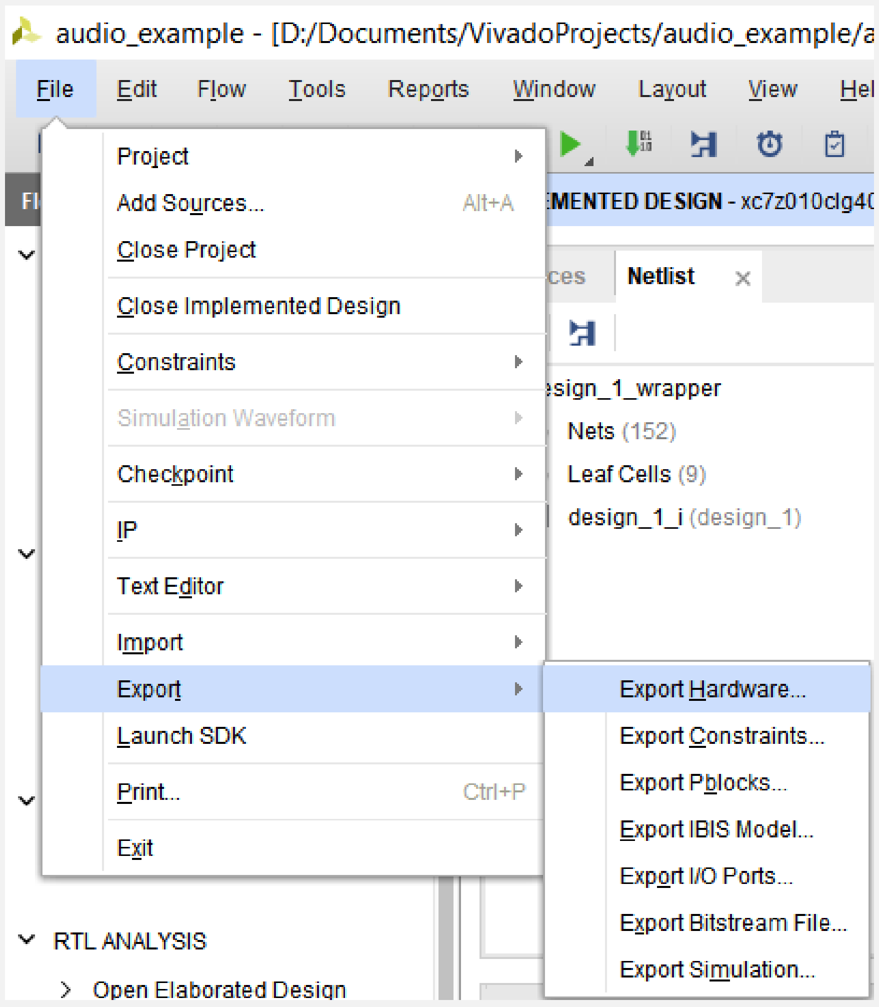
To finish up with the hardware portion of the project, click on "Open Implemented Design" under "IMPLEMENTATION" in the Flow Navigator.



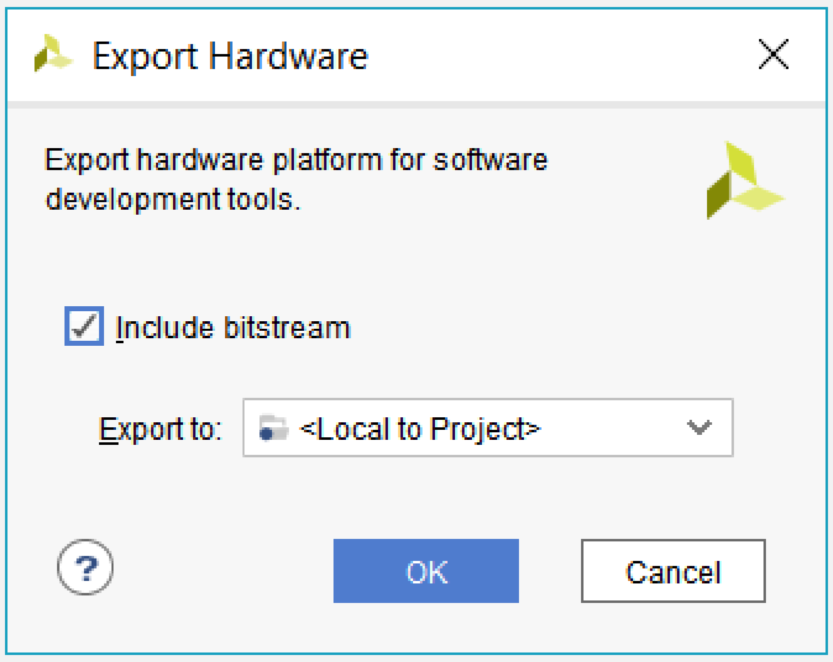


When the Device Layout window appears, shown above, we are ready to hand off the project to the SDK.

Click on File > Export > Export Hardware...



Be sure to include the bitstream.


Export Hardware

Export hardware platform for software development tools.

