**Working with the Mono Audio**

Goal

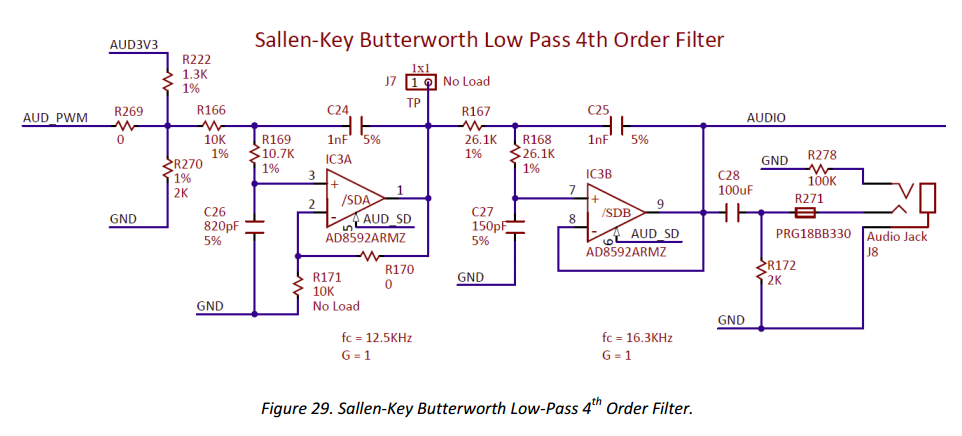
The objective of this tutorial is to utilize the mono audio jack on the Nexys 4 DDR board to produce an audible square wave. This tutorial is divided into three main sections: PWM IP, Hardware Block Design and SDK Software Design.

Requirements

* Xilinx Vivado software
* Xilinx SDK software
* Xilinx Nexys 4 board and a programming cable
* Speaker/Headphone
* Enough disk space for the project files

Background

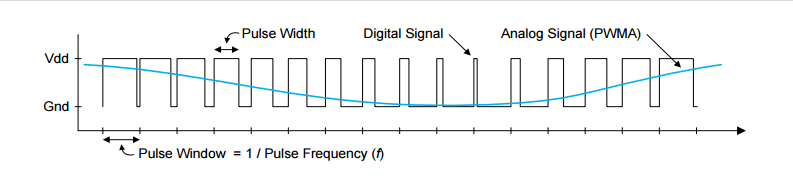
Analog audio signals are encoded in PCM (Pulse Code Modulation) format which is used by WAV files on your PC. In this project, a number of bits are written to the PWM (Pulse Width Modulation) module each cycle. The number of bits defines the audio resolution (how clean/accurate it sounds). PCM data is stored in the DDR memory and then PWM module reads the data from memory and transfers data into 1-bit PWM signal. The AUD\_PWM signal is connected to Port A11 on the Nexys 4 board. Port A11 is then connected to a 4th order low pass filter as shown in Figure 1 below:



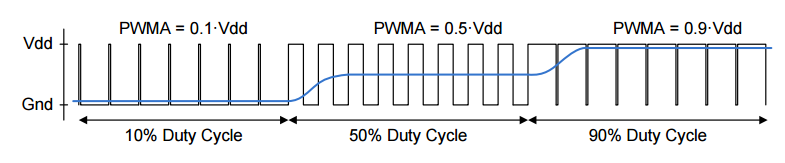
*Figure1. Low Pass Filter on Nexys4 DDR Board*

1. Producing the PWM IP

In this section, we will create a simple AXI Stream slave IP that will produce the digital input to Port A11. A pulse-width modulated (PWM) signal is a chain of pulses at some fixed frequency with each pulse potentially having different width. For example, if the pulses are high for an **average** of 10% of the available pulse period, then an integrator will produce an analog value that is 10% of the Vdd voltage. Figure 2 shows an example of a waveform represented as a PWM signal. Figure 3 shows examples of output voltage with respect to different average PWM signals.



*Figure 2. Waveform Represented as a PWM Signal*



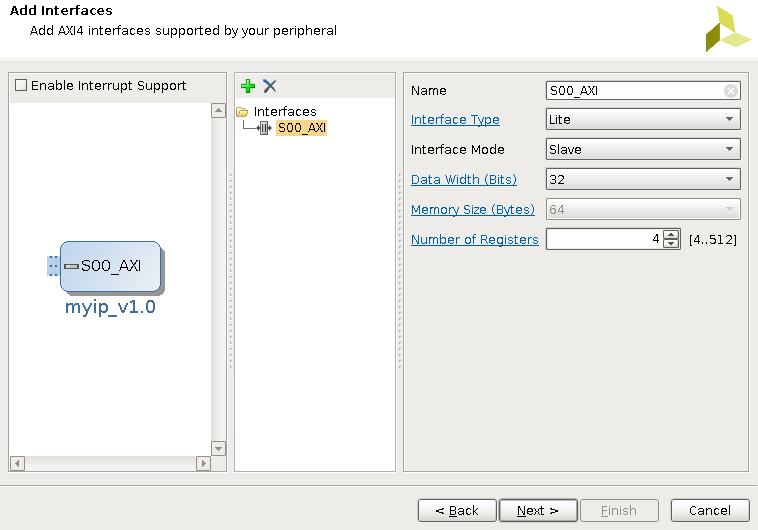
*Figure 3. Output Voltage vs. Average PWM Input*

1.1 Open Vivado 2016.1

1.2 Select Tools -> Create and Package IP

1.3 Create a new AXI4 Peripheral and give it a name such as “Audio\_PWM”

1.4 Click Next and change the Interface Type to **Stream**.



1.5 Click Next and select “Edit IP”. This will open a new Vivado project for the IP. Under sources you will see both the Verilog sources that describes your IP. There are two sources, a top level “Audio\_PWM\_v1\_0v” and an interface module “Audio\_PWM\_v1\_0\_S00\_AXIS”.

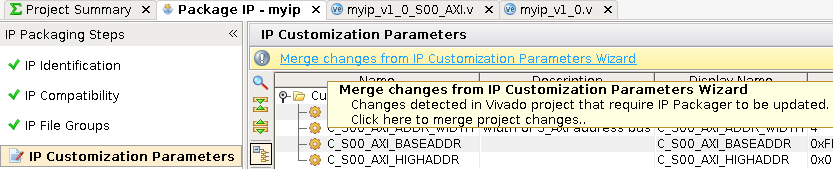
Modifying the AXI Stream IP

1.6 The skeleton AXI Stream core contains a series of skeleton codes. However, since we have provided you with the source code, you can delete all codes within Audio\_PWM\_v1\_0\_S00\_AXIS. Copy and paste the provided code into Audio\_PWM\_v1\_0\_S00\_AXIS. Make sure the module name match with your project name.

1.7 Modify the top level module Audio\_PWM\_v1\_0v to define PWM output and AUD\_SD as output ports. Copy and paste the source code into Audio\_PWM\_v1\_0v. Once again, make sure the top level module and the instance name match with your current project name.

