Using the Integrated Logic Analyzer (ILA)

Acknowledgement

This tutorial refers to Integrated Logic Analyzer v6.1 Product Guide from Xilinx.

Goal

- Use the ILA to monitor the internal signals of a design during runtime
- Set up a trigger event using the Vivado logic analyzer

Requirements

- Xilinx Vivado software
- Xilinx SDK software
- Xilinx Nexys 4 board and a programming cable
- Enough disk space for the project files

Background

The Vivado Integrated Logic Analyzer is a core that can be used to monitor the internal signals of your FPGA in real time through the Vivado logic analyzer software. The data is stored in the On-chip block RAM memory before it is uploaded by the software. After a trigger occurs, you can view the data captured in the waveform window of the Vivado Logic Analyzer.

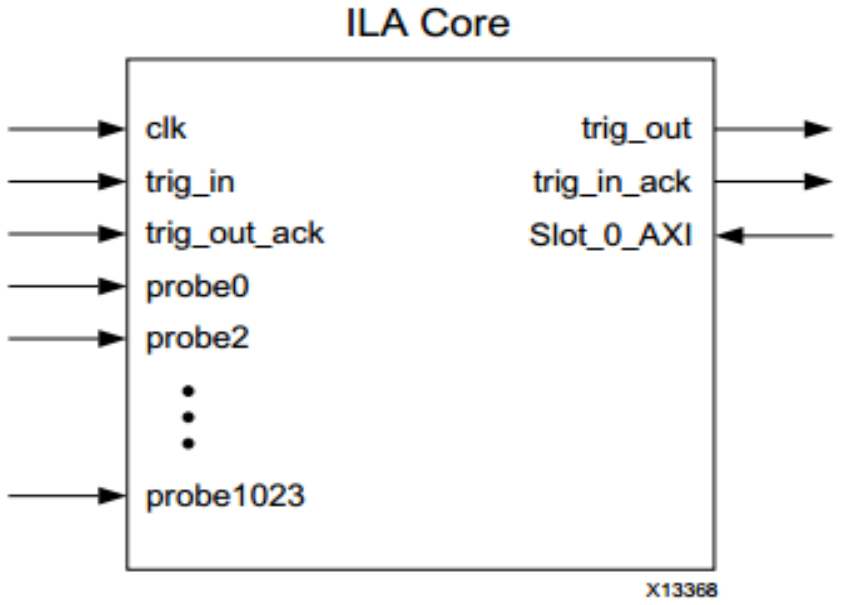
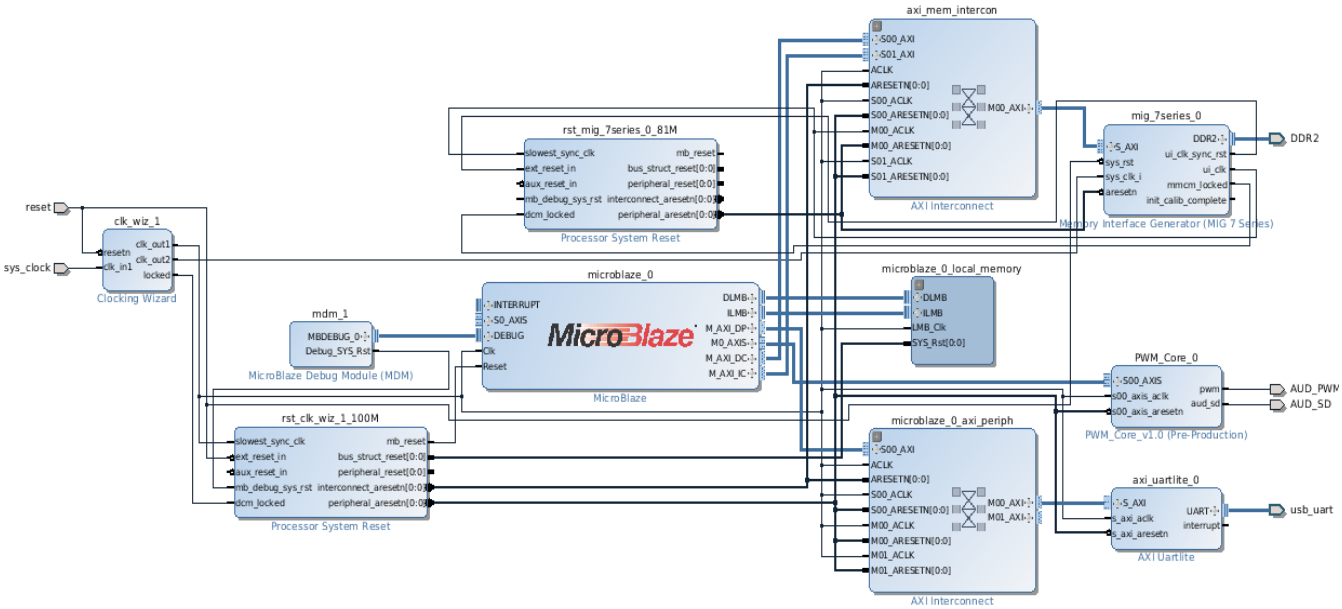


