

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:

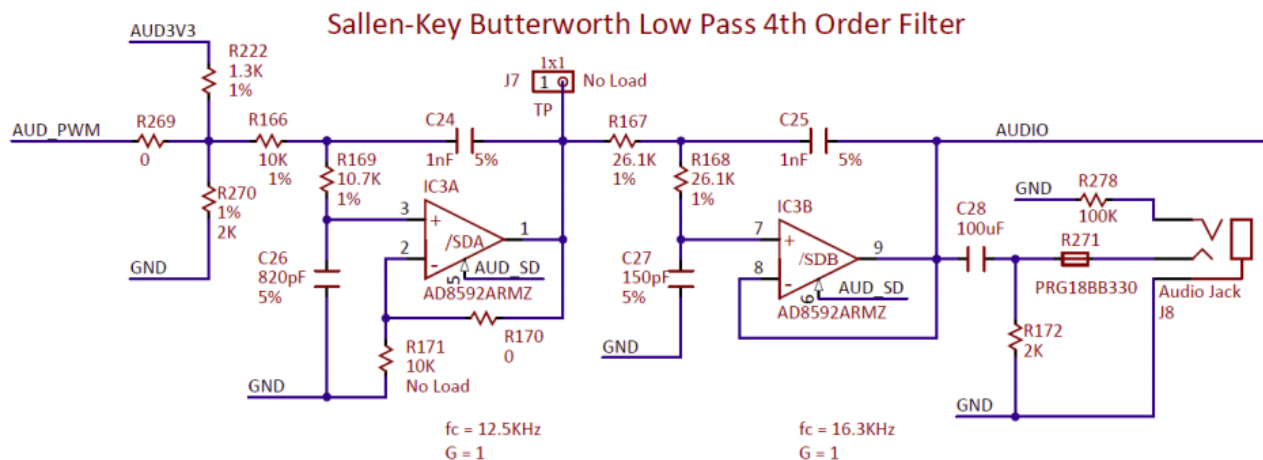


Figure1. Low Pass Filter on Nexys4 DDR Board

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.

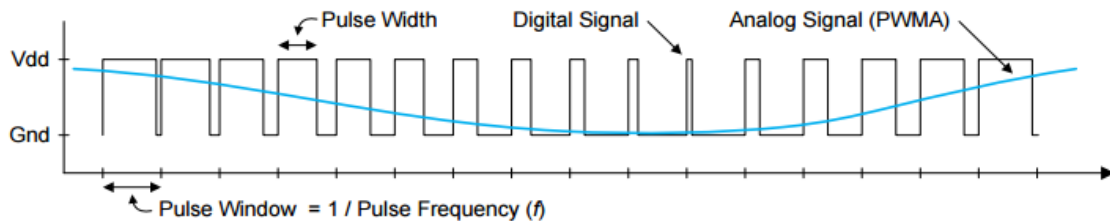


Figure 2. Waveform Represented as a PWM Signal

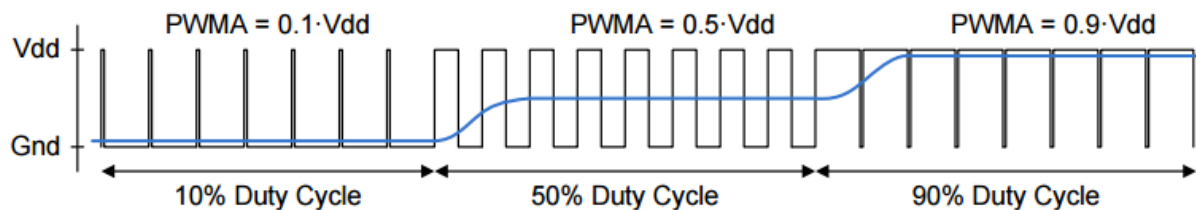


Figure 3. Output Voltage vs. Average PWM Input

1.1 Open Vivado 2016.1

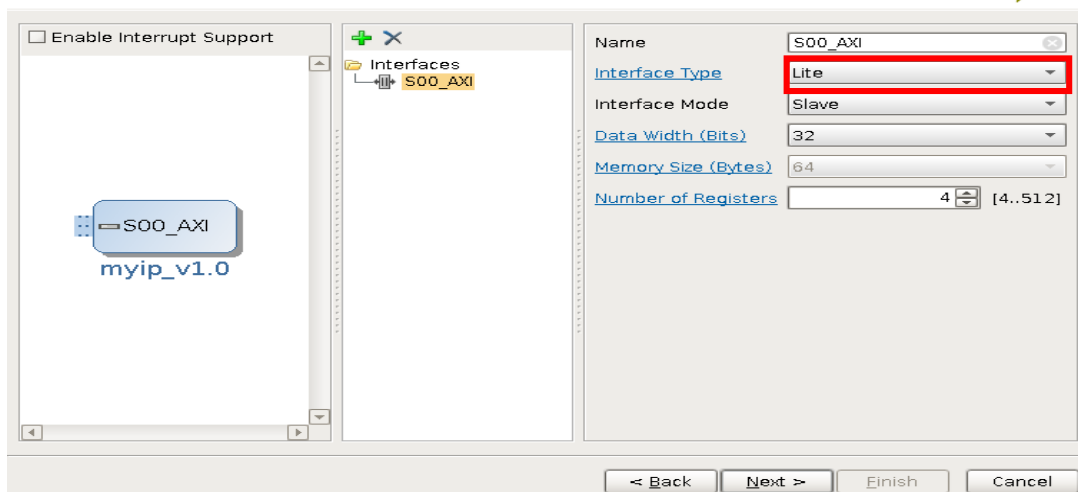
1.2 Select Tools -> Create and Package IP

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**.