Package the IP

1.8 Select the Package IP tab. To automatically merge the changes, select the IP Packaging Step with an edited icon and click the Merge changes from IP Customization Parameter Wizard.



1.9 Examine the rest of the Packaging Steps then on the Review and Package step click Re-Package IP.

2. Setting up the Hardware

2.1. Invoke the Vivado IDE

2.2. Bring up the Tcl Console at the window and enter

**Set\_param board.repoPaths <path-to-board-fles>/board\_files/**

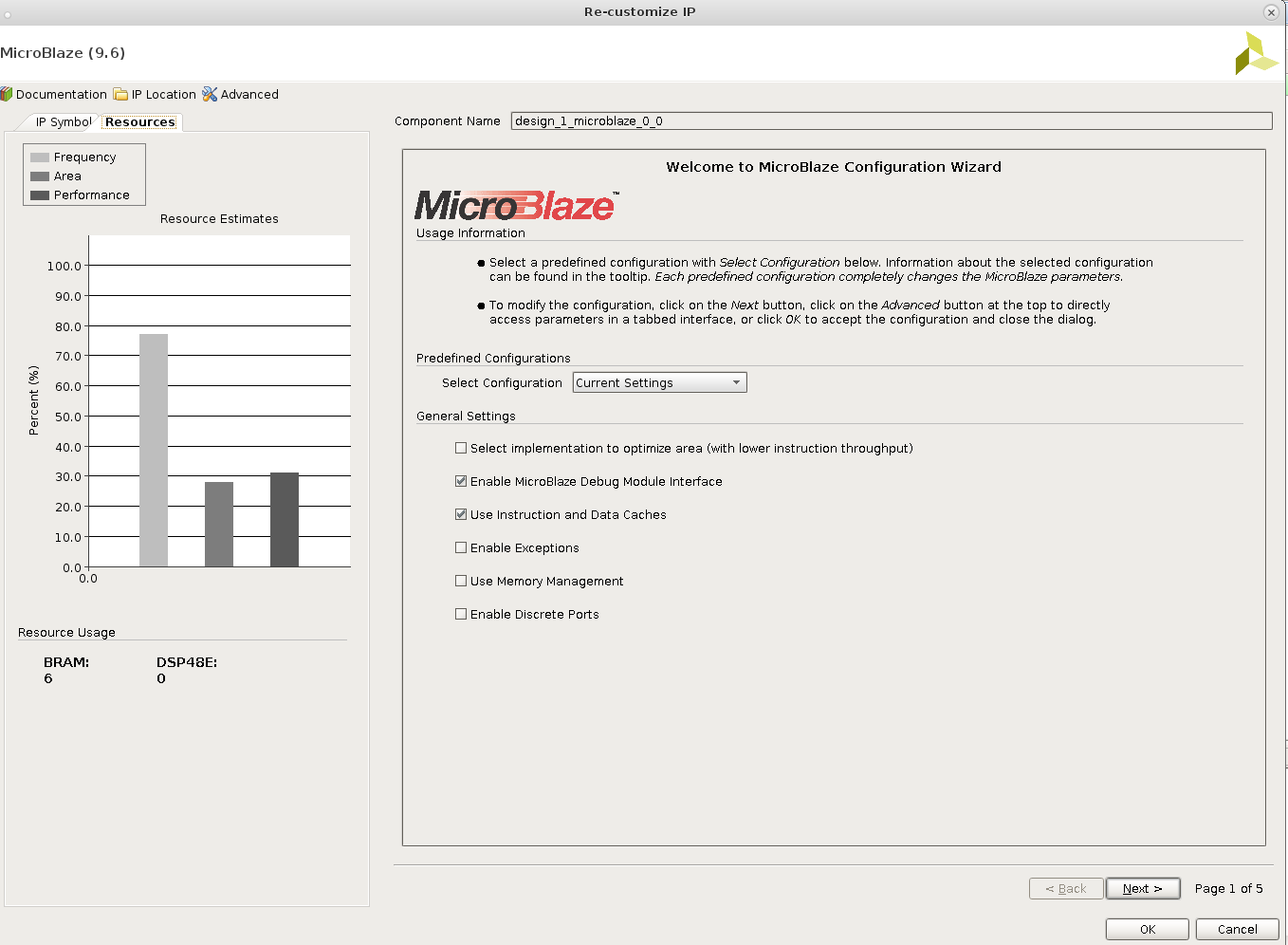
2.3. Create a **New Project** specifying the **Nexys 4 DDR** board and leaving the rest of the setting as default

2.4. From Navigator > IP Integrator, select **Create Block Diagram**

2.5. Right click anywhere in the Diagram and select Add IP to and add a MicroBlaze block to the design

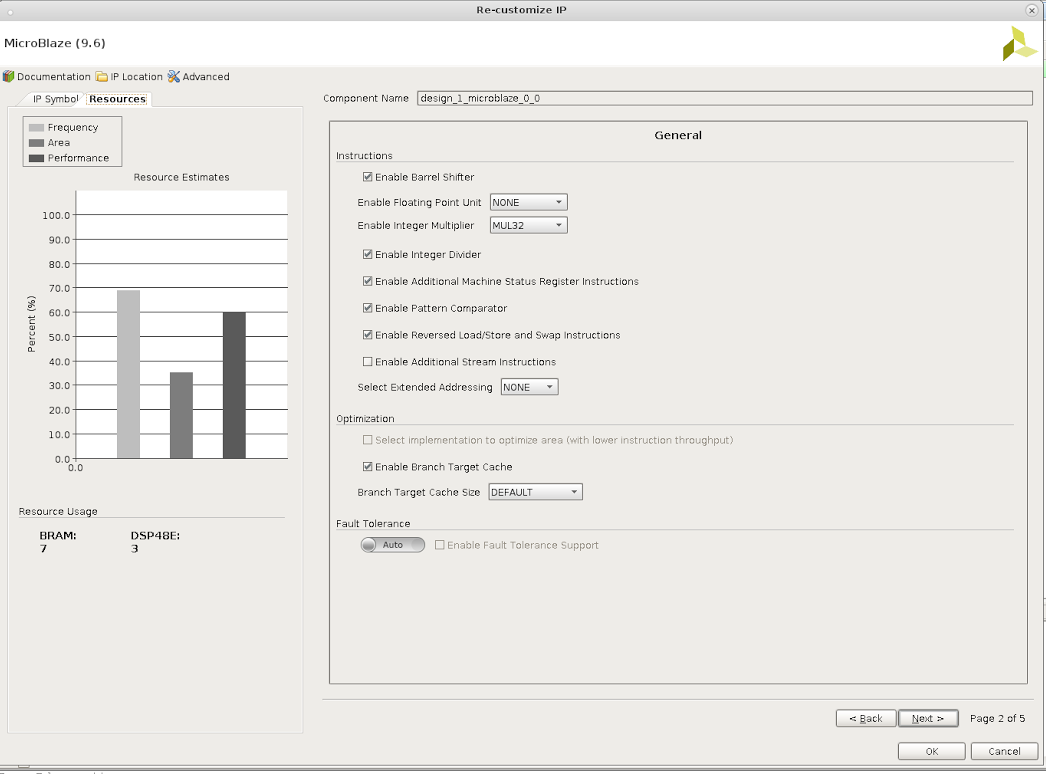
2.6. Double-click the Microblaze Processor diagram to open the Re-customize IP dialog box