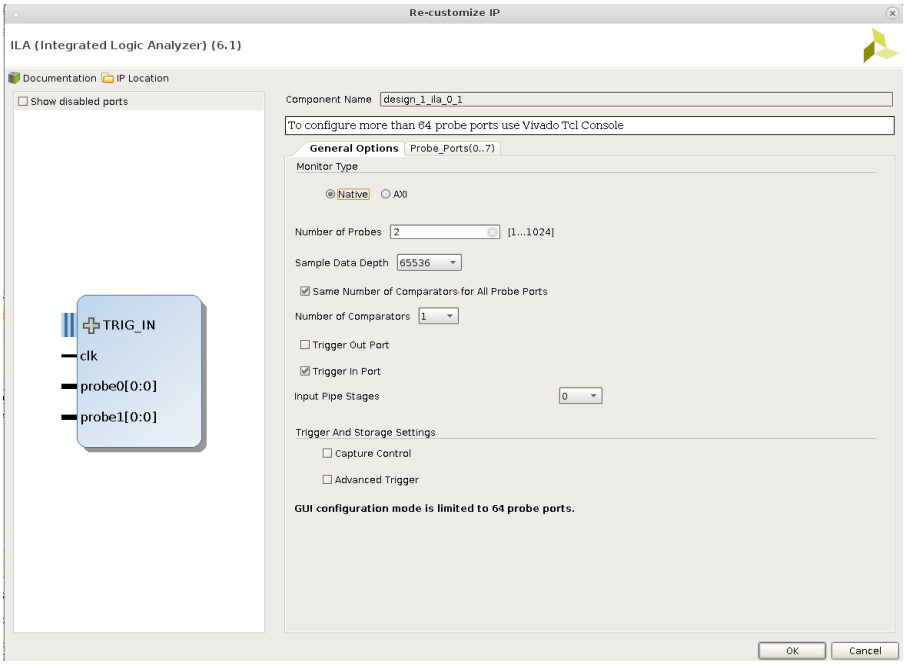
Figure 1-1: **ILA Core Symbol**

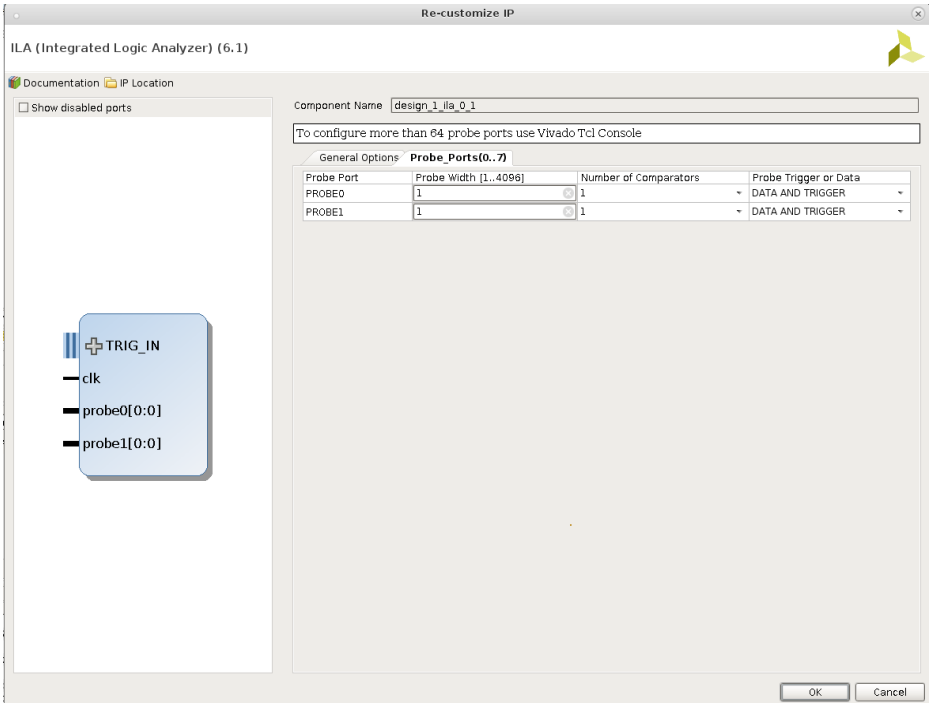
Connecting ILA Module to Previous Audio Project