Add Interfaces

Add AXI4 interfaces supported by your peripheral



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”.

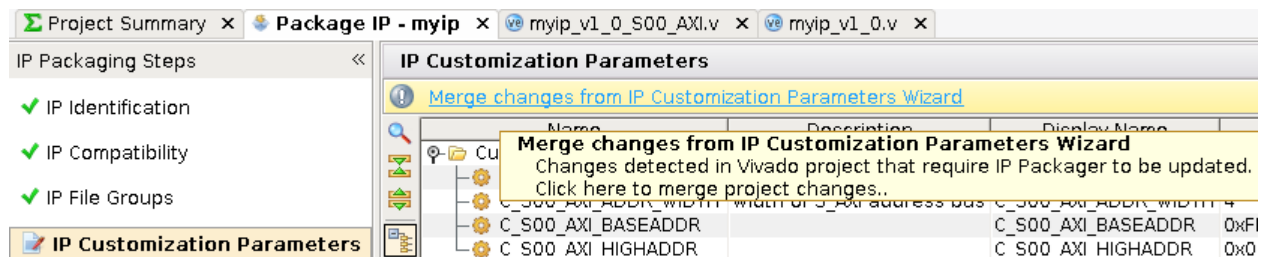
Modifying the AXI Stream IP

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-files>/board_files/

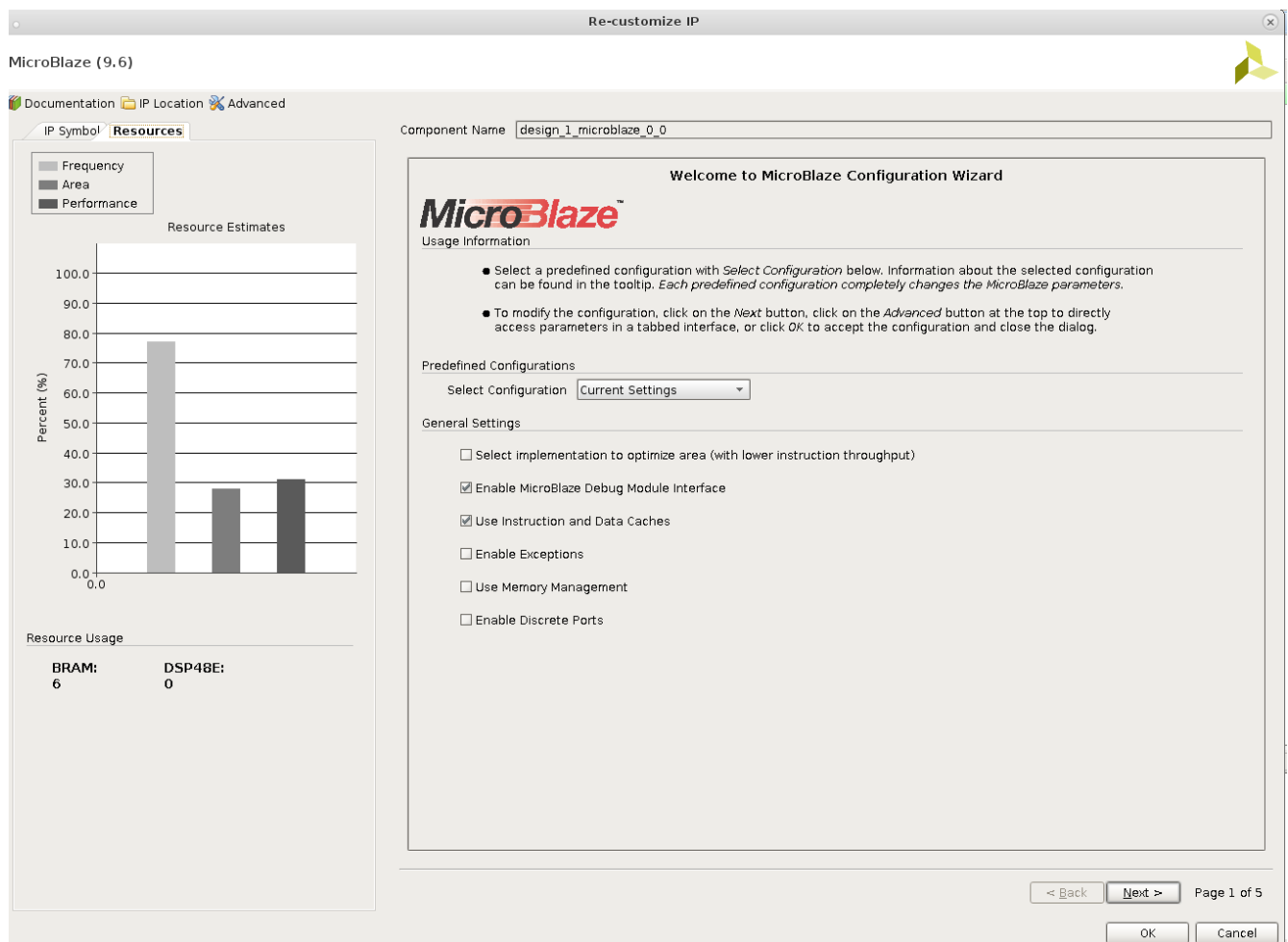
2.3. Create a **New Project** specifying the **Nexys 4 DDR** board and leaving the rest of the setting as default

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 Reversed Load/Store and Swap Instructions** option
- Check the **Enable Branch Target Cache** option
- Click **Next**