2.7. On page 1, check the use Instruction and Data Caches Option

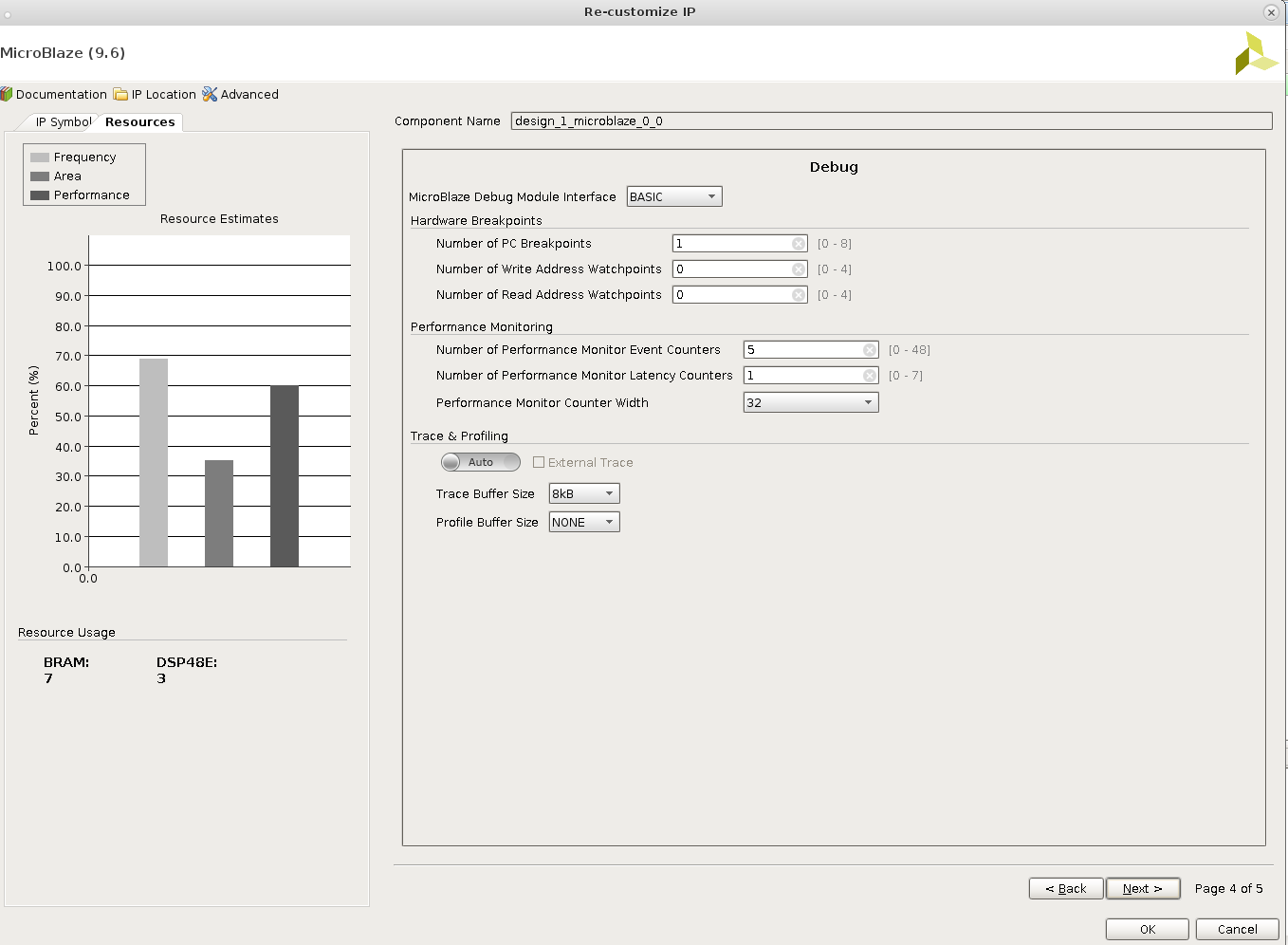


2.8. On page 2 of the Re-customize IP dialog box:

* Check the **Enable Barrel Shifter** option.
* From the pulldown menu, in option **Enable Integer Multiplier** select **MUL32** (32-bit)
* Check the **Enable Integer Divide** option.
* Check the **Enable Addition Machine Status Register Instructions** option
* Check the **Enable Pattern Comparator** option
* Check the **Enable Revered Load/Store and Swap Instructions** option
* Check the **Enable Branch Target Cache** option
* Click **Next**