1. Open the block diagram of the previous audio project

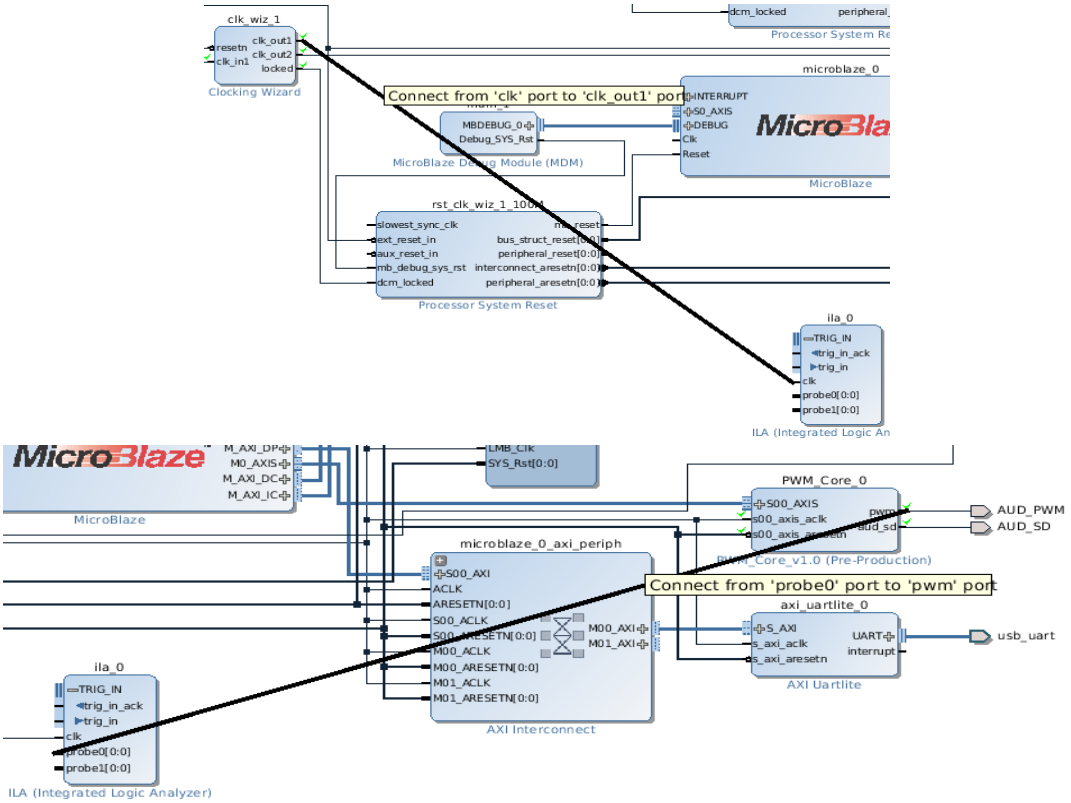


2. **Add IP**. Search for **ILA (Integrated Logic Analyzer)**
3. Double click on the ILA module and customize it.
    - Choose **Native** for Monitor Type
    - Change the **number of probes** to 2
    - Change **Sample Data Depth** to 64k
    - Check the **Trigger In Port**
    - Open **Probe** tab. Make sure it is same as in the picture