The screenshot shows the 'Re-customize IP' dialog box for MicroBlaze (9.6). The 'Resources' tab is active, displaying a bar chart titled 'Resource Estimates' with three bars: Frequency (approx. 70%), Area (approx. 35%), and Performance (approx. 60%). Below the chart, 'Resource Usage' is shown: BRAM: 7 and DSP48E: 3. The 'Component Name' is 'design_1_microblaze_0_0'. The 'General' tab is selected, showing the 'Instructions' section with several options checked: 'Enable Barrel Shifter', 'Enable Integer Divider', 'Enable Addition Machine Status Register Instructions', 'Enable Pattern Comparator', and 'Enable Reversed Load/Store and Swap Instructions'. The 'Enable Floating Point Unit' is set to 'NONE' and 'Enable Integer Multiplier' is set to 'MUL32'. The 'Optimization' section has 'Select implementation to optimize area (with lower instruction throughput)' unchecked and 'Enable Branch Target Cache' checked. The 'Branch Target Cache Size' is set to 'DEFAULT'. The 'Fault Tolerance' section has 'Auto' selected and 'Enable Fault Tolerance Support' unchecked. Navigation buttons at the bottom include '< Back', 'Next >', 'OK', and 'Cancel'. The page number 'Page 2 of 5' is displayed.

Category	Percent (%)
Frequency	~70.0
Area	~35.0
Performance	~60.0

Resource	Usage
BRAM	7
DSP48E	3

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.

Re-customize IP

MicroBlaze (9.6)

Documentation IP Location Advanced

IP Symbol Resources

Resource Estimates

Category	Percent (%)
Frequency	70.0
Area	35.0
Performance	60.0

Resource Usage

Resource	Usage
BRAM:	7
DSP48E:	3

Component Name design_1_microblaze_0_0

Debug

MicroBlaze Debug Module Interface BASIC

Hardware Breakpoints

Number of PC Breakpoints 1 [0 - 8]

Number of Write Address Watchpoints 0 [0 - 4]

Number of Read Address Watchpoints 0 [0 - 4]

Performance Monitoring

Number of Performance Monitor Event Counters 5 [0 - 48]

Number of Performance Monitor Latency Counters 1 [0 - 7]

Performance Monitor Counter Width 32

Trace & Profiling

Auto External Trace

Trace Buffer Size 8kB

Profile Buffer Size NONE

< Back Next > Page 4 of 5

OK Cancel

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

Re-customize IP

MicroBlaze (9.6)

Documentation IP Location Advanced

IP Symbol Resources

Component Name

Buses

Local Memory Bus Interfaces

- ☒ Enable Local Memory Bus Instruction Interface
- ☒ Enable Local Memory Bus Data Interface

AXI and ACE Interfaces

Select Bus Interface

- ☐ Enable Peripheral AXI Instruction Interface
- ☒ Enable Peripheral AXI Data Interface

Stream Interfaces

Number of Stream Links [0 - 16]