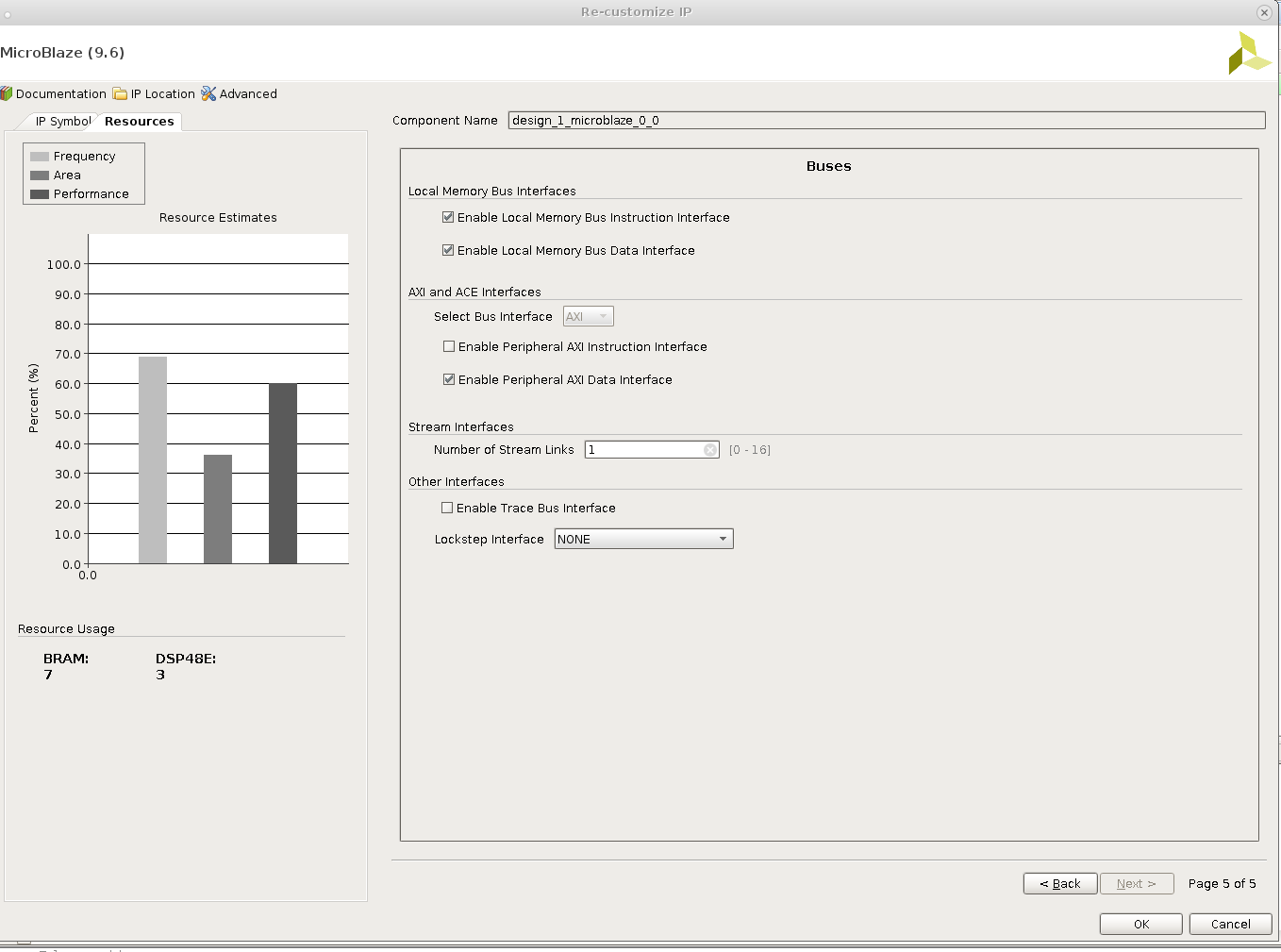


2.9. On page 4 of the Re-customize IP dialog box, ensure that the **MicroBlaze Debug Module** is enabled (i.e. BASIC), and click Next.

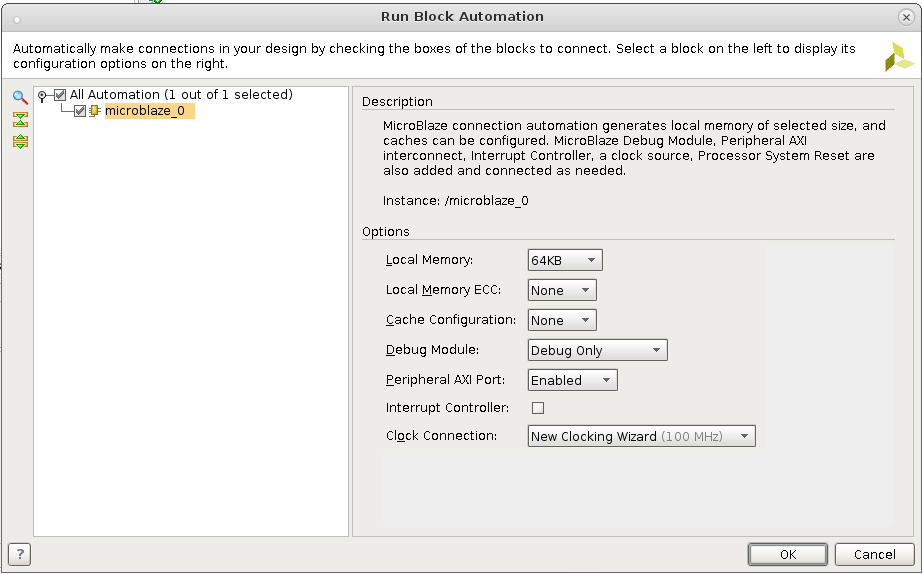


2.10. On page 5 of the Re-customize IP dialog box:

* Check the **Enable Peripheral AXI Data Interface** option
* Enter **1** for **Number of Stream Links**
* Click OK to re-configure the MicroBlaze processor



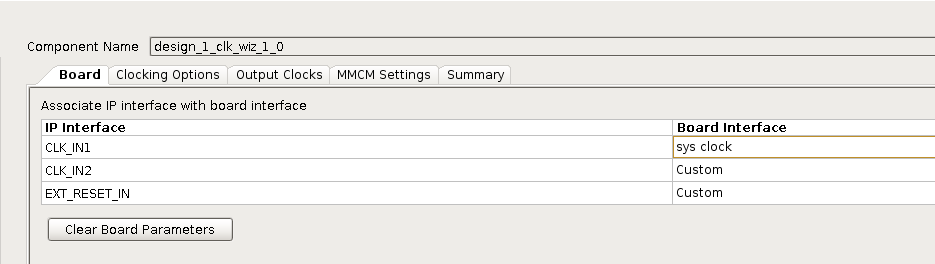
2.11. **Run Block Automation** for the MicroBlaze with **Local Memory** set to 64KB