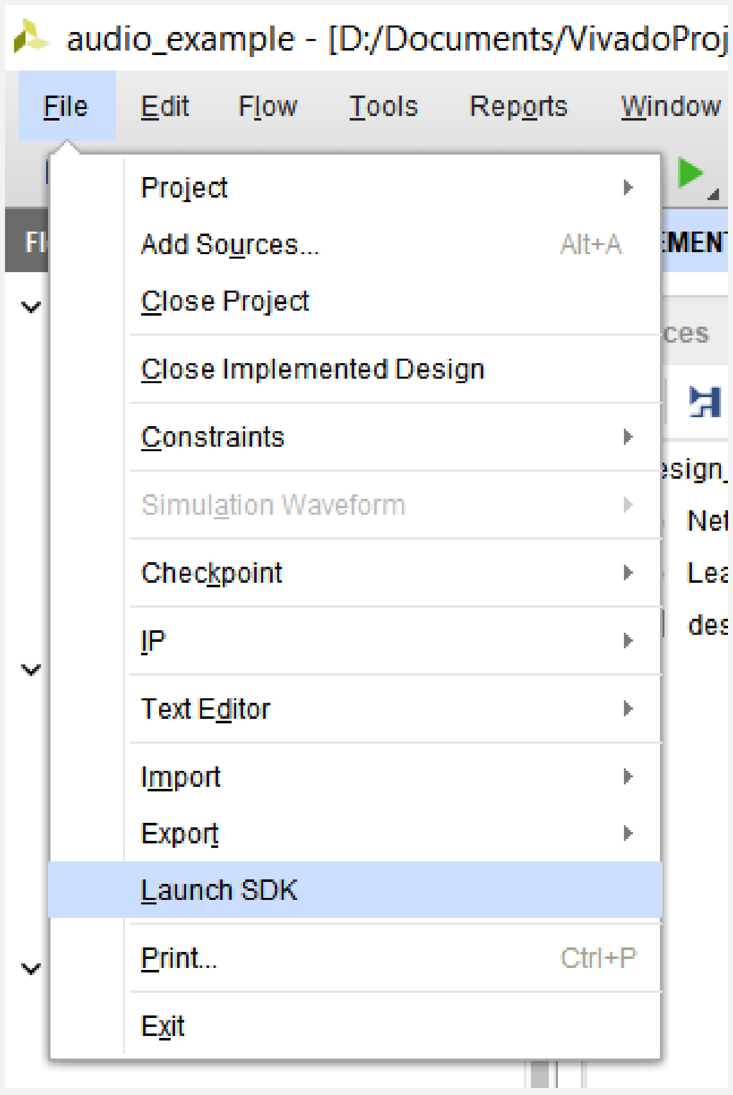
[✓] Include bitstream

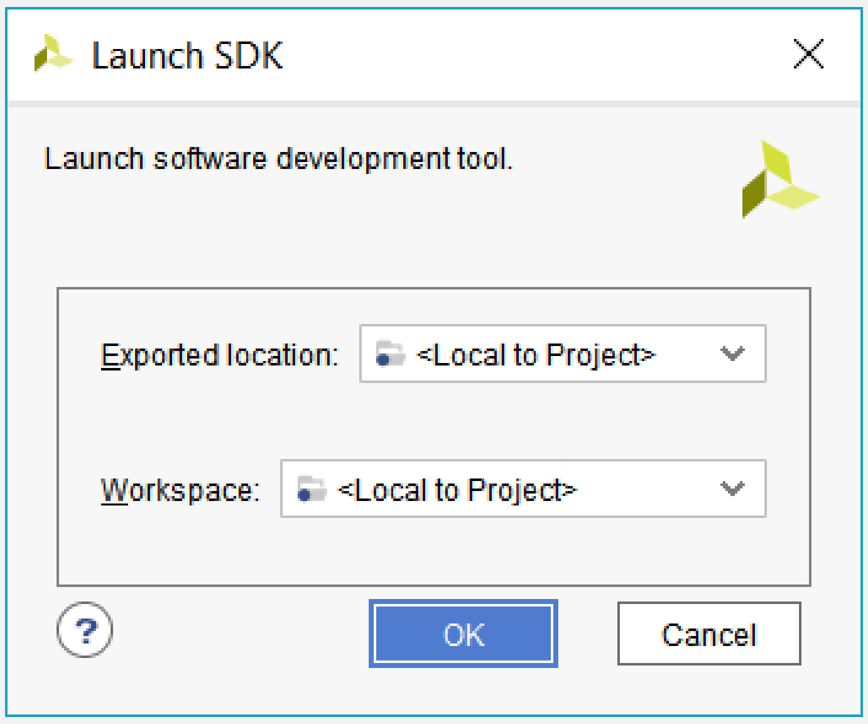
Export to: <Local to Project>

OK | Cancel

We are now ready to move on to the SDK.

Click on File > Launch SDK to continue.




Launch SDK

Launch software development tool.

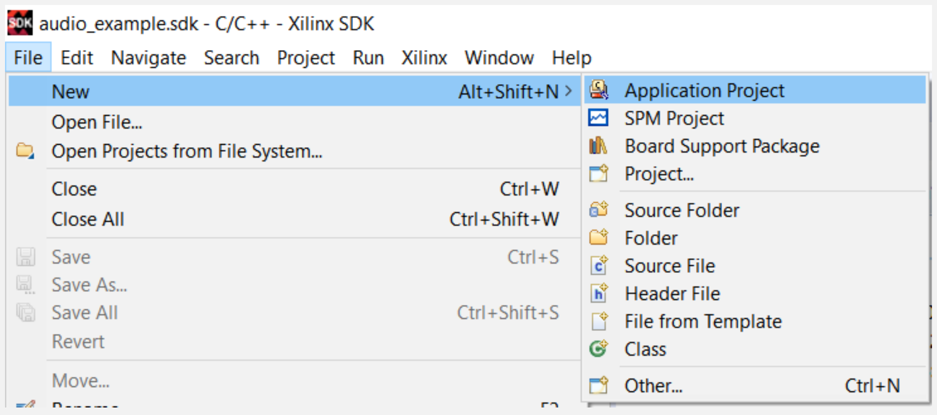
Exported location: <Local to Project>

Workspace: <Local to Project>

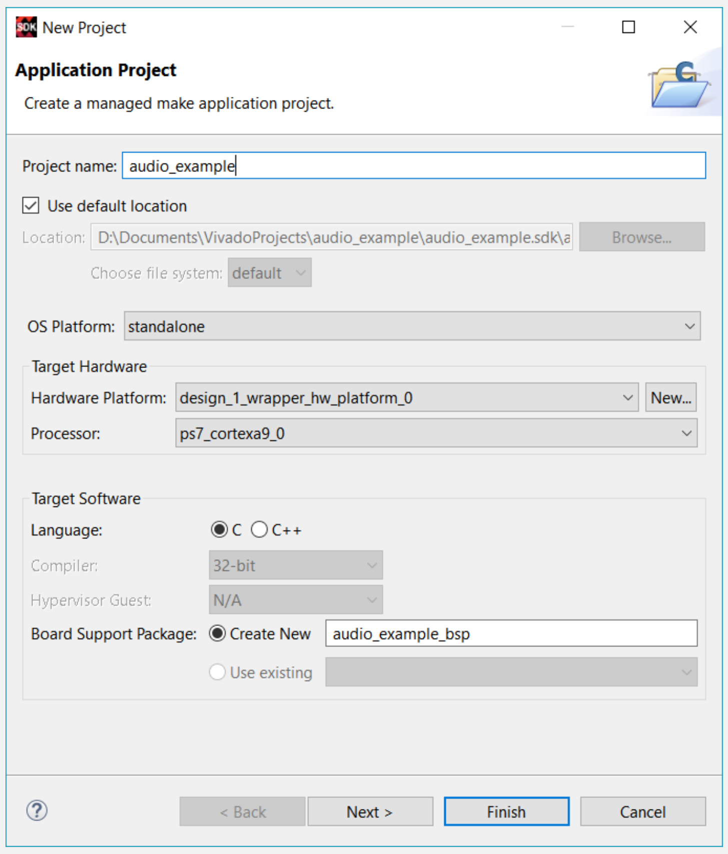
Special note: The Vivado SDK may not play nice with some anti-virus software. If the SDK does not open correctly or stalls out, check your AV software.

Vivado SDK

We will begin by creating a new Application Project.



We will be naming our SDK project "audio_example"


New Project

Application Project

Create a managed make application project.

Project name: audio_example

[✓] Use default location

Location: D:\Documents\VivadoProjects\audio_example\audio_example.sdk\a

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: design_1_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Target Software

Language: [✓ C] C++

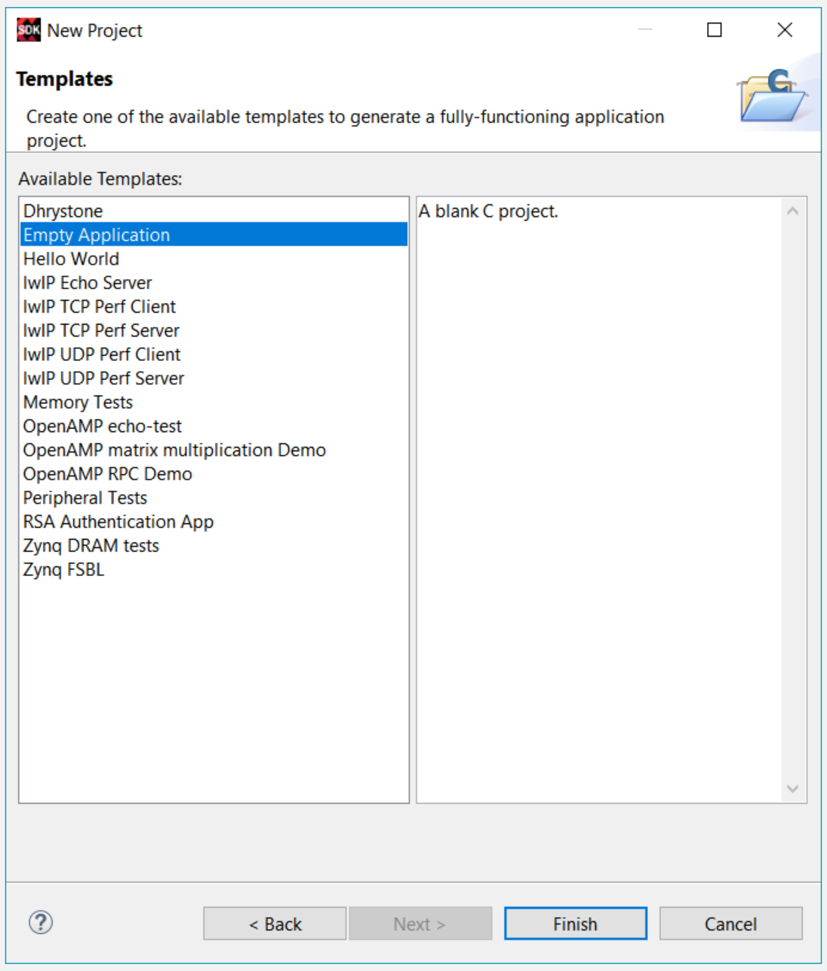
Compiler: 32-bit

Hypervisor Guest: N/A

Board Support Package: [✓ Create New] audio_example_bsp Use existing

Click on Next.