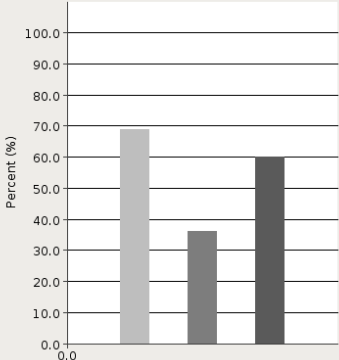
Other Interfaces

- ☐ Enable Trace Bus Interface

Lockstep Interface

Resource Estimates

Frequency Area Performance



Resource	Percent (%)
Frequency	70.0
Area	35.0
Performance	60.0

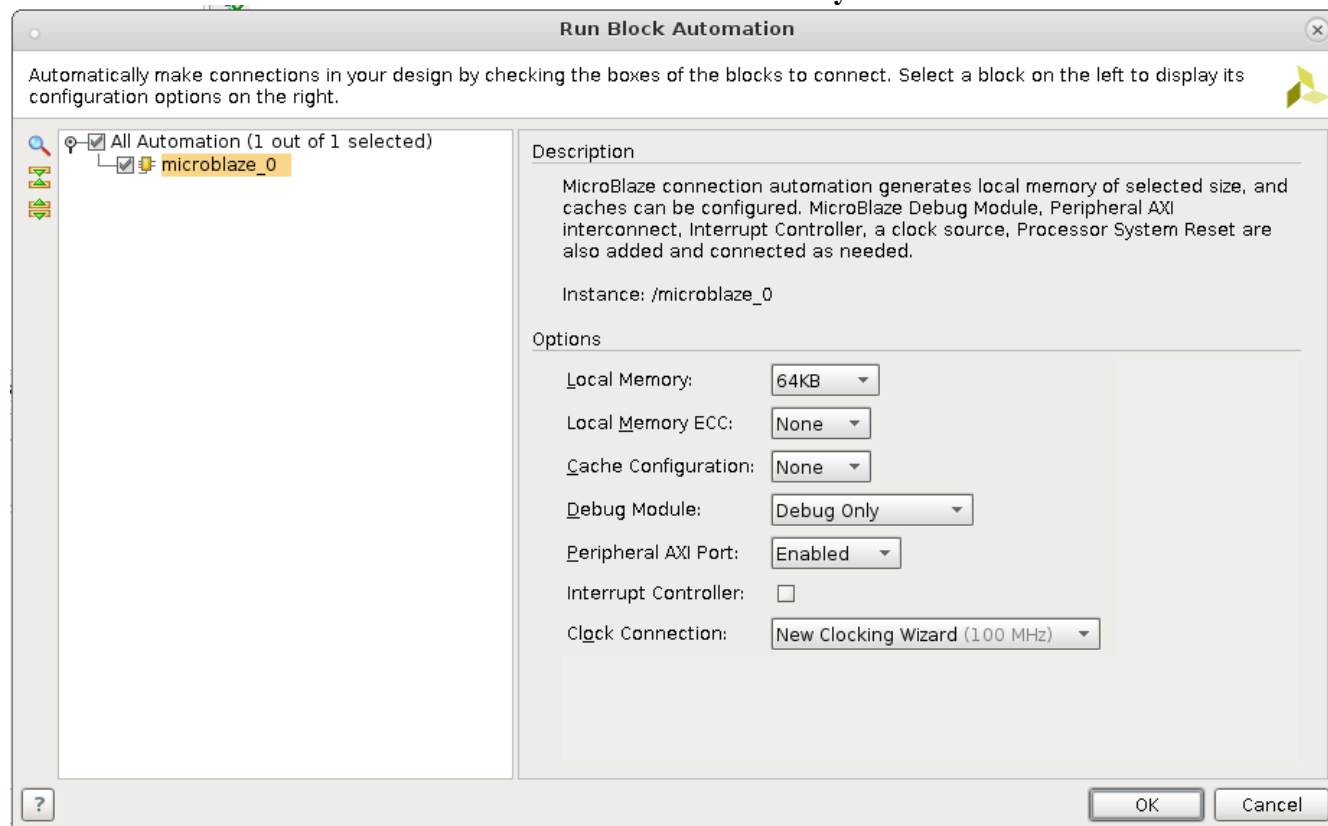
Resource Usage

Resource	Usage
BRAM:	7