Clock Customization

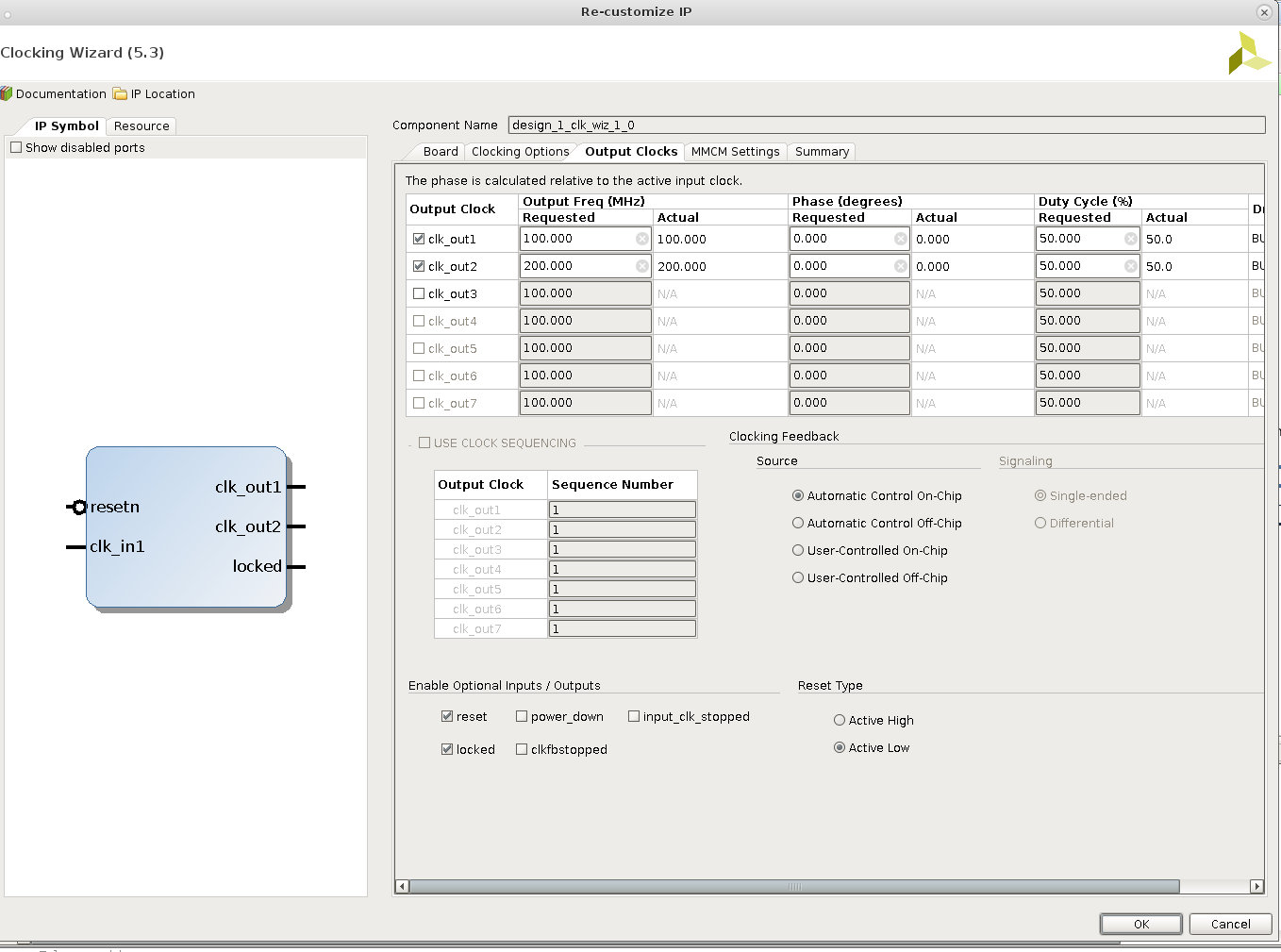
2.12. Double click the **Clock Wizard** (clk\_wiz\_1) block to re-customize it.

2.13 Under the **Board** tab, use the Board Interface pull-down menu for IP interface CLK\_IN1 and select sys clock.



2.14 Under the **Output Clocks** tab, check the radio box for Output Clock clk\_out2 and enter a requested clock frequency of 200 MHz.

2.15 While in Output Clocks tab, change the Reset Type to Active Low.



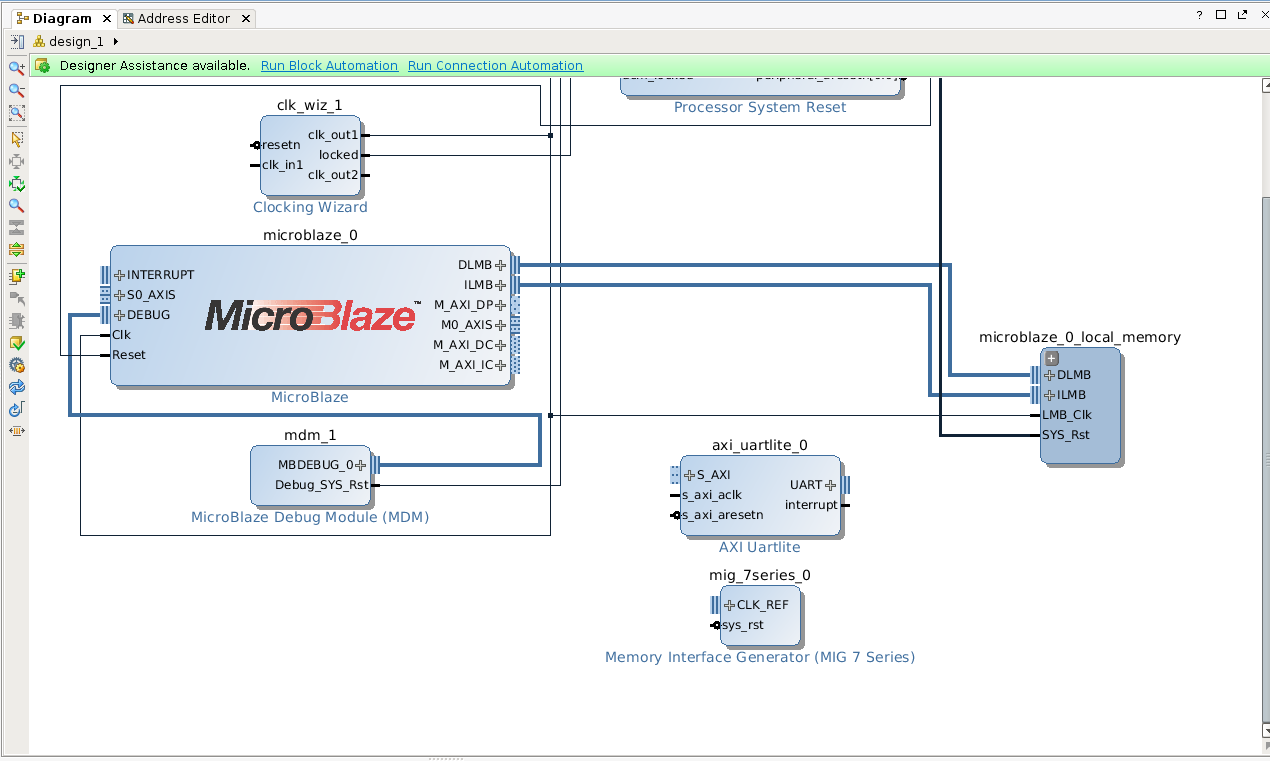
DDR2 Memory Block

2.16. Right click anywhere in the Diagram and select Add IP and add an AXI Uartlite block to the design.

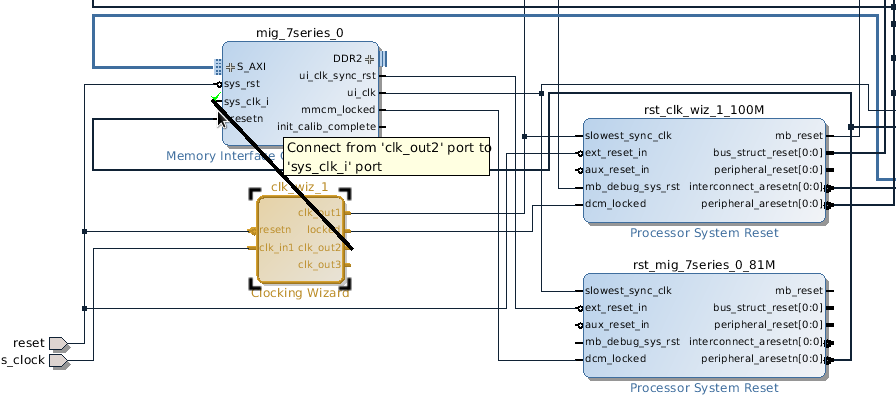
2.17 Repeat the process to search and add a Memory Interface Generator (MIG 7) peripheral.