You may choose any of the available templates, but for our tutorial, we will be starting with the "Empty Application" template. Click on Finish.


New Project

Templates

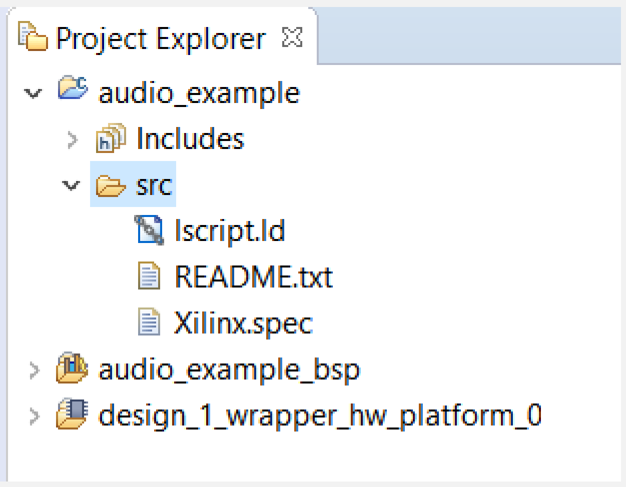
Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- lwIP TCP Perf Client
- lwIP TCP Perf Server
- lwIP UDP Perf Client
- lwIP UDP Perf Server
- Memory Tests
- OpenAMP echo-test
- OpenAMP matrix multiplication Demo
- OpenAMP RPC Demo
- Peripheral Tests
- RSA Authentication App
- Zynq DRAM tests
- Zynq FSBL

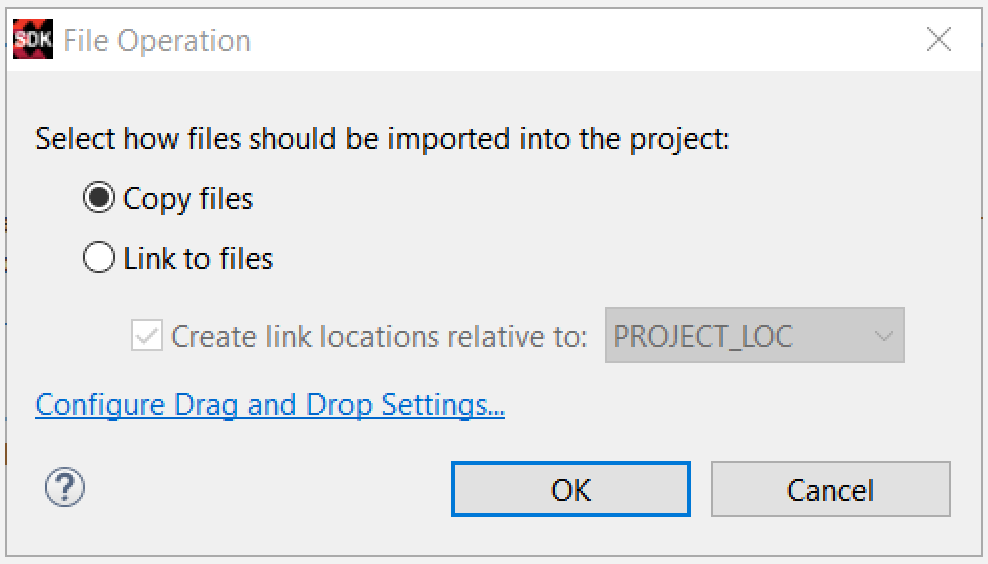
A blank C project.

In the project explorer on the left, there should be three "projects" open. Click on the drop-down arrow for the project we created and navigate to the "src" folder. It is here where we will be creating our C code (.c) files and header (.h) files.



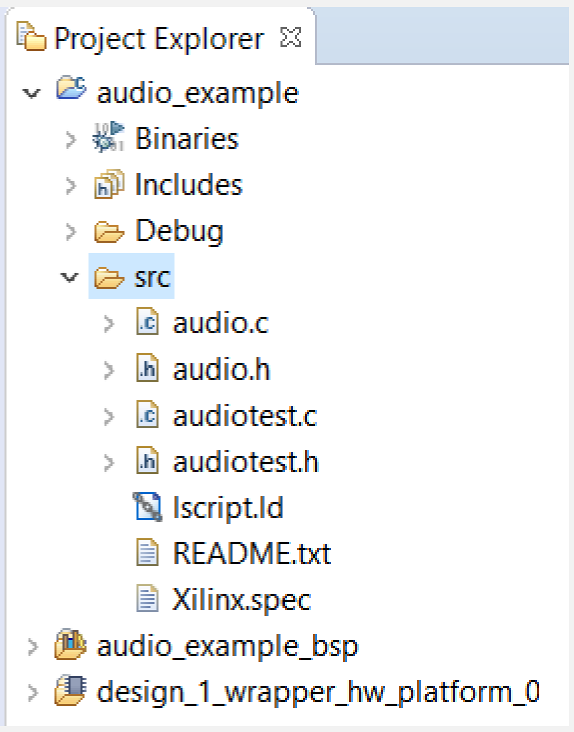
The files we will be using are a modified version of the files located within the source download at the Zynq Book website. The source code they provide has multiple parts that are dependent on IP that are not present in our project, thus the author has stripped out most of what is unnecessary and has left the relevant parts for the audio.
These files are: audio.c, audio.h, audiotest.c, and audiotest.h

Extract the files and drag and drop it into the /src folder for our project.
There will be a File Operation dialog box that pops up. We will want to create a copy of the file for our specific project.


File Operation
Select how files should be imported into the project:
Copy files
Link to files
Create link locations relative to: PROJECT_LOC
Configure Drag and Drop Settings...
OK
Cancel

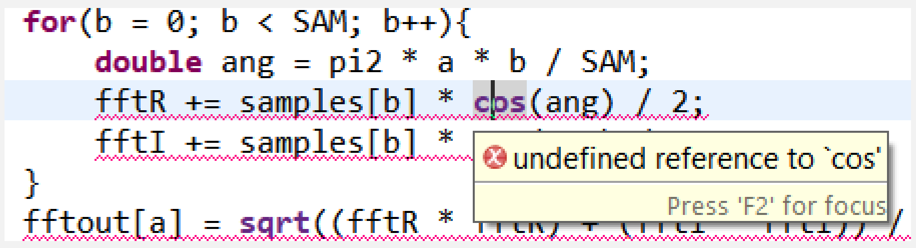
Click on "Copy files" and click on OK.

The Project Explorer should look similar to this:



Double click on "audiotest.c", which contains our 'main()', to open it in the editor.

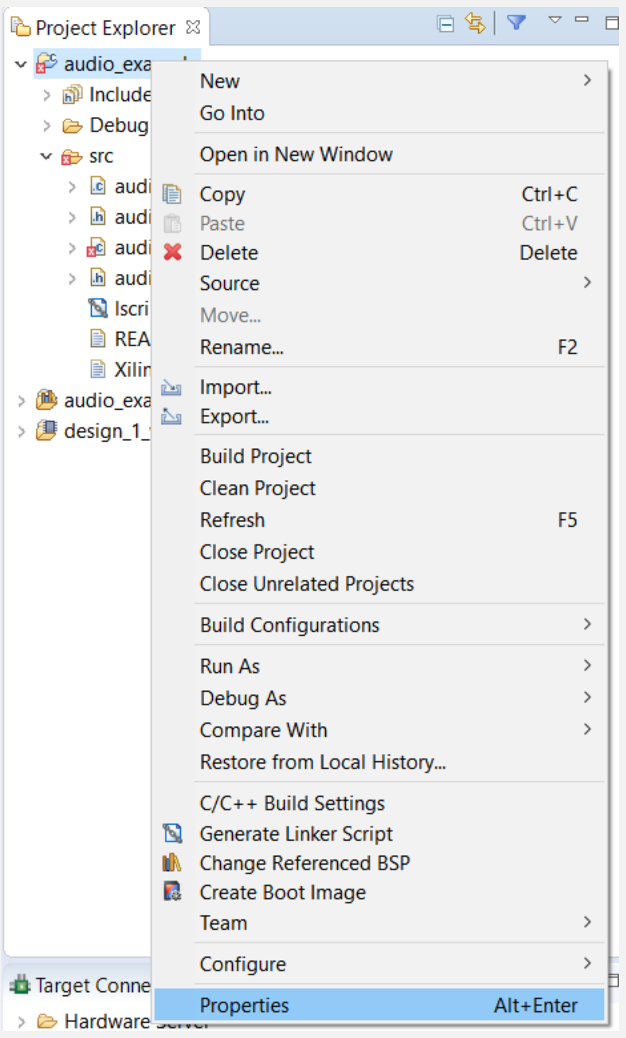
Most of the code will be commented to walk you through the algorithms present. Though the code includes the 'math.h' header file for the math function calls involved in the program, the device itself requires a different method for including it in the compiler.



```
for(b = 0; b < SAM; b++){
    double ang = pi2 * a * b / SAM;
    fftR += samples[b] * cos(ang) / 2;
    fftI += samples[b] *
}
fftout[a] = sqrt((fftR *
```