DSP48E:	3

< Back Next > Page 5 of 5

OK Cancel

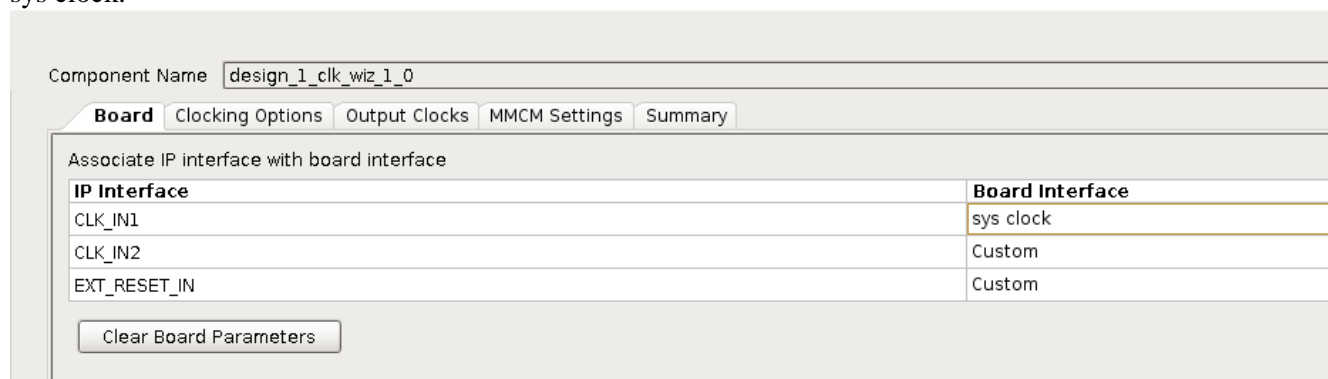
2.11. Run Block Automation for the MicroBlaze with Local Memory set to 64KB



Clock Customization

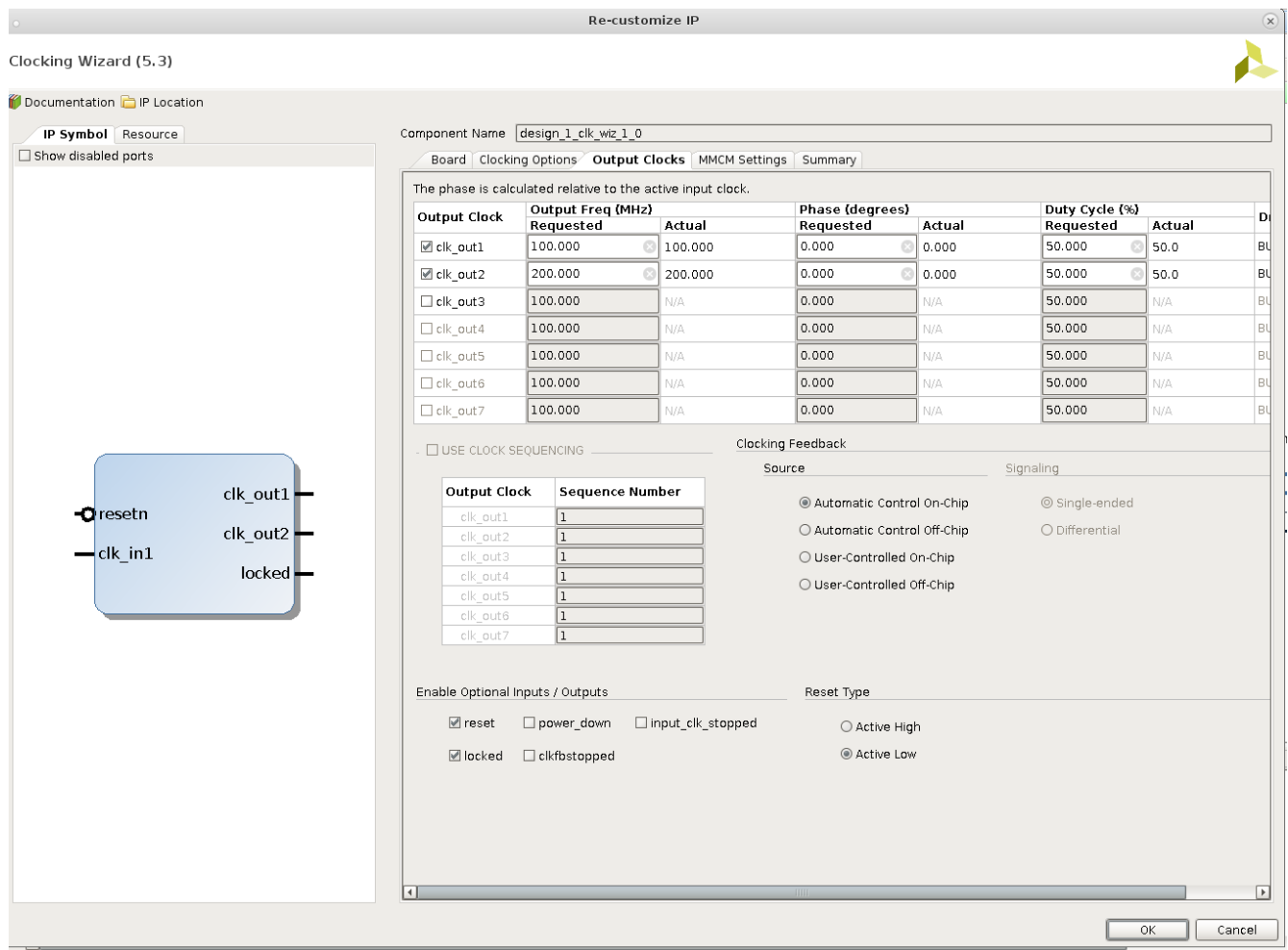
2.12. Double click the **Clock Wizard** (clk_wiz_1) block to re-customize it.