2.18 Run Block Automation for the Memory Interface Generator. There may be an error during the process, you may ignore that and move on.

2.19 Your design should now look something like this. Run Connection Automation and select all connections.



2.20 Manually connect clk\_out2 from clk\_wiz\_1 to the Memory Interface Generator sys\_clk\_i



2.21 Right click **DDR2** in the Memory Interface Generator and make it **external**.

Connect PWM module to the design

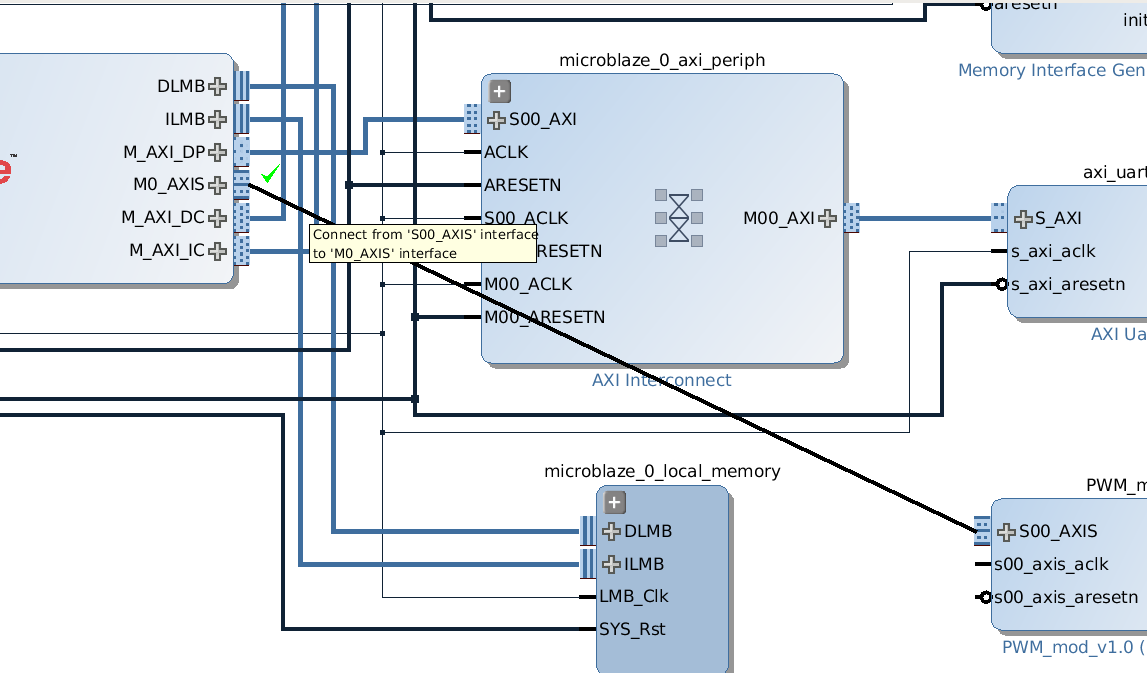
2.22 Choose from menu: **window > ip catalog**

2.23 Right click the blank space and choose **add repository**

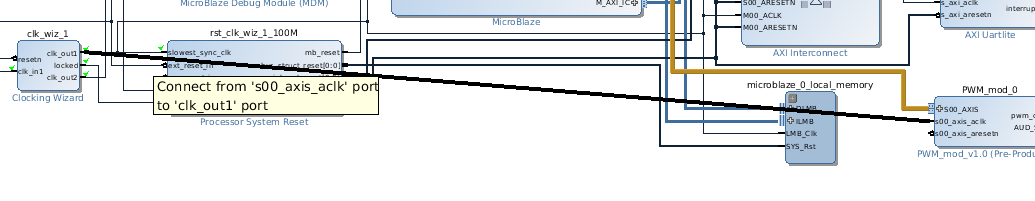
2.24 Choose the IP repository

2.25 Right click on block diagram and then **Add IP**. Choose the ip core you just added

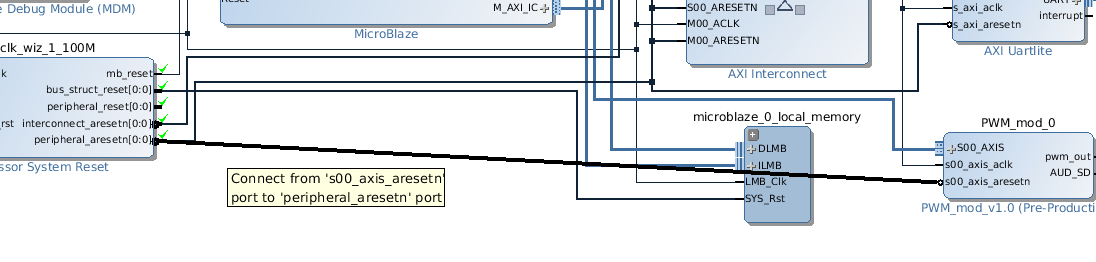
2.26 Connect S00\_AXIS to M0\_AXIS



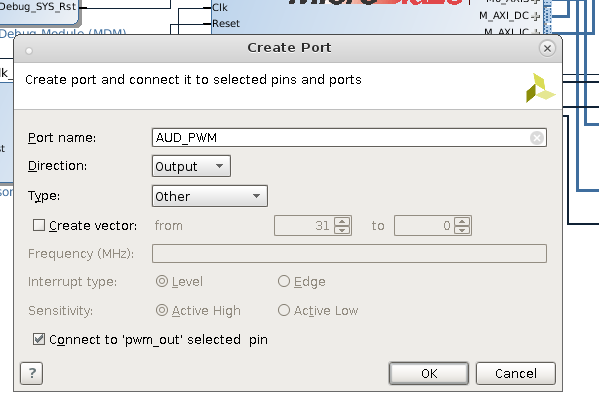
2.27 Connect clock to clk\_out1, which is 100MHz



2.28 Connect PWM reset signal to peripheral\_aresetn on Processor System Rest



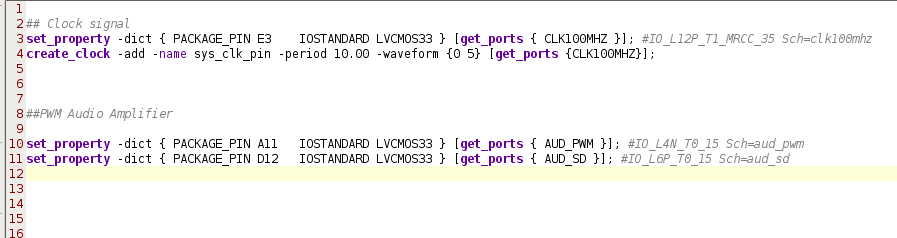
2.29 Create ports for pwm\_out amd aud\_sd on the PWM module, specifying the direction as output