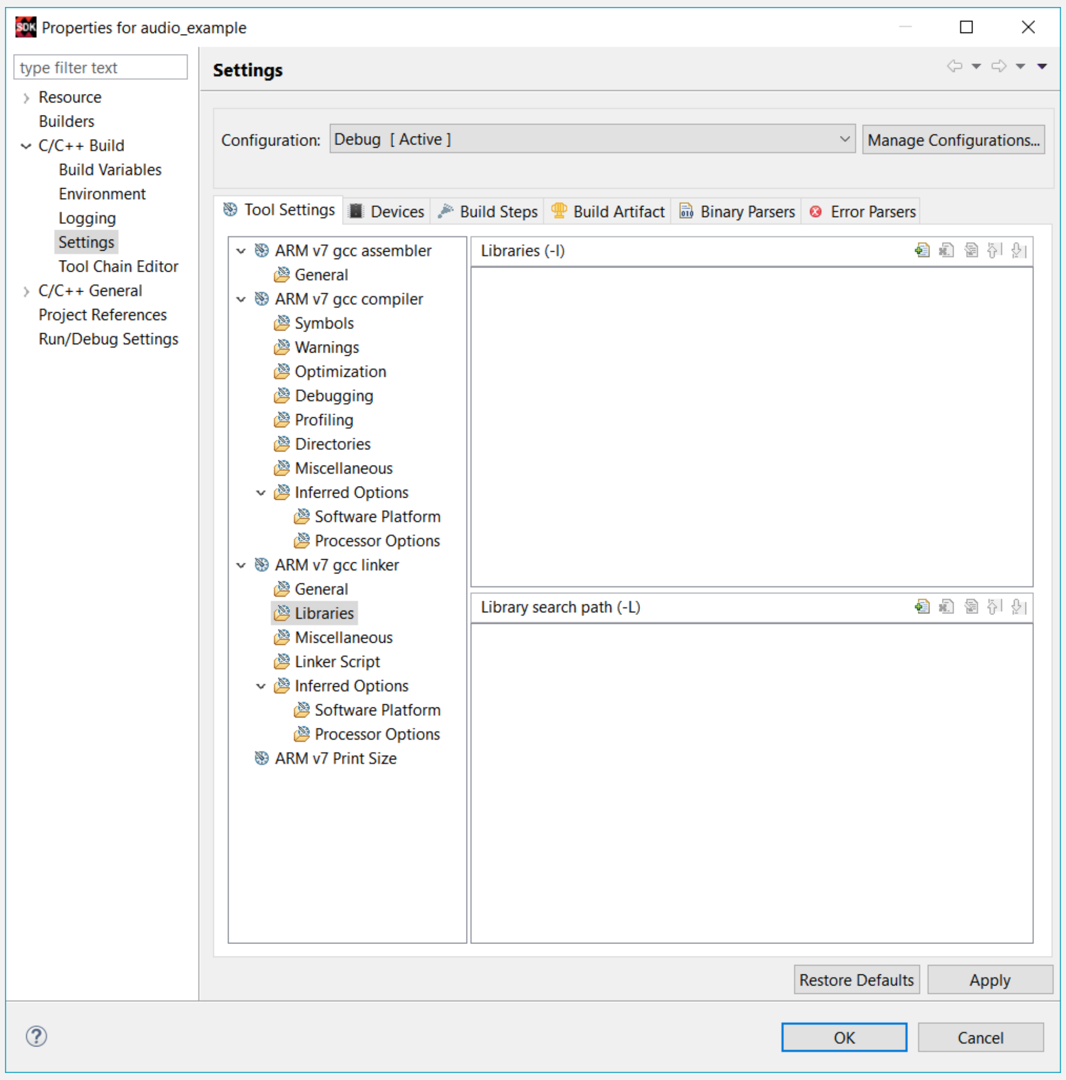
The error associated with the missing header file will appear as "undefined reference to [math function]"

In order to properly import the math functions required, we will have to make adjustments to the project options.



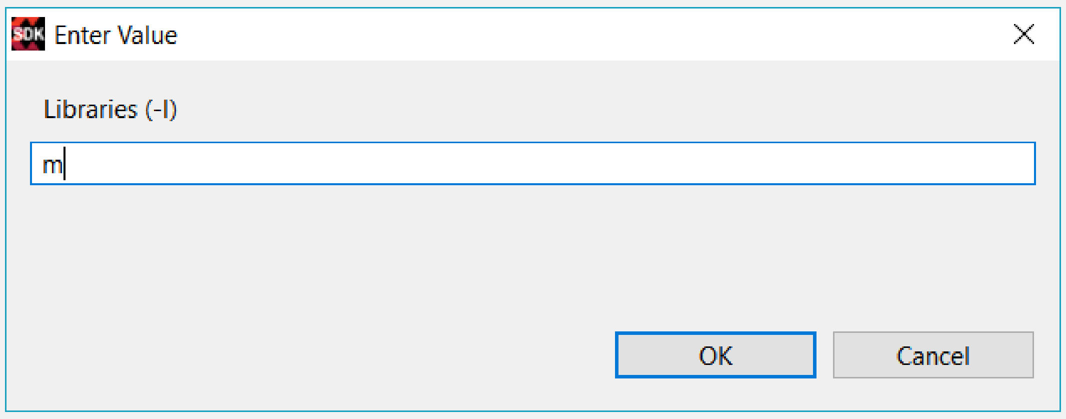
Right click on “audio_example” in the Project Explorer and click on “Properties” at the bottom of the list.

Navigate to “C/C++ Build -> Settings -> Tool Settings -> ARM v7 gcc linker -> Libraries”



Configuration: Debug [Active]

In the bar to the right of “Libraries (-l)” click on “Add…”

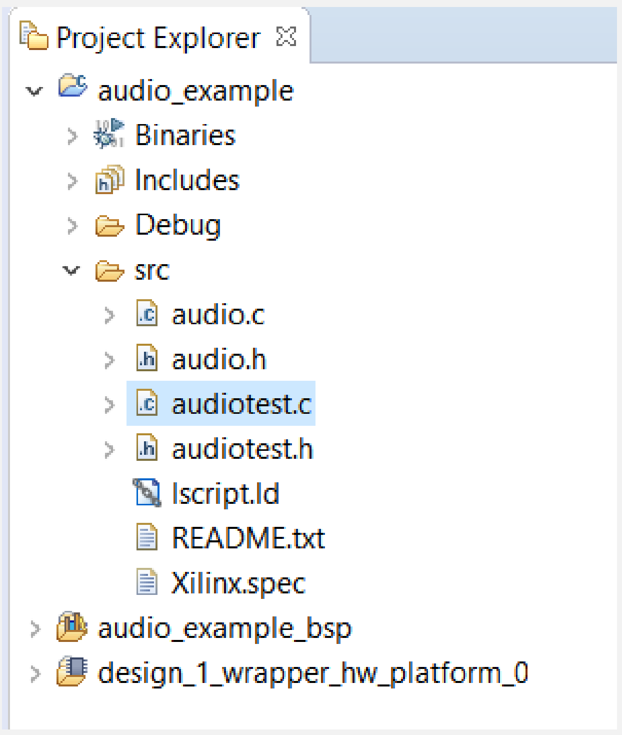


Libraries (-l)

m

Type the letter “m” and press OK.
Click on “Apply” and “OK” to close the properties window.