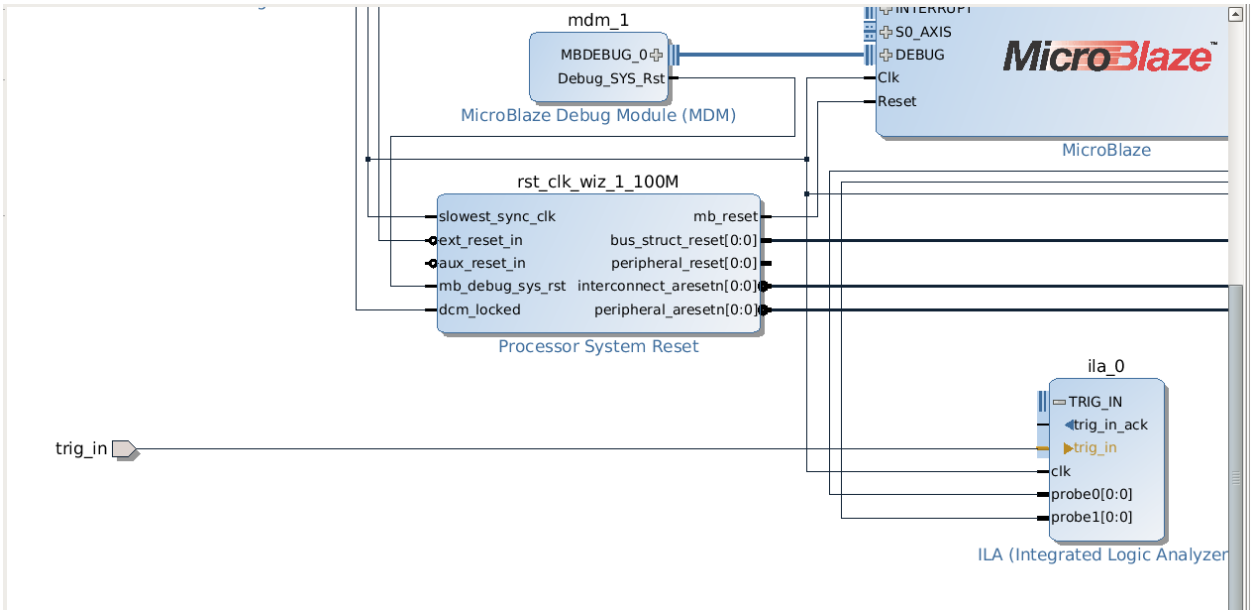




4. Connect the wires.
   - clk to system clock 100mhz. This clock indicates sampling rate and must be sufficient for the PWM core
   - Connect probe0 to AUD_PWM
   - Connect probe1 to AUD_SD



5. Expand trig_in. Right click trig in, make external.



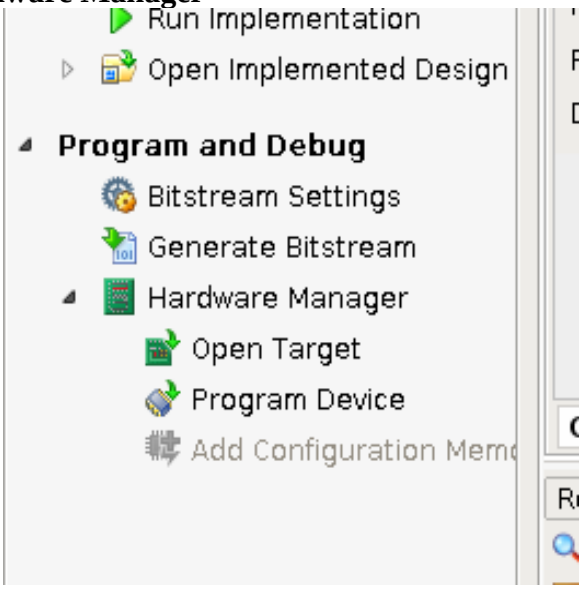
6. Change the constraint file. We connect the trig_in signal to SW[0] (Port J15).

```
1
2 ## Clock signal
3 set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
4 create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];
5
6 ##Switches
7 set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { trig_in }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
8
9 ##PWM Audio Amplifier
10 set_property -dict { PACKAGE_PIN A11   IOSTANDARD LVCMOS33 } [get_ports { AUD_PWM }]; #IO_L4N_T0_15 Sch=aud_pwm
11 set_property -dict { PACKAGE_PIN D12   IOSTANDARD LVCMOS33 } [get_ports { AUD_SD }]; #IO_L6P_T0_15 Sch=aud_sd
12
```