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.



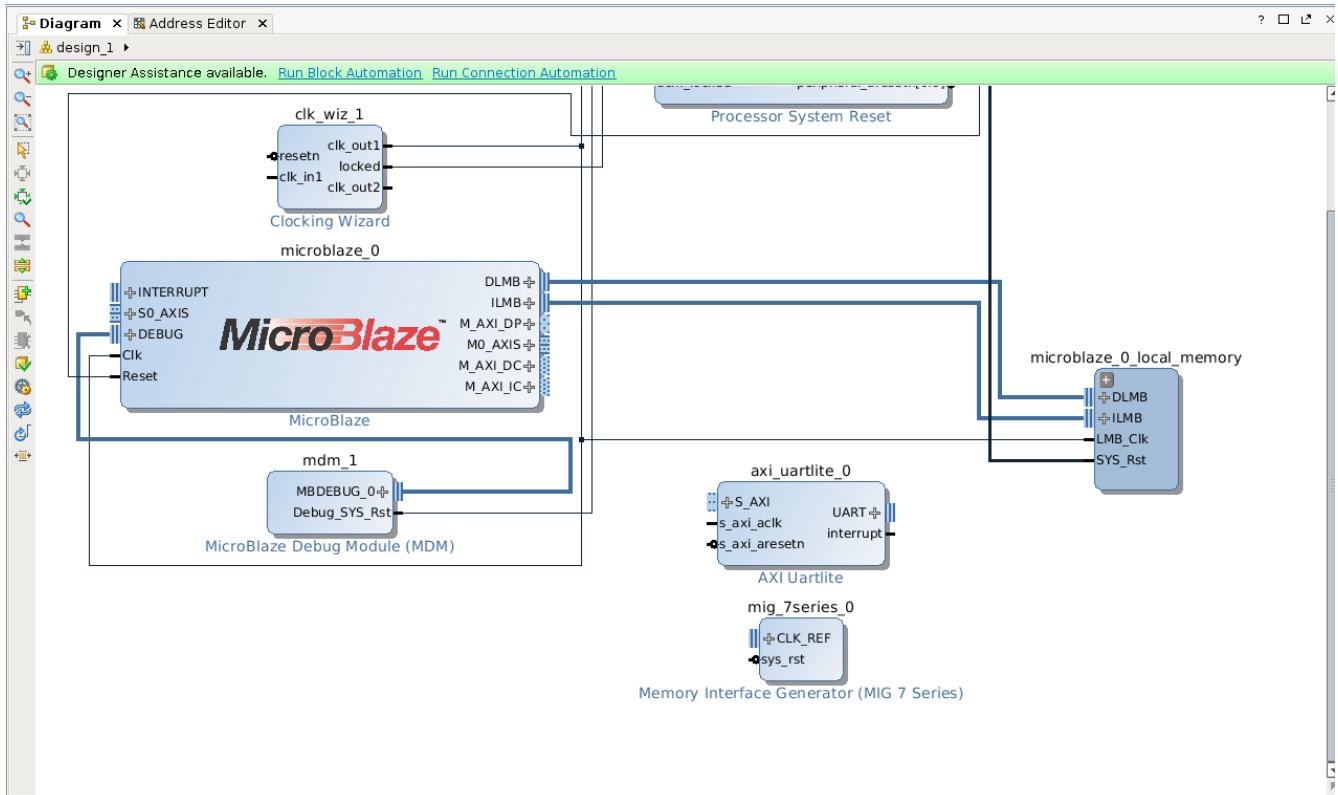
DDR2 Memory Block

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

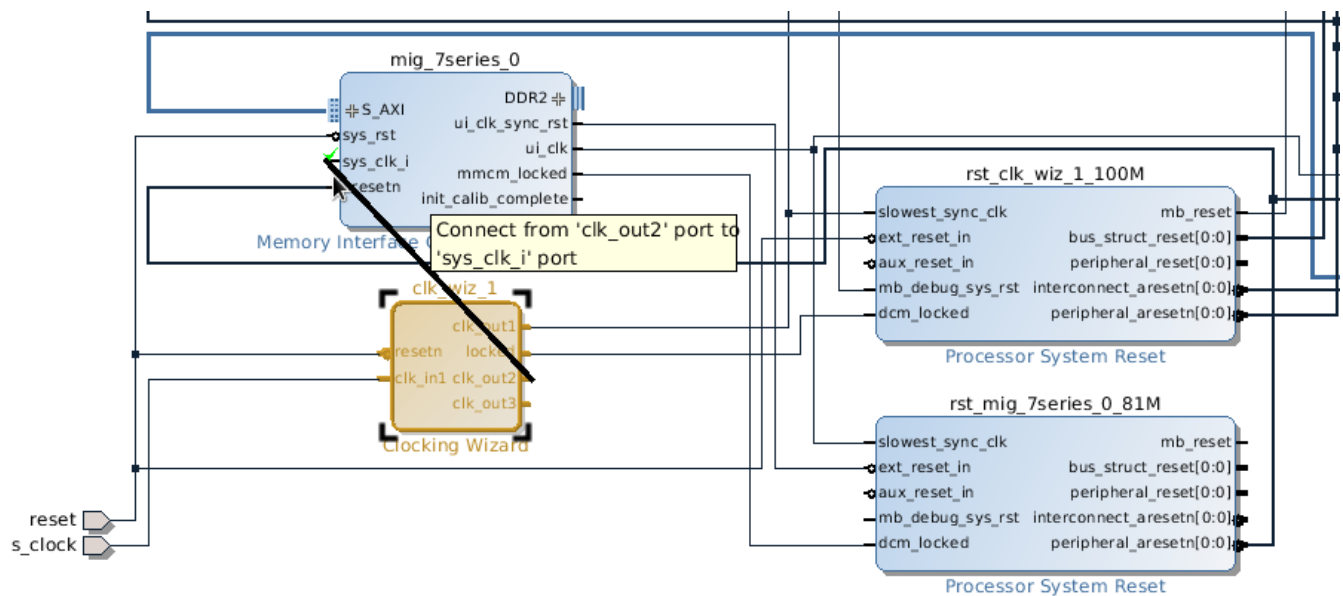
2.17 Repeat the process to search and add a Memory Interface Generator (MIG 7) peripheral.

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

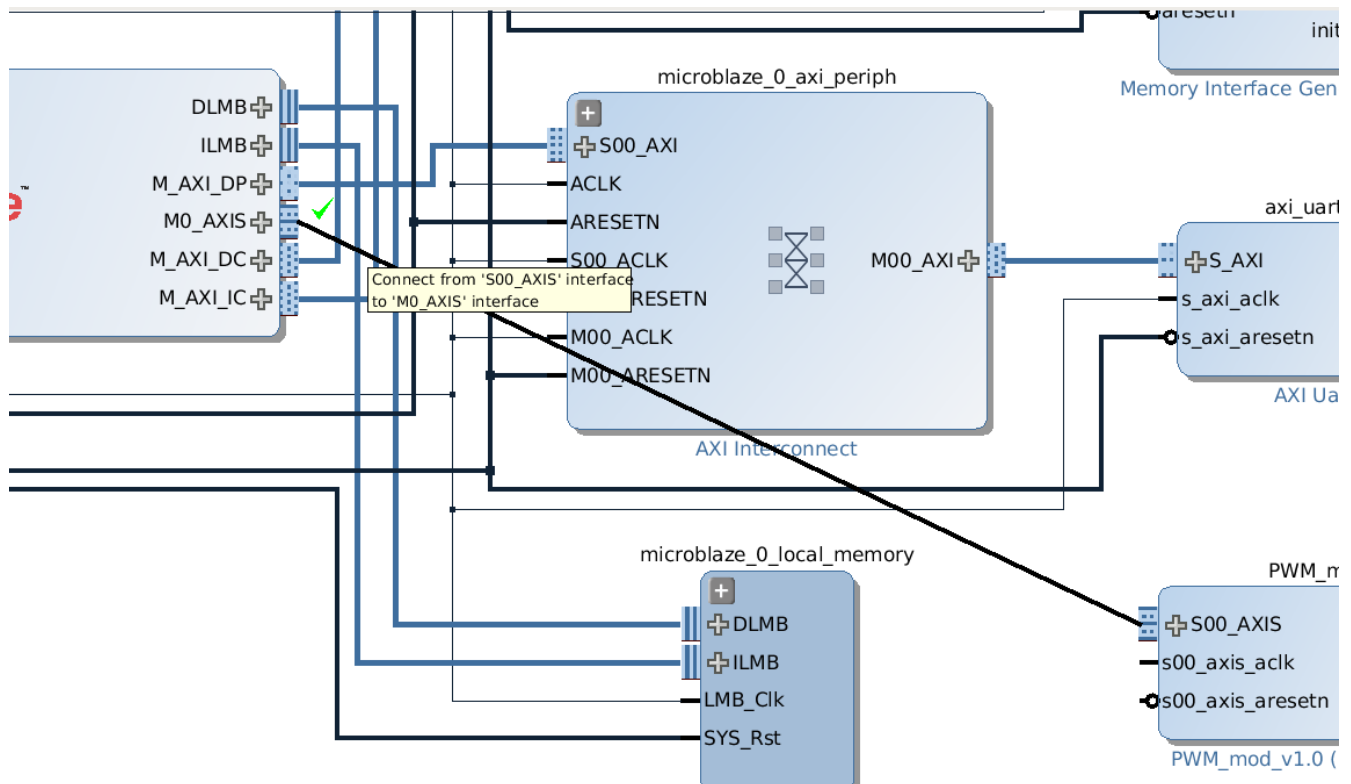
2.22 Choose from menu: **window > ip catalog**