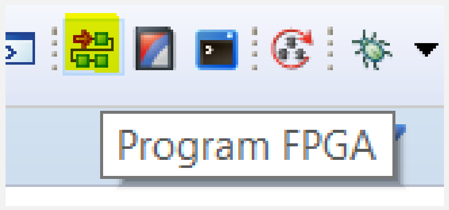
The project should rebuild itself with the math library now included, removing all previous errors present in the code. If the project did not rebuild, you can simply hit ‘Ctrl+S’ with the “audiotest.c” file window active to save the file and rebuild the project.



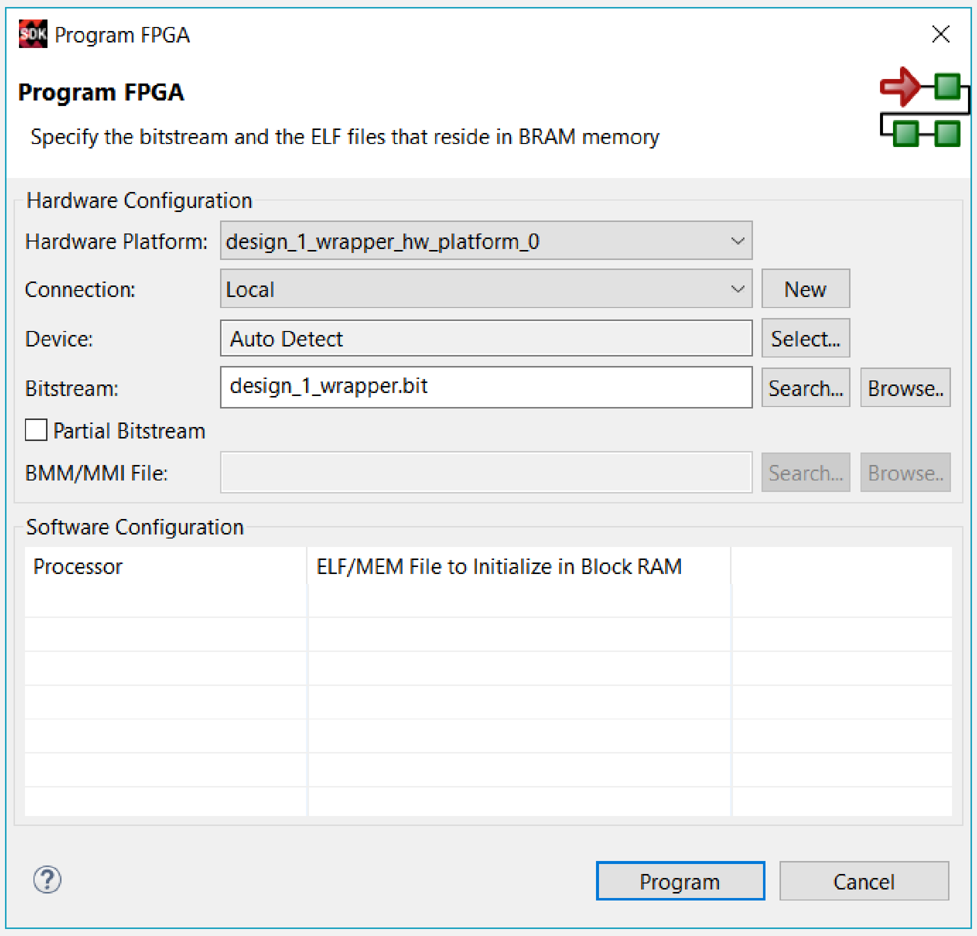
We will now run the code on the Zybo FPGA.

First, ensure that the FPGA is connected to your computer and that all jumpers are set properly (JTAG on JP5, appropriate power source next to the power switch). Turn on the FPGA device. The red ‘PGOOD’ light (LD13) should turn on.

In the SDK, click on the Program FPGA button on the top menu.



The default options should be correct. Reference your settings to the image below. Click on “Program” at the bottom of the window.



Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

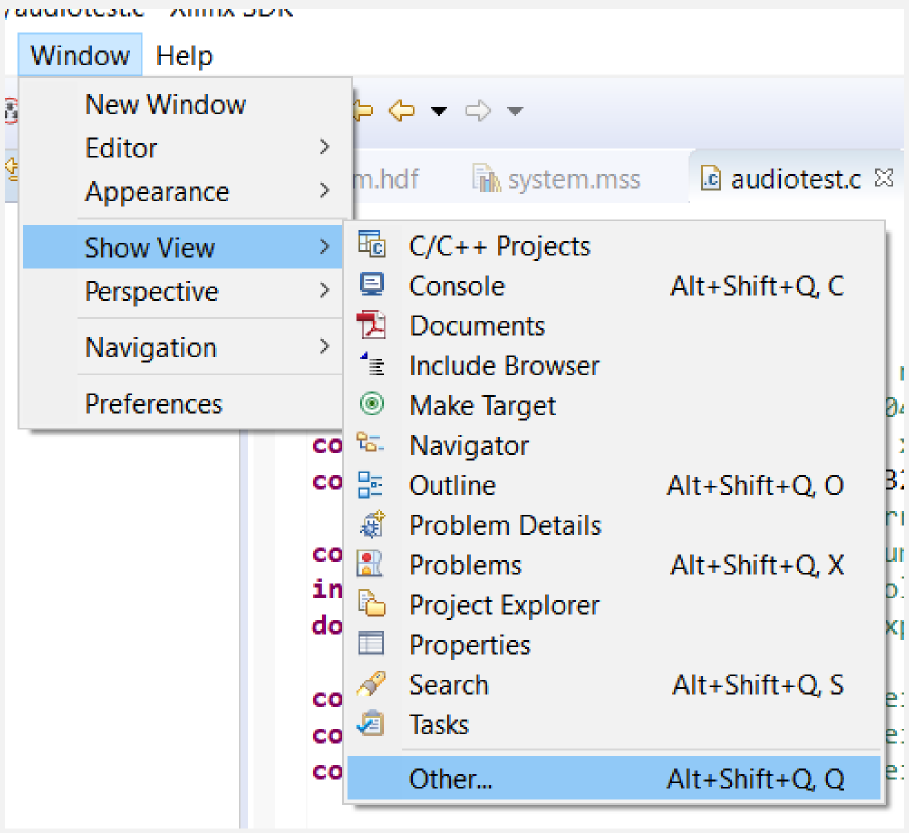
Hardware Platform:	design_1_wrapper_hw_platform_0
Connection:	Local
Device:	Auto Detect
Bitstream:	design_1_wrapper.bit
☐ Partial Bitstream	
BMM/MMI File:	

Software Configuration

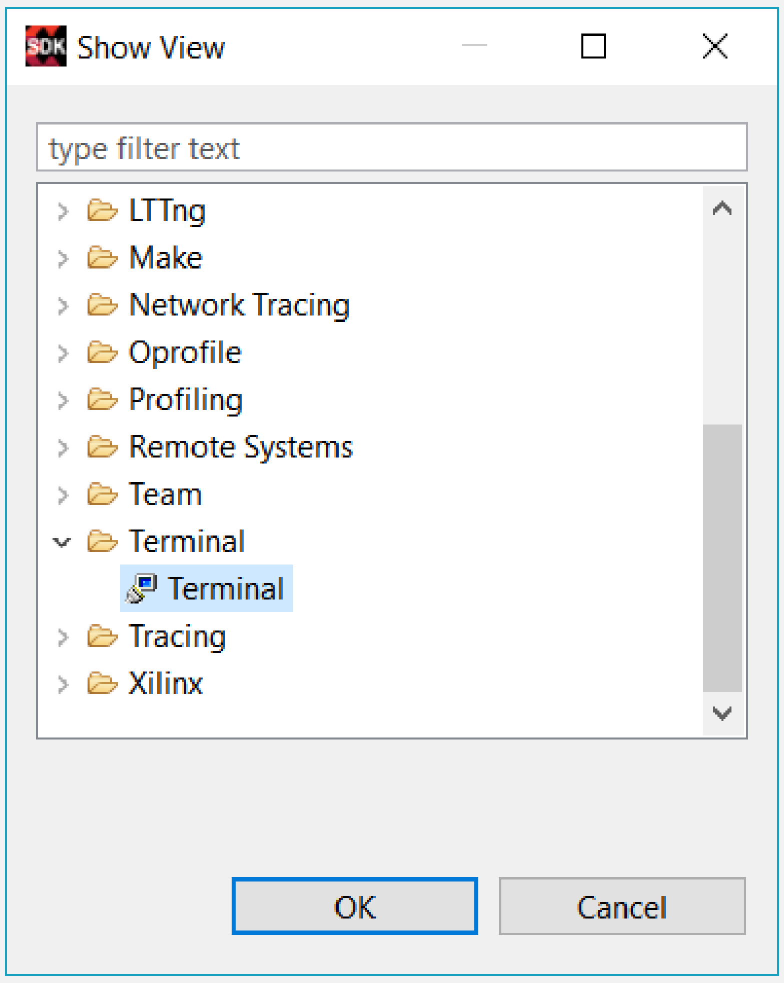
Processor	ELF/MEM File to Initialize in Block RAM