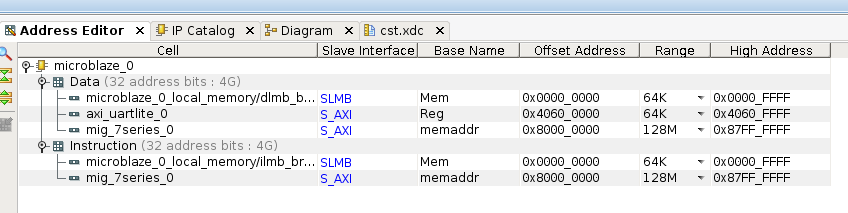
2.30 Validate the design

2.31 Create a new constraint file and copy provided content into file.

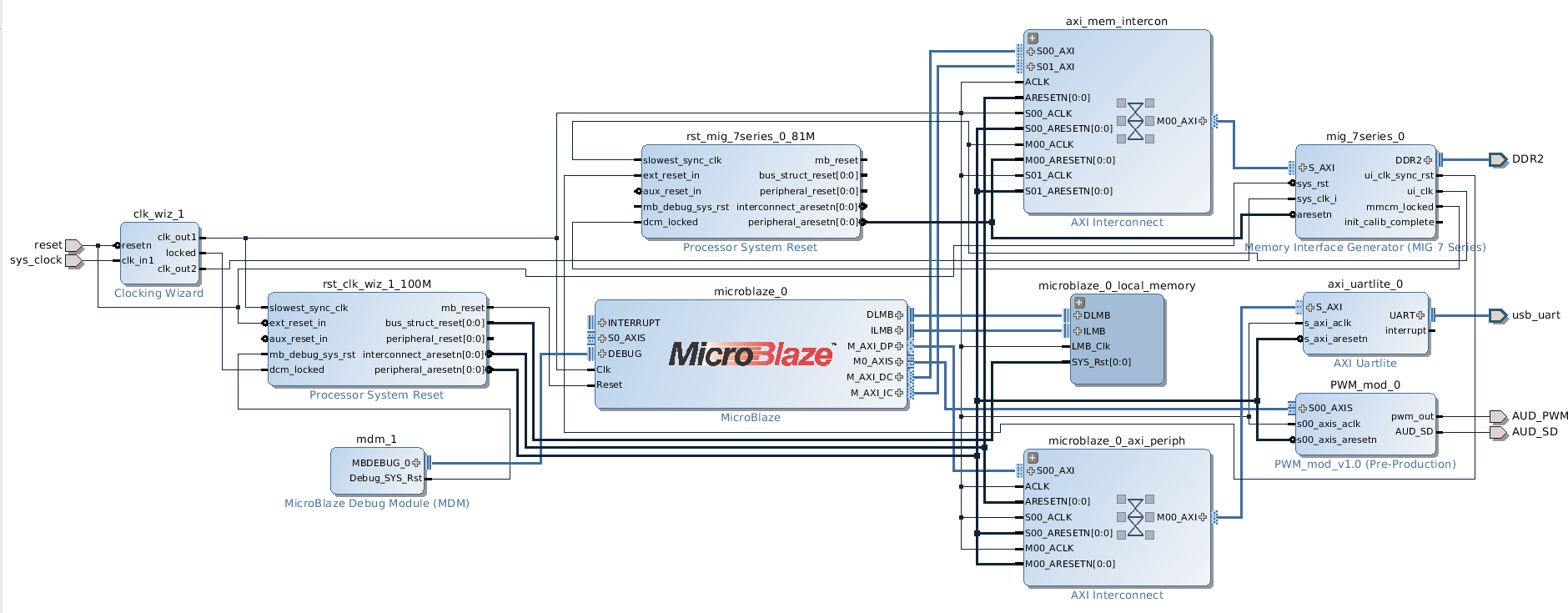
Notice that we connect AUD\_PWM to Port A11 and AUD\_SD to Port D12 on the board. Port A11 is the input port to a 4th order low pass filter on the board to transfer PWM signal to audio output. AUD\_SD controls the audio output. When AUD\_SD is 1, the system plays sound. We set the AUD\_SD signal in PWM IP configuration part (using Verilog).



2.32 Check address editor



The complete design looks like:



2.33 Generate HDL wrapper

2.24 Generate Bitstream

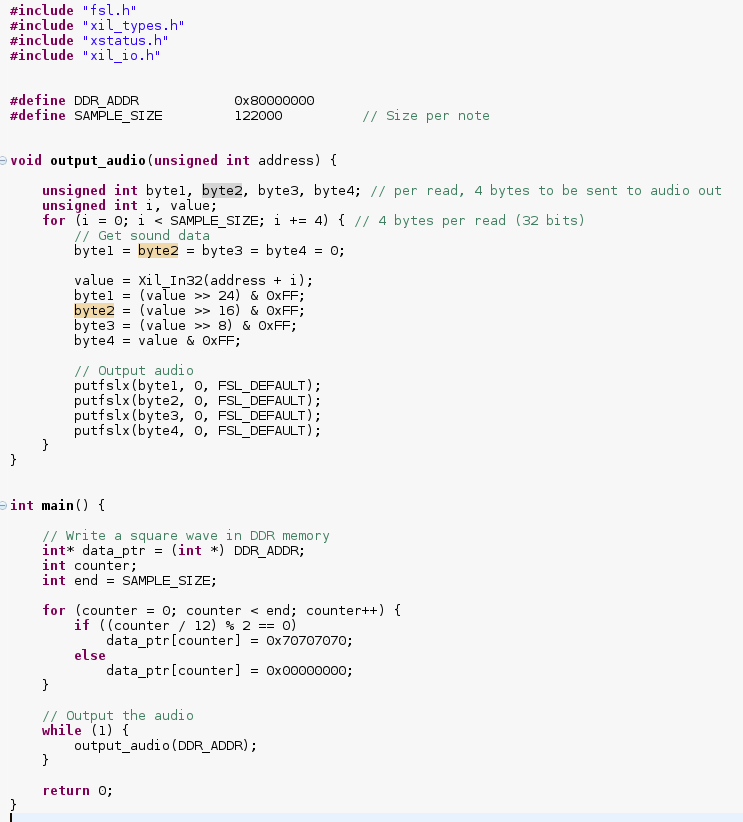
2.25 Export Hardware and Launch SDK

SDK Configuration

3.1. Open SDK and create a new project (Xilinx Application). Make it an empty application.

3.2. Right click the new project to create a new C source file, give it the name main.c.

3.3. Copy and paste the provided main.c into the project. Try to understand the code.



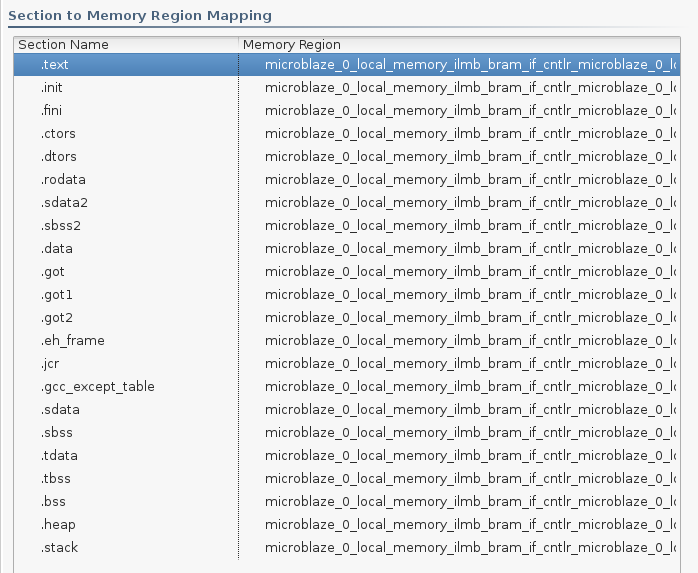
The output audio function reads in the memory address of the audio to be played, and writes the audio bytes into the PWM module by streams.