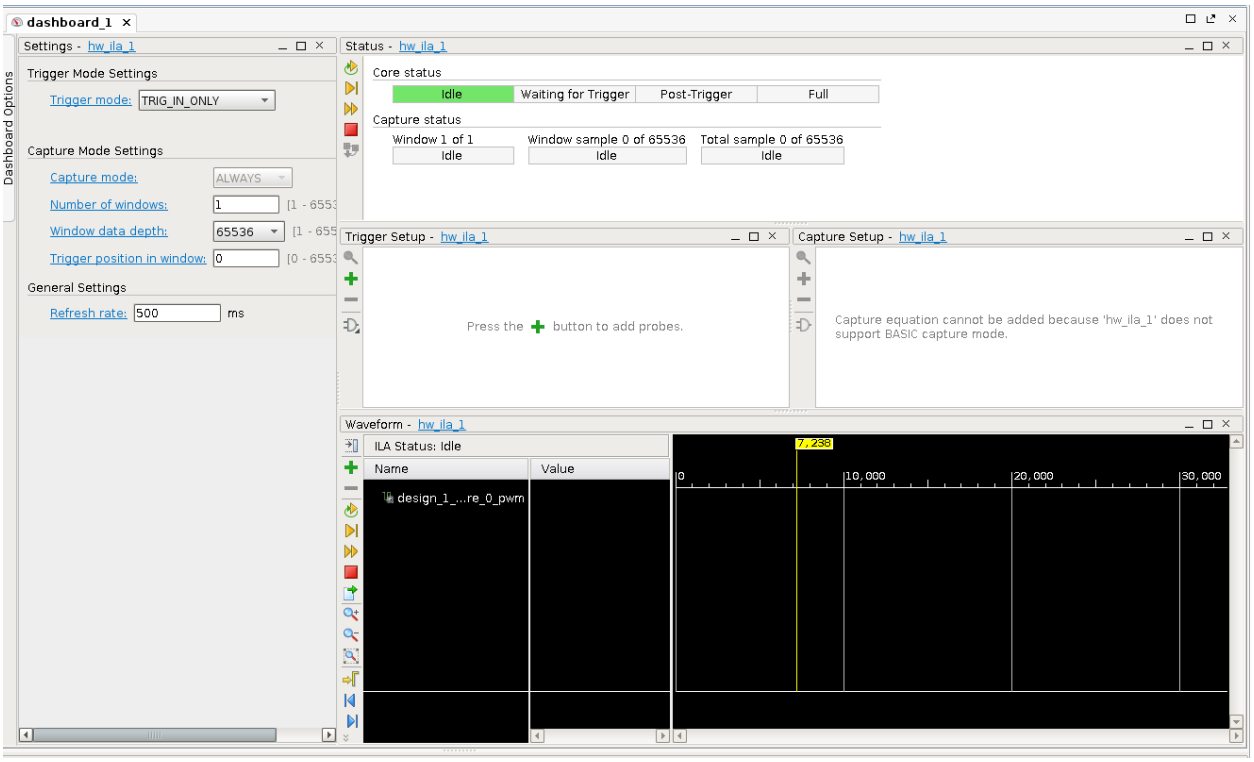
Remember to delete the original SDK folder from the project directory. Otherwise, there will be problems when launching SDK. We will rebuild the SDK project later.

7. Generate bit stream. Export to SDK, be sure to include the bitstream.

8. Going back to Vivado, make sure your device is connected and click **Program Device** under **Program and Debug > Hardware Manager**



Then, the new perspective of windows will be shown automatically as below:



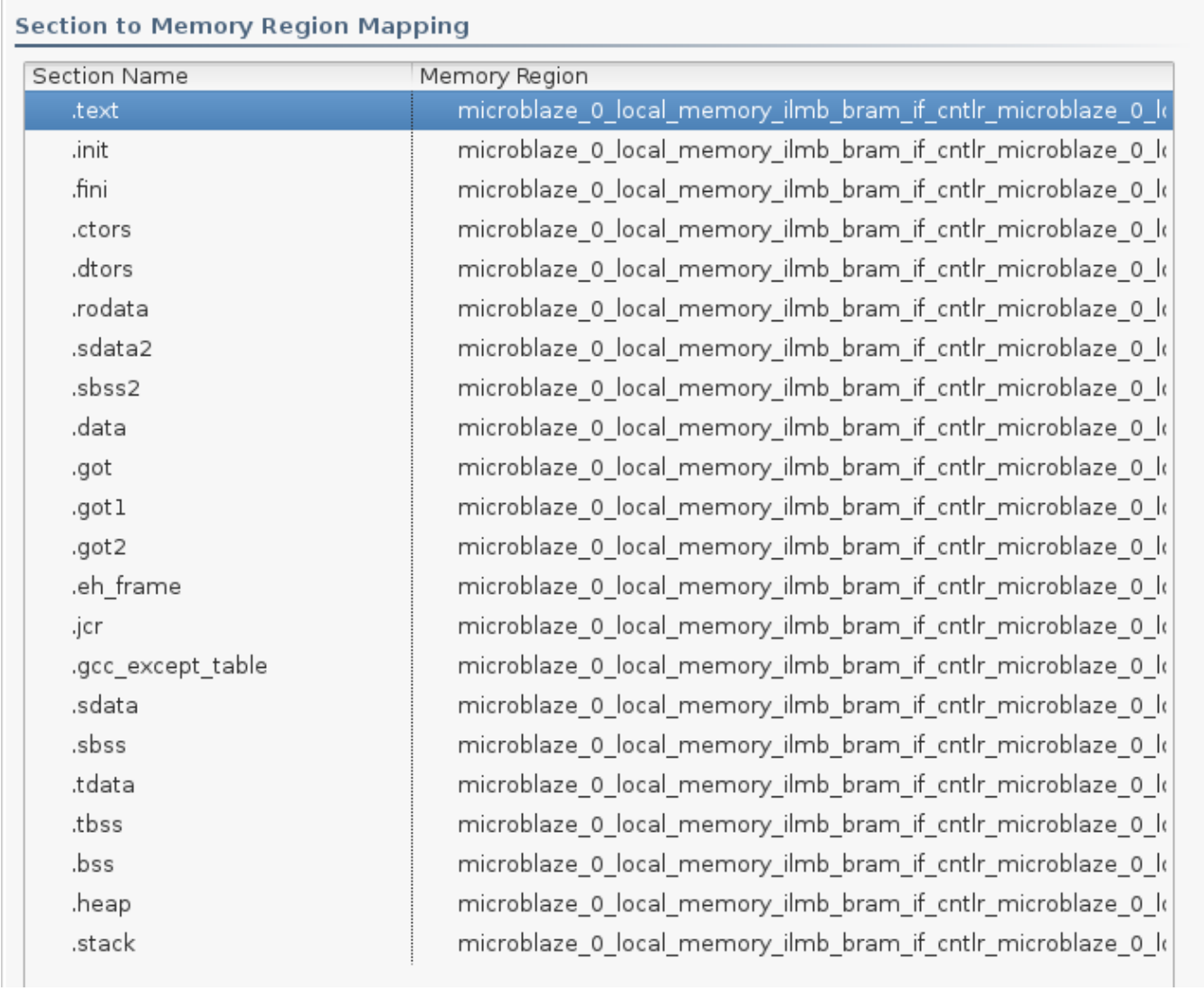
SDK Configuration

The SDK configuration is similar to the previous audio tutorial, except for a few changes.

1. Open SDK and create a new project (Xilinx Application). Make it an empty application.
2. Right click the new project to create a new C source file, give it a name main.c.
3. Copy and paste the provided main.c into the project.

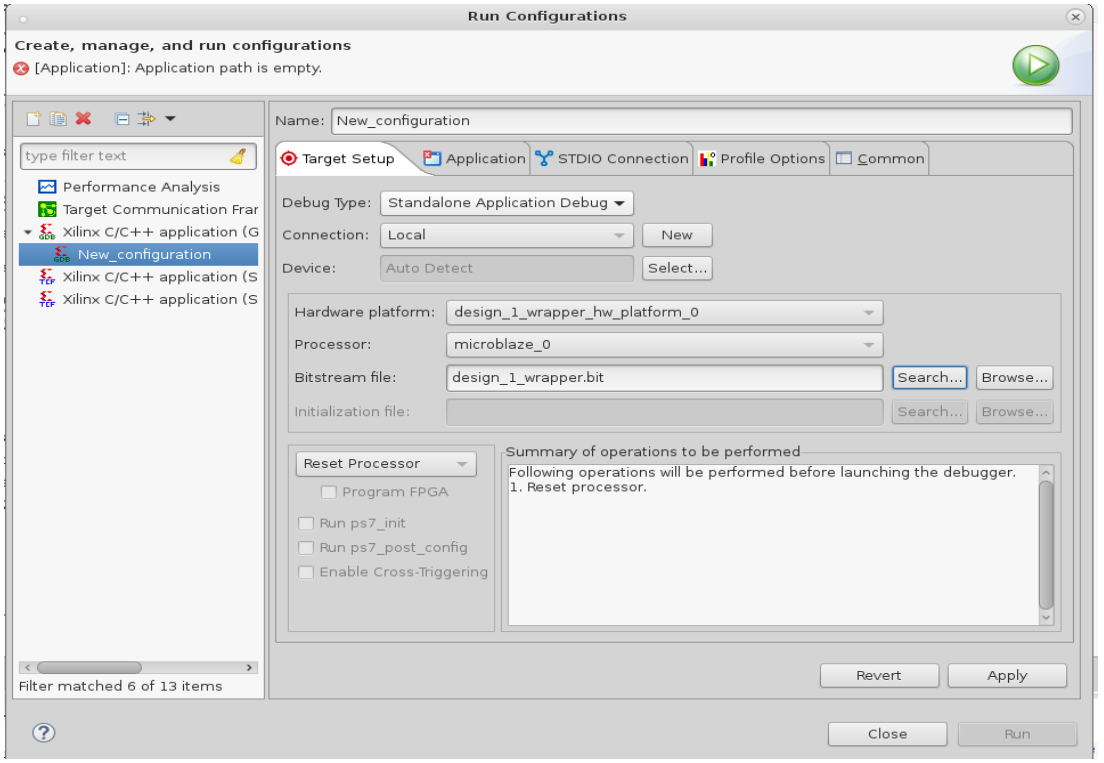
```
#include "fsl.h"
#include "xil_types.h"
#include "xstatus.h"
#include "xil_io.h"


#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;
}
```