If the FPGA has been properly programmed, the green DONE light (LD12) should light up.
We are now ready to launch our software on the FPGA.

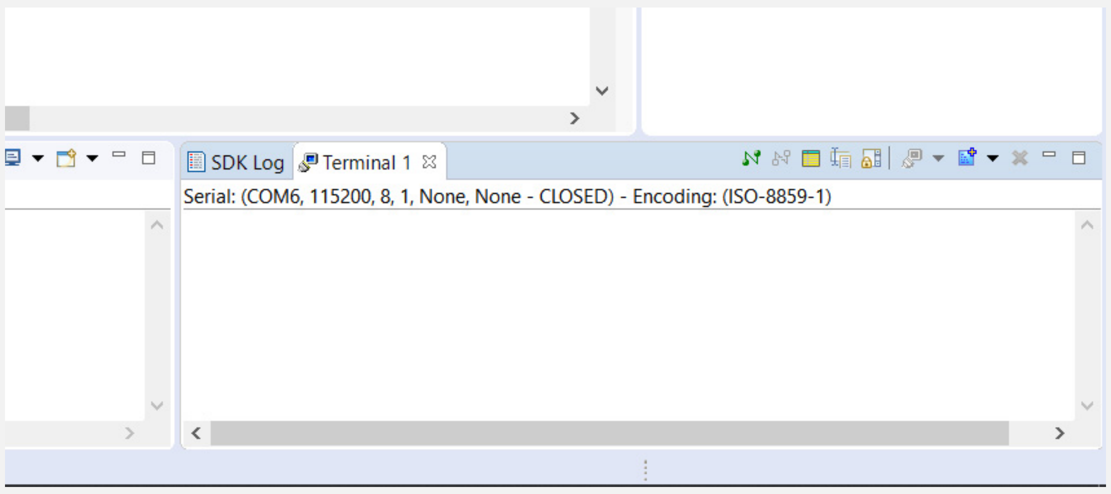
First, let us open the SDK Terminal Window to get the messages from the FPGA.



Navigate to "Window -> Show View -> Other..." or press Alt+Shift+Q, then Q, to open the Show View window. Under "Terminal", double click on "Terminal."

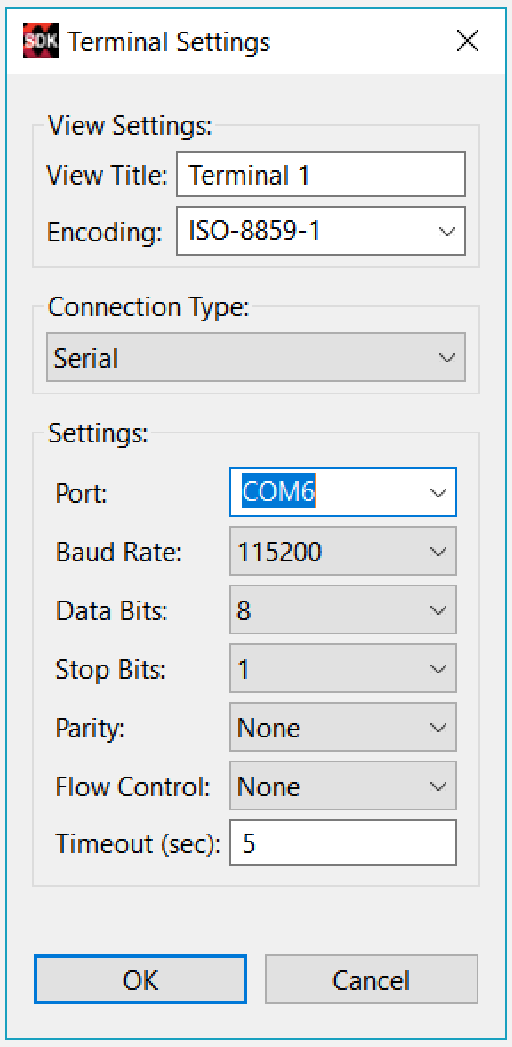


This should open the Terminal window at the bottom of your screen. Click on the settings button to adjust the settings of the Terminal window.



Serial: (COM6, 115200, 8, 1, None, None - CLOSED) - Encoding: (ISO-8859-1)

Upon changing the connection type to Serial, the settings shown in the image below should be the default settings for the Terminal.



View Settings:

View Title:	Terminal 1
Encoding:	ISO-8859-1

Connection Type: Serial

Settings:

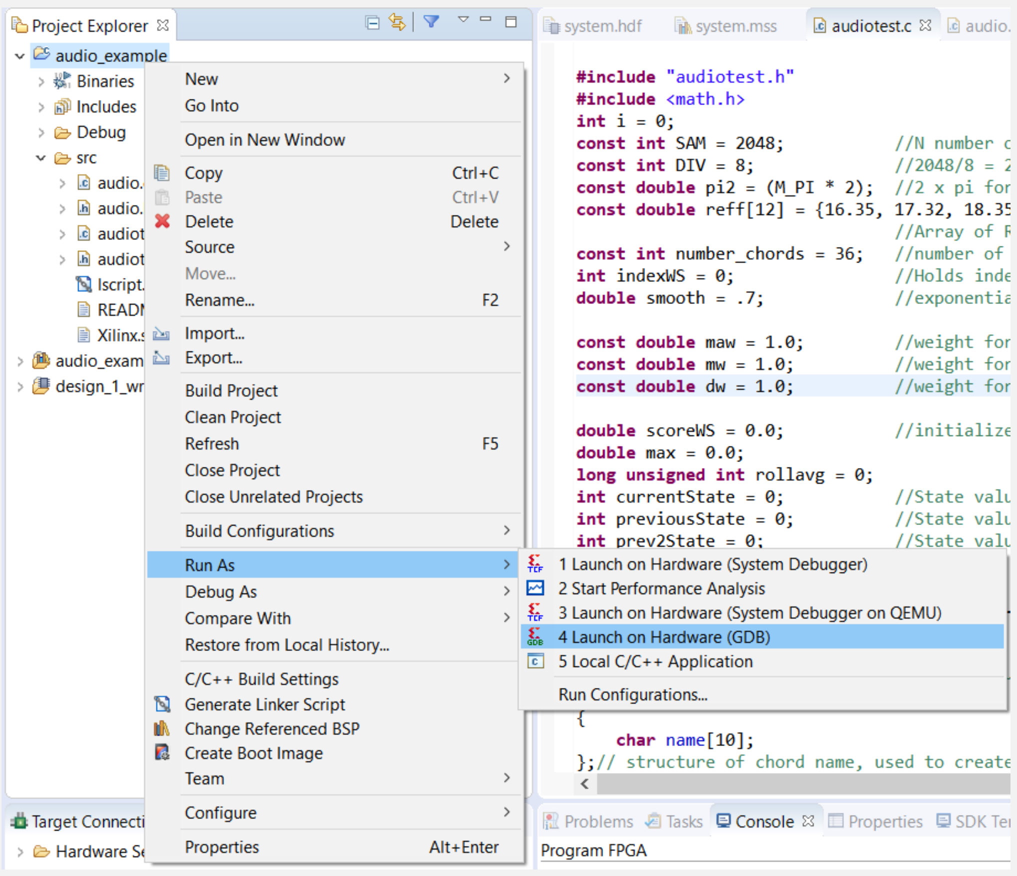
Port:	COM6
Baud Rate:	115200
Data Bits:	8
Stop Bits:	1
Parity:	None
Flow Control:	None
Timeout (sec):	5

Of note is the Port option. The Zybo FPGA opens two communication ports to the computer. The first port (COM3, COM5, etc.) is used to program the FPGA. The second port (COM4, COM6, etc.) is used to communicate with the software running on the device. On the author's computer, COM6 is the port used to communicate with the software running on the Zybo.