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

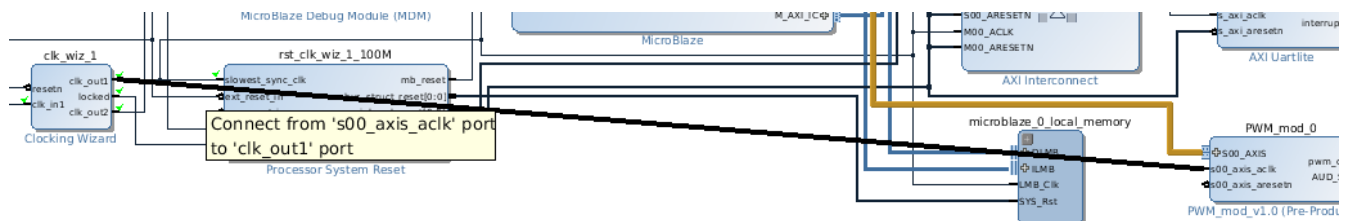
2.24 Choose the IP repository

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

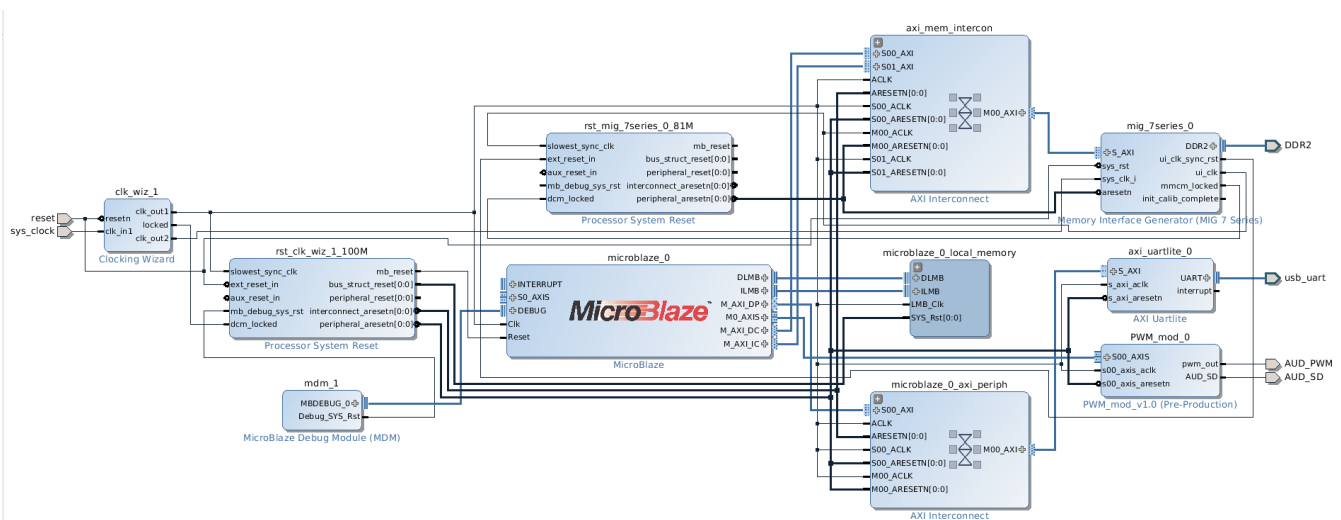


2.28 Connect PWM reset signal to peripheral_aresetn on Processor System Rest

2.32 Check address editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
microblaze_0_local_memory/dlmb_b...	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_br...	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF

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.

```
#include "fsl.h"
#include "xil_types.h"
#include "xstatus.h"
#include "xil_io.h"

#define DDR_ADDR          0x80000000
#define SAMPLE_SIZE      122000          // Size per note

void output_audio(unsigned int address) {
    unsigned int byte1, byte2, byte3, byte4; // per read, 4 bytes to be sent to audio out
    unsigned int i, value;
    for (i = 0; i < SAMPLE_SIZE; i += 4) { // 4 bytes per read (32 bits)
        // Get sound data
        byte1 = byte2 = byte3 = byte4 = 0;

        value = Xil_In32(address + i);
        byte1 = (value >> 24) & 0xFF;
        byte2 = (value >> 16) & 0xFF;
        byte3 = (value >> 8) & 0xFF;
        byte4 = value & 0xFF;

        // Output audio
        putfslx(byte1, 0, FSL_DEFAULT);
        putfslx(byte2, 0, FSL_DEFAULT);
        putfslx(byte3, 0, FSL_DEFAULT);
        putfslx(byte4, 0, FSL_DEFAULT);
    }
}

int main() {
    // Write a square wave in DDR memory
    int* data_ptr = (int *) DDR_ADDR;
    int counter;
    int end = SAMPLE_SIZE;

    for (counter = 0; counter < end; counter++) {
        if ((counter / 12) % 2 == 0)
            data_ptr[counter] = 0x70707070;
        else
            data_ptr[counter] = 0x00000000;
    }

    // Output the audio
    while (1) {
        output_audio(DDR_ADDR);
    }

    return 0;
}
```

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.