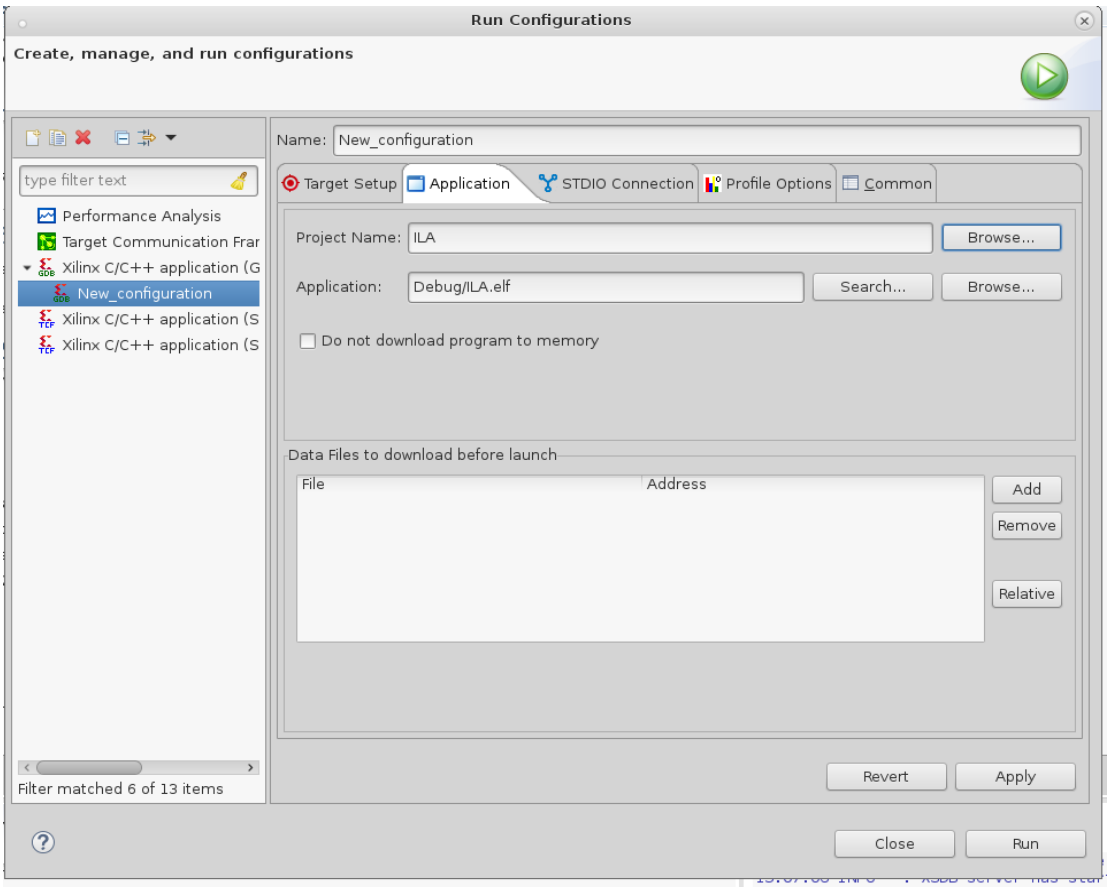
4. Open the lscript.ld (linker script) under src. Change all Memory Region Mapping to local memory as shown.