Pressing OK on the Terminal Settings should automatically begin the connection with the FPGA.

Now that the FPGA is programmed with the bitstream and the communication terminal is connected, we will be able to see if our program is running properly. For this program to work, it should be noted that a microphone should be attached to the Zybo FPGA's pink MIC IN connection.

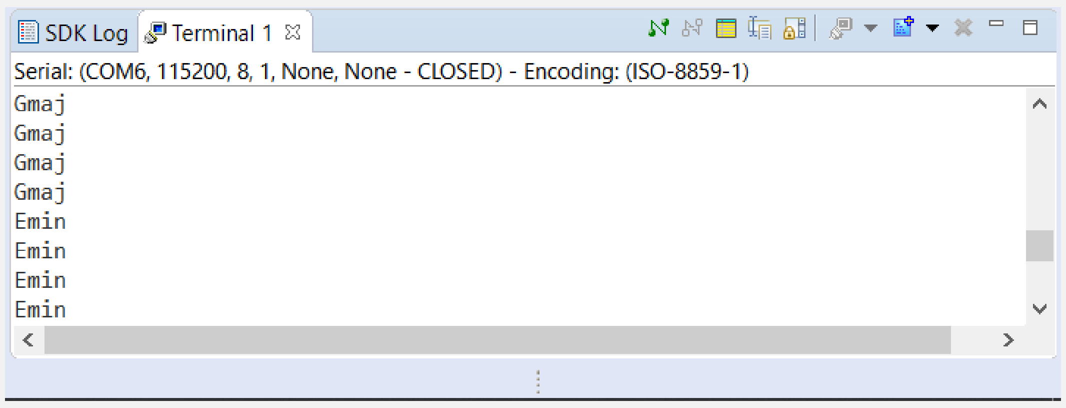
Right click again on the "audio_example" project in the Project Explorer. We will want to Launch this project on the Hardware. Navigate to "Run As -> 4 Launch on Hardware (GDB)"



```
#include "audiotest.h"
#include <math.h>
int i = 0;
const int SAM = 2048;             //N number c
const int DIV = 8;                //2048/8 = 2
const double pi2 = (M_PI * 2);   //2 x pi for
const double reff[12] = {16.35, 17.32, 18.35
                                  //Array of F
const int number_chords = 36;    //number of
int indexWS = 0;                  //Holds inde
double smooth = .7;               //exponentia

const double maw = 1.0;           //weight for
const double mw = 1.0;            //weight for
const double dw = 1.0;            //weight for

double scoreWS = 0.0;             //initialize
double max = 0.0;
long unsigned int rollavg = 0;
int currentState = 0;             //State valu
int previousState = 0;            //State valu
int prev2State = 0;               //State valu
{
    char name[10];
};// structure of chord name, used to create
```

```
Program FPGA
```

If everything is working as expected, then the Terminal window should begin to display the project's outputs.



```
Serial: (COM6, 115200, 8, 1, None, None - CLOSED) - Encoding: (ISO-8859-1)
Gmaj
Gmaj
Gmaj
Gmaj
Emin
Emin
Emin
Emin
```

It is advised to use a different Terminal window to view the project's outputs. The SDK terminal window is great for debugging purposes, but it is lacking in versatility. TeraTerm is a free program that can also pick up the output from the Zybo upon proper configuration.

Due to the nature of the code implemented on the Zybo, it is necessary to play chords or complex note combinations to receive an output that differs from "nothing detected." A guitar, or playing a song to the microphone should have it pick up different readings.

The core of the work with audio on the Zybo FPGA will stem from collecting the samples from the on-board SSM2603 Audio Codec; all of the configuration for the codec is found in the audio.c file. The audiotest.c code is the author's example on processing the samples using complex algorithms.

This tutorial thus far has been on programming the Zybo FPGA to process audio signals. When it comes time to finalize your project, loading your software to boot off of an SD card will be necessary to remove the restrictions of having to connect to a host computer running the SDK. We will now briefly cover the SD Boot process.

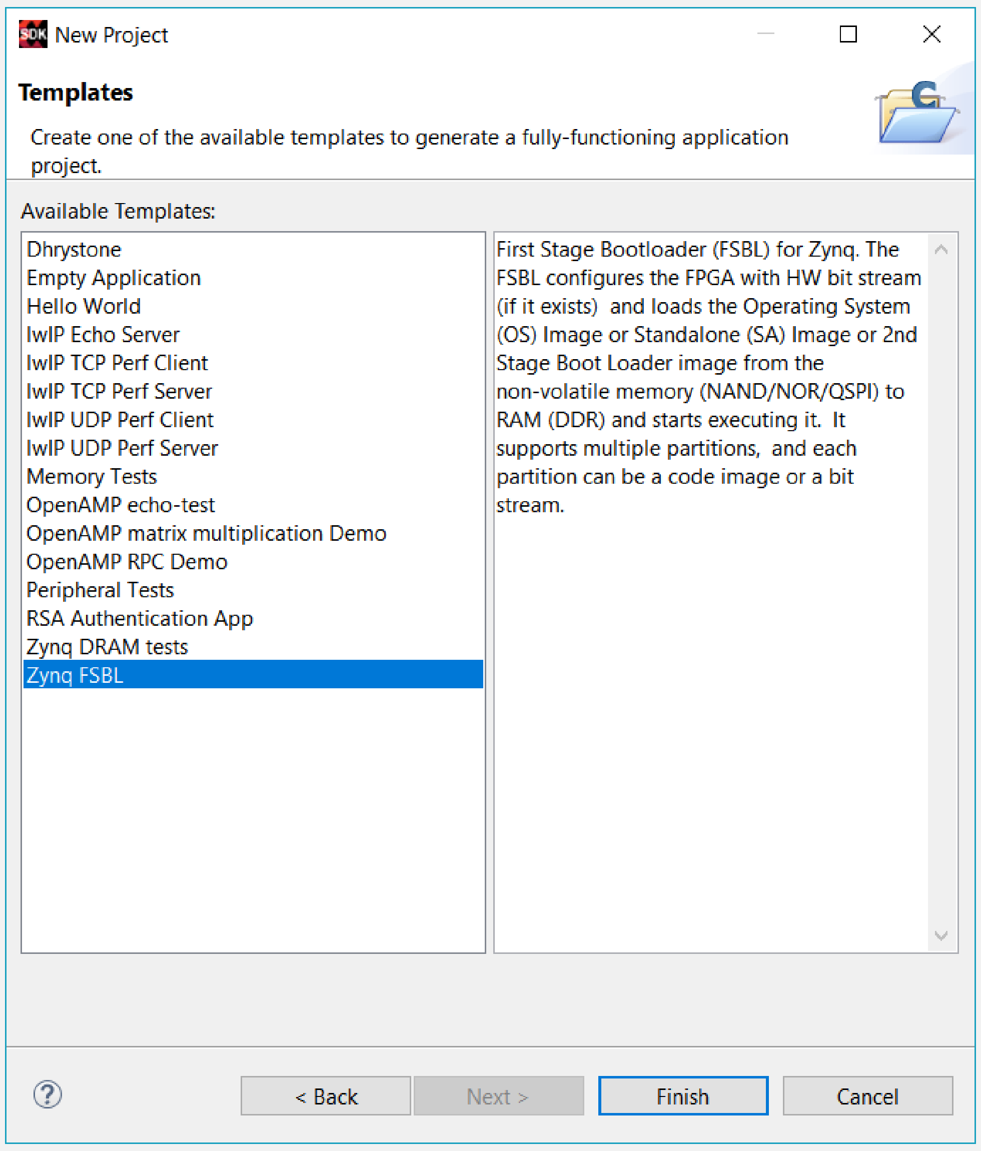
SD Boot

Booting from the SD card will be necessary to remove the need for a host computer.