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

$$\text{Frequency} = 48828 \text{ (sample rate)} / 96 \text{ samples/period} = 508.625 \text{ Hz}$$

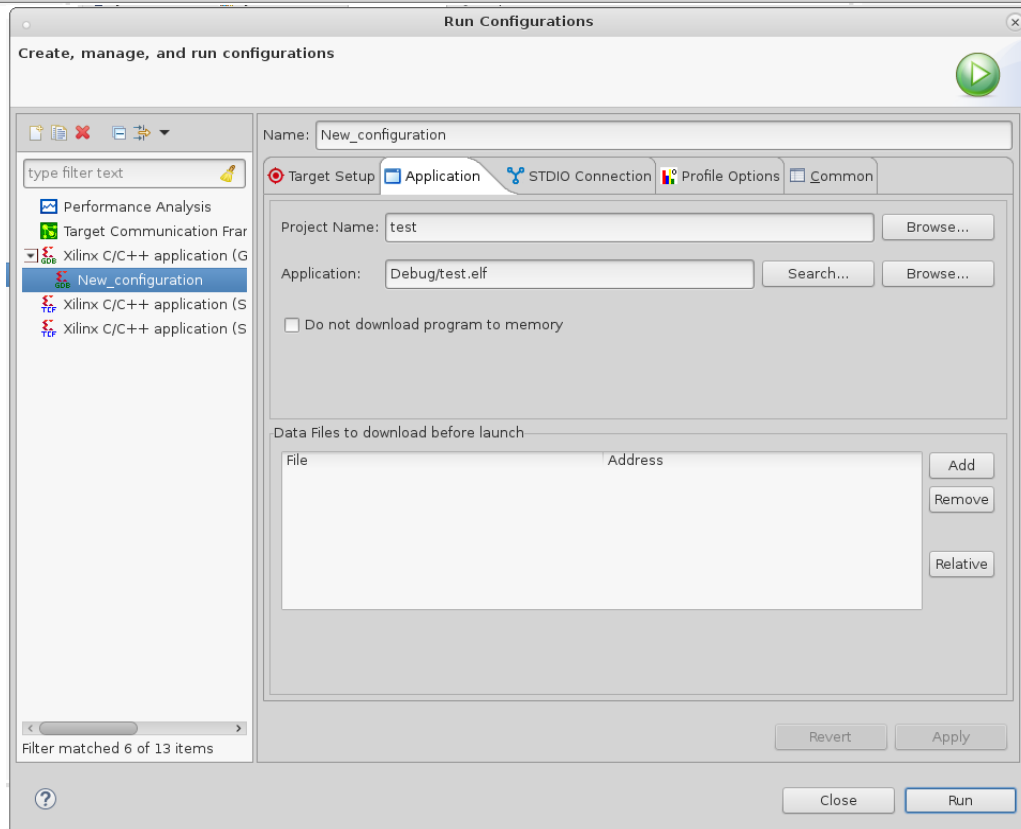
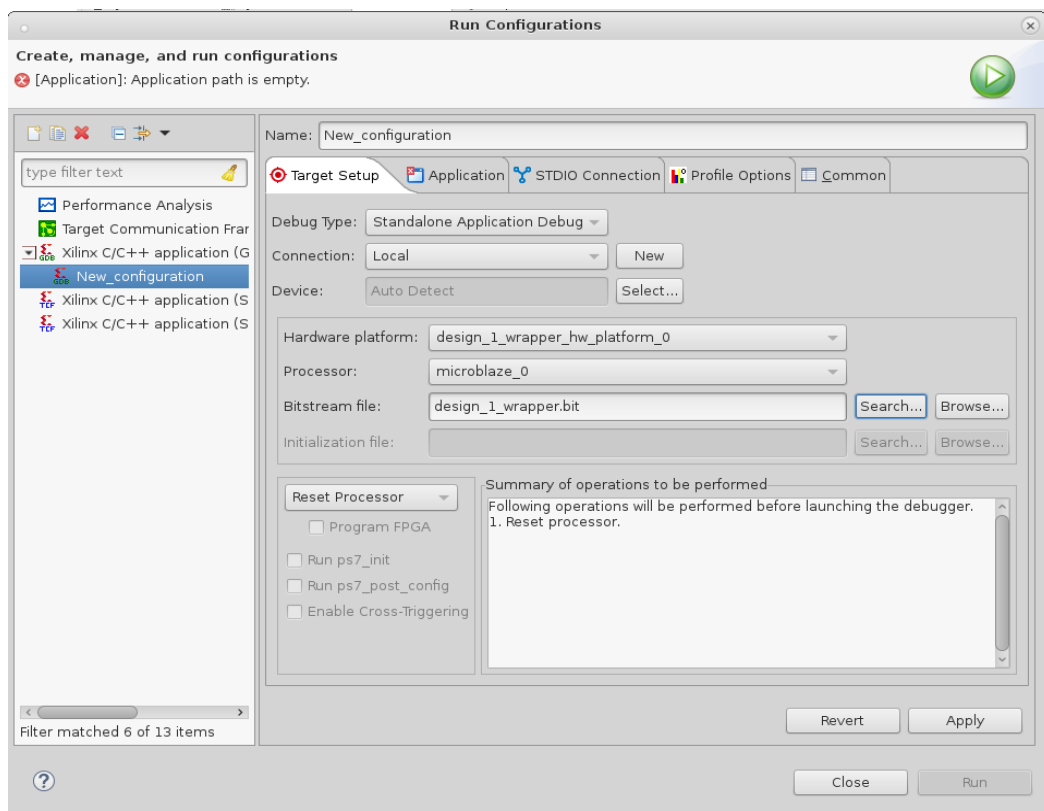
3.4. Open the lscript.ld (linker script) under src. Change all Memory Region Mapping to local memory as shown.

Section to Memory Region Mapping	
Section Name	Memory Region
.text	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.init	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.fini	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.ctors	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.dtors	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.rodata	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.sdata2	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.sbss2	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.data	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.got	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.got1	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.got2	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.eh_frame	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.jcr	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.gcc_except_table	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.sdata	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.sbss	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.tdata	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.tbss	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.bss	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.heap	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k
.stack	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_k

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:

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. 🐱