5. Save and build the project.

6. 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. **Note: Do not Program FPGA in SDK, since the FPGA should have already been programmed by using the Hardware Manager - Program Device in Vivado. Program the FPGA again in SDK might affect the functionality of the ILA module.**





7. Click Run. If you have connected your headphone to the audio jack, you should hear the same sound as produced from the previous audio tutorial.

Using ILA

There are multiple ways to start the ILA. In this tutorial, we will give only two examples, the other methods can be explored from the Xilinx user manual.

1. Make sure your SW[0] is low and the Trigger Mode is BASIC_OR_TRIG_IN

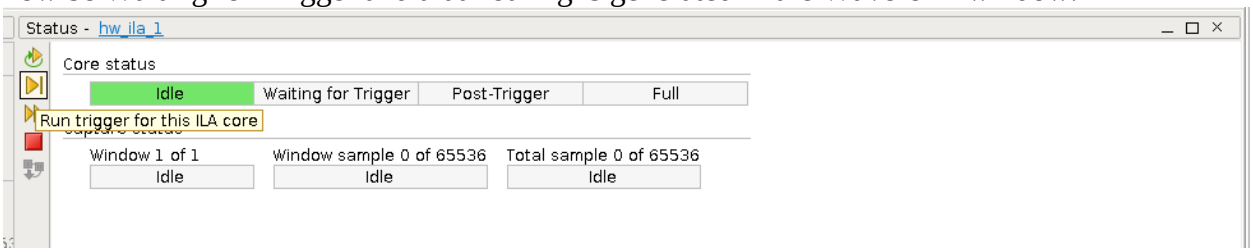
Trigger Mode of BASIC_OR_TRIG_IN means that the ILA can be evoked by both the trig_in signal that is connected to SW0 or a combinational logic of the probes.

2. Under Trigger Setup, click on the "+" sign and add AUD_SD. Recall that AUD_SD is always set to be logic 1 by the PWM IP.

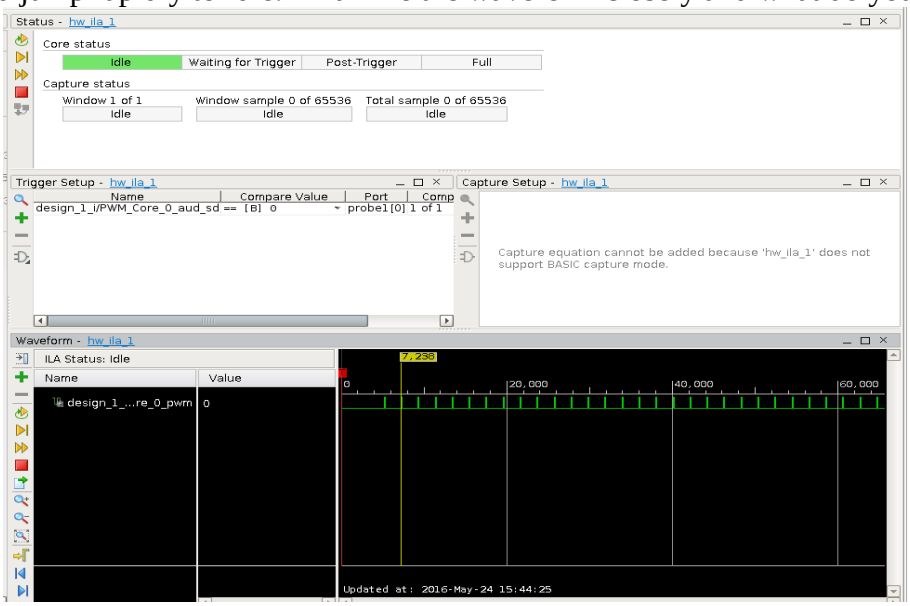
3. Under Trigger Setup -> Compare Value for AUD_SD, select Operator "==", Radix [B] and Value logic "0".

With the above setup, this ILA will only record data when either the SW0 (Trig_In) is high or AUD_SD is low. But since AUD_SD is always high, when we will start the ILA, we should see that the ILA will be in waiting for trigger state.

4. Under Status, click on **Run trigger for this ILA core**. You should see that the Core status should now be Waiting for Trigger and that nothing is generated in the Waveform window.



5. Now turn SW[0] to high. You should see that a sequence of waveform should be recorded and the core status should jump rapidly to Idle. Examine the waveform closely and what do you see?



6. Instead of using SW[0] to trigger the recording, you can also use the PWM output itself. Modify the Trigger Setup such that the ILA will start recording ONLY when PWM output become a logic "1".