The main function firstly creates a square wave of frequency 508.625Hz, and writes the audio into the DDR. Then it constantly writes the audio into the PWM module for audio output.

*1 Period = 12 \* 4 (one byte per sample, integer is four byte) \* 2 = 96 samples*

*Frequency = 48828 (sample rate) / 96 samples/period = 508.625 Hz*

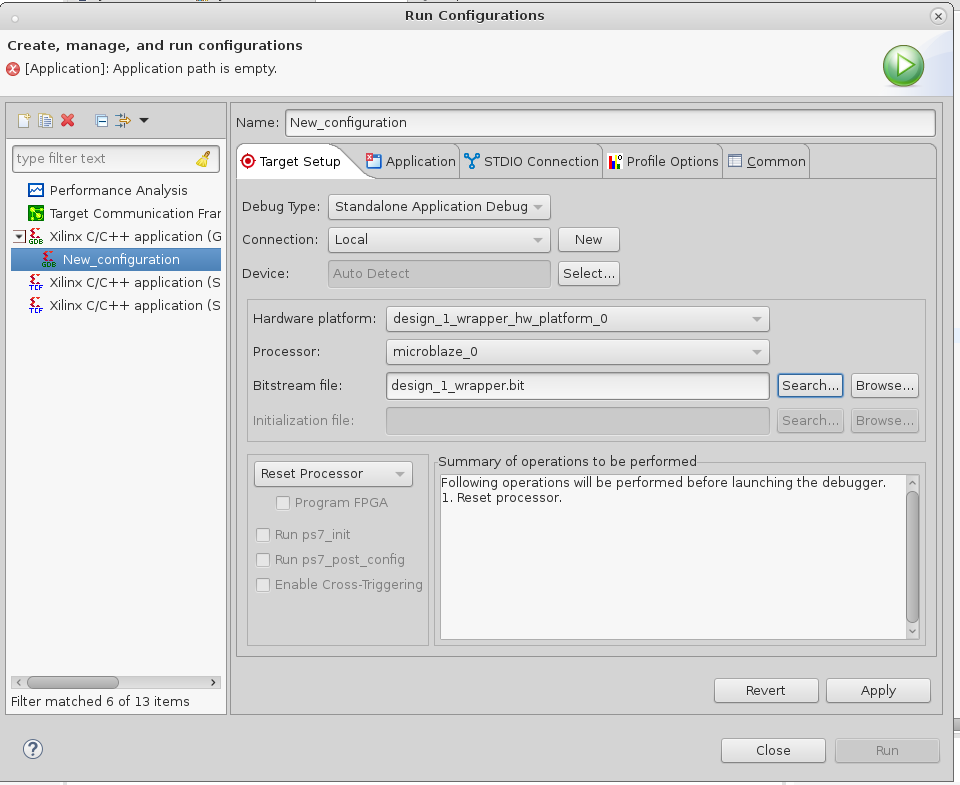
3.4. Open the lscript.ld (linker script) under src. Change all Memory Region Mapping to local memory as shown.

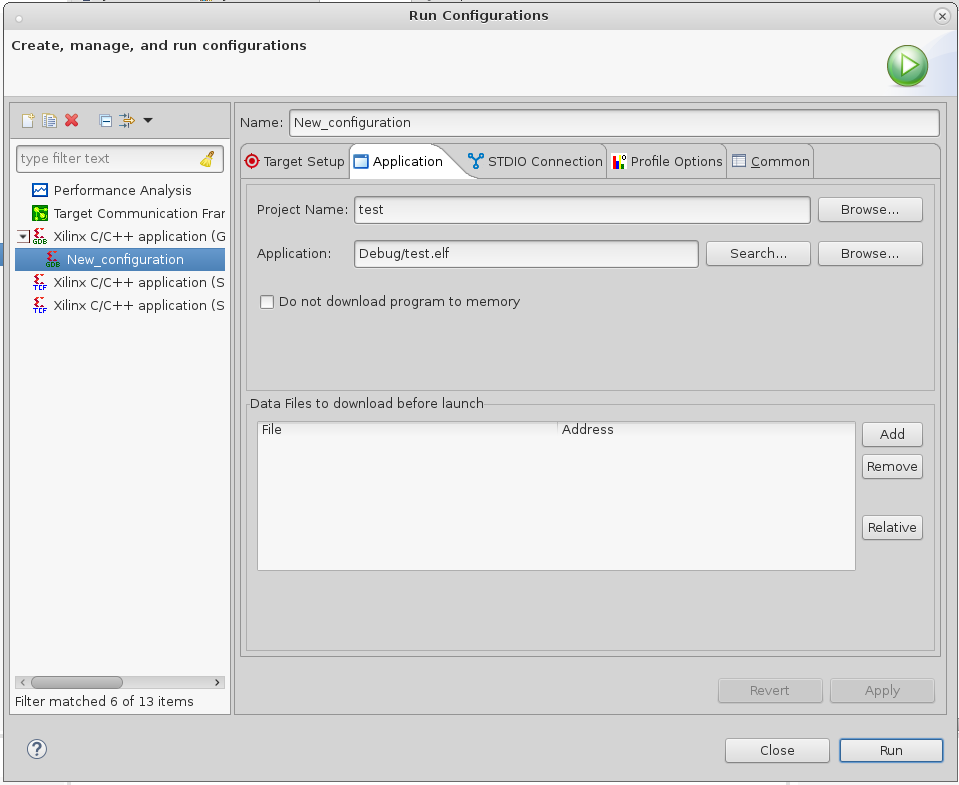


3.5. Save and build the project.

3.6. Program FPGA.

3.7. Run configuration. Create a new Xilinx C/C++ application (GDB) configuration. On the Target Setup page, choose the wrapper as Bitstream file. On the Application page, select this project.





3.8. Click Run.

**Exercise:**

C:\Users\Jiayuan\Desktop\3fd10020gw1eyfwur1dghj208y064a9z.jpg With the given project and XMD, load a song in PCM format into the DDR at 0x80000000 and play it through the mono audio jack.