We will begin in SDK by creating a new First Stage Boot Loader application project.

Navigate to “File -> New -> Application Project” We will name ours “fsbl_example.” The default options will suffice.

Click on Next, and choose Zynq FSBL


New Project

Templates

Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- lwIP TCP Perf Client
- lwIP TCP Perf Server
- lwIP UDP Perf Client
- lwIP UDP Perf Server
- Memory Tests
- OpenAMP echo-test
- OpenAMP matrix multiplication Demo
- OpenAMP RPC Demo
- Peripheral Tests
- RSA Authentication App
- Zynq DRAM tests
- Zynq FSBL

First Stage Bootloader (FSBL) for Zynq. The FSBL configures the FPGA with HW bit stream (if it exists) and loads the Operating System (OS) Image or Standalone (SA) Image or 2nd Stage Boot Loader image from the non-volatile memory (NAND/NOR/QSPI) to RAM (DDR) and starts executing it. It supports multiple partitions, and each partition can be a code image or a bit stream.

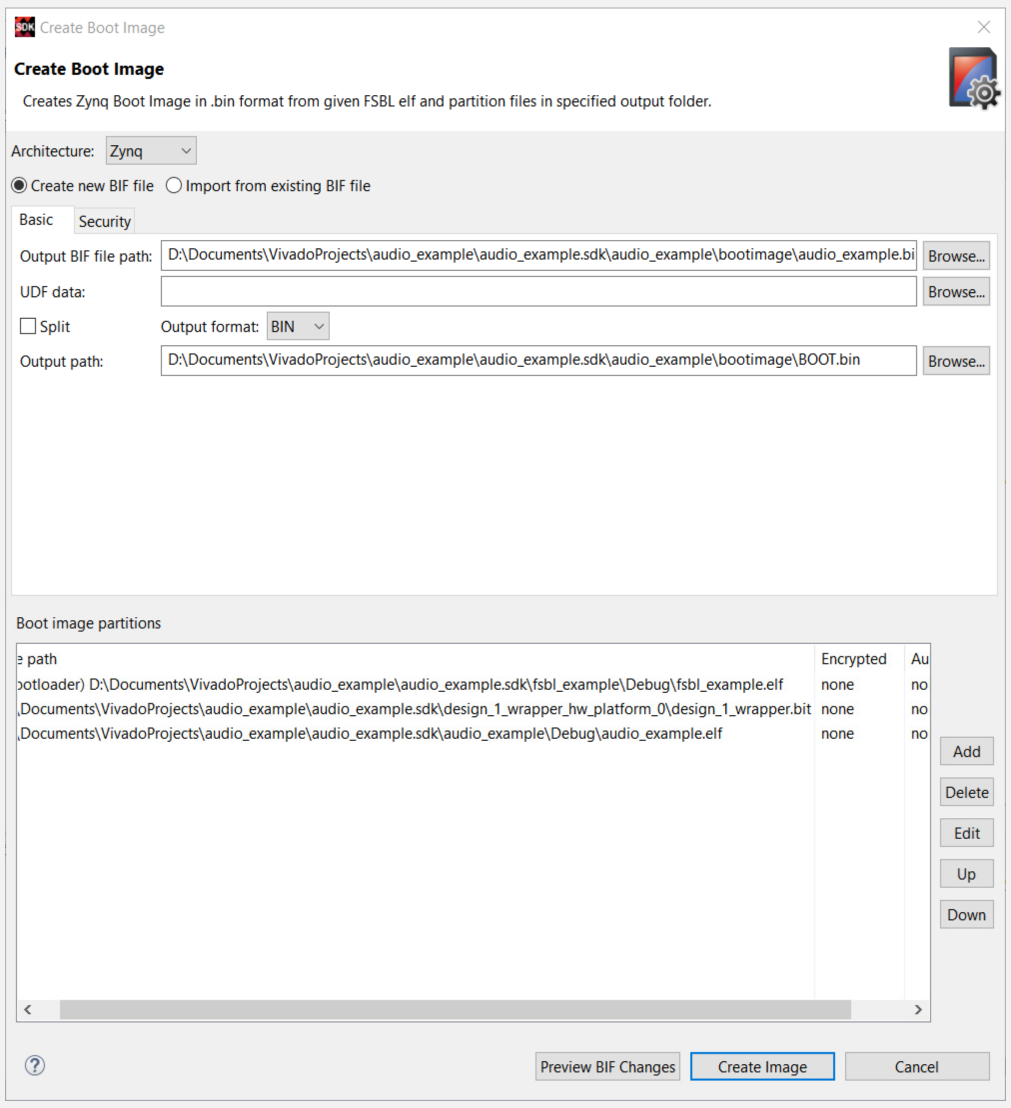
Click on Finish

The SDK will create the first stage boot loader based off of the bitstream that was handed off by the Vivado software.

The FSBL does not need to be edited.

Right click on your project “audio_example” and choose “Create Boot Image.”

Pay special attention to the output path, as we will have to navigate there to find the BOOT.bin file that will be copied onto the micro SD card.


Create Boot Image

Create Boot Image

Creates Zynq Boot Image in .bin format from given FSBL elf and partition files in specified output folder.

Architecture: Zynq

Create new BIF file [✓] Import from existing BIF file

Basic Security

Output BIF file path: D:\Documents\VivadoProjects\audio_example\audio_example.sdk\audio_example\bootimage\audio_example.bi

UDF data:

Split Output format: BIN

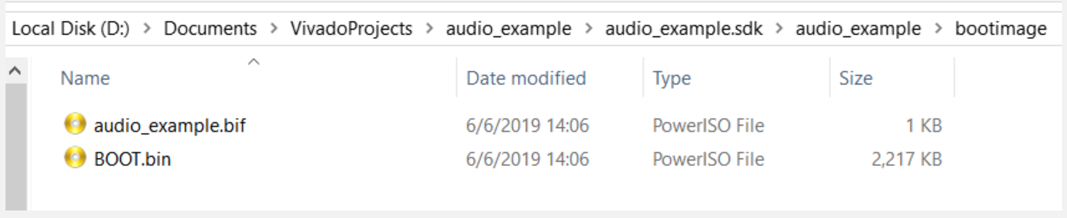
Output path: D:\Documents\VivadoProjects\audio_example\audio_example.sdk\audio_example\bootimage\BOOT.bin

Boot image partitions

e path	Encrypted	Au
otloader) D:\Documents\VivadoProjects\audio_example\audio_example.sdk\fsbl_example\Debug\fsbl_example.elf	none	no
Documents\VivadoProjects\audio_example\audio_example.sdk\design_1_wrapper_hw_platform_0\design_1_wrapper.bit	none	no
Documents\VivadoProjects\audio_example\audio_example.sdk\audio_example\Debug\audio_example.elf	none	no

Under Boot image partitions, ensure that there are three files. The order in which they are listed is also important. The topmost file should be the “fsbl_example.elf” bootloader file. The second should be the bitstream wrapper for the project. The last will be the “audio_example.elf” file associated with the project we created.

Click on “Create Image” and the SDK will create the Boot image.


Local Disk (D:) › Documents › VivadoProjects › audio_example › audio_example.sdk › audio_example › bootimage

Name	Date modified	Type	Size
audio_example.bif	6/6/2019 14:06	PowerISO File	1 KB
BOOT.bin	6/6/2019 14:06	PowerISO File	2,217 KB

Navigating to the folder where the output was created, we should have two files created. The ‘.bif’ file and the ‘.bin’ file.

Copying the BOOT.bin file to an SD card, inserting the SD card to the FPGA, setting the jumper from JTAG to SD, and powering the FPGA should automatically run the project we uploaded into it upon booting. Using the same programming cable to connect to a PC and connecting to the FPGA through a terminal should allow us to read the outputs of the device.

This concludes our brief tutorial on booting from an SD card.