Bluetooth, DMA, FFT, Float-point IP Tutorial

Peter Li

Team 5

Table of Contents

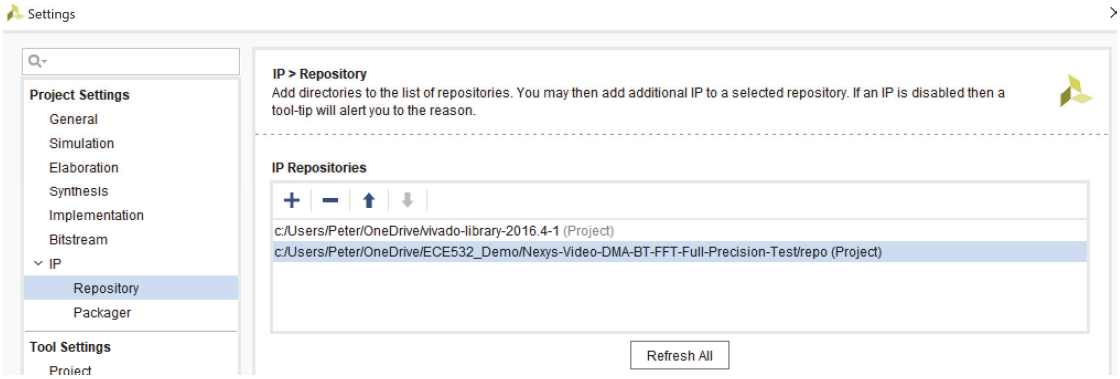
1. Introduction

This tutorial introduces the configuration, usage and API for the following Xilinx and Digilent IP using the Nexys Video Board:

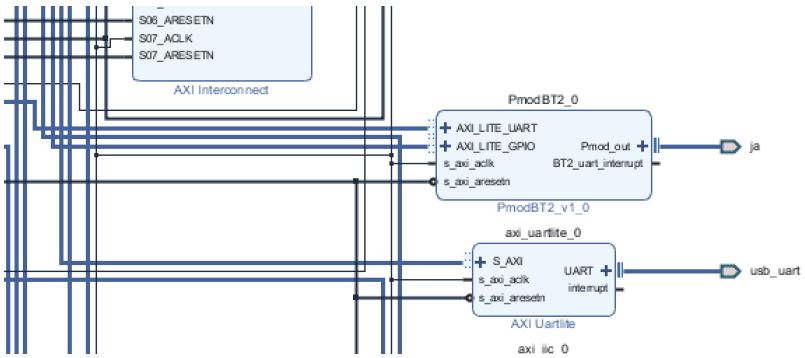
1. Bluetooth
2. DMA
3. FFT
4. Float-point IP

Project Setup for the Controller Board

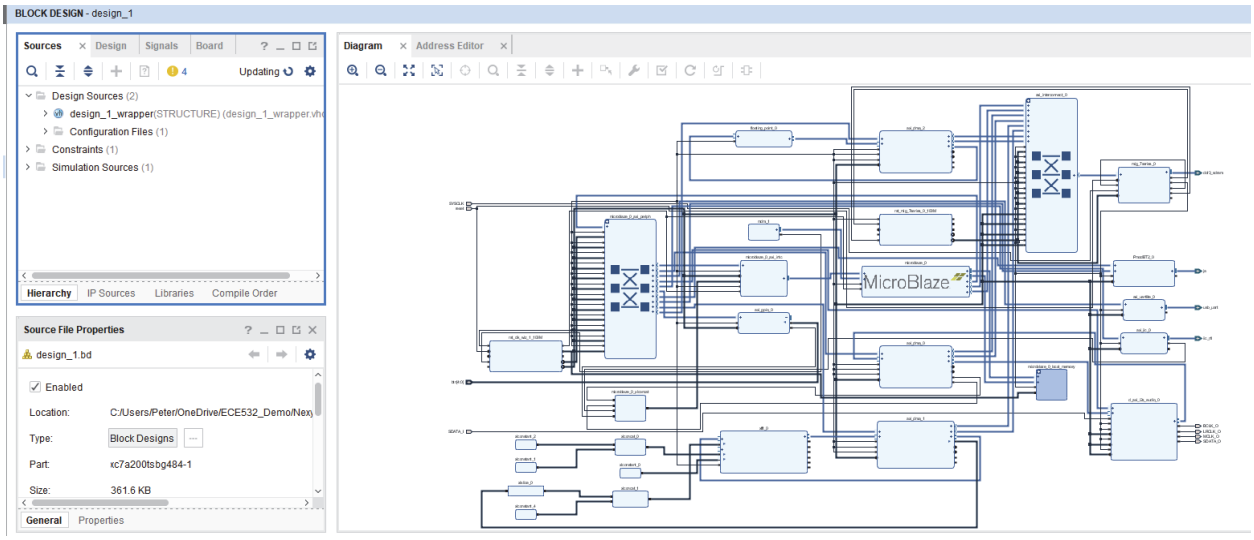
1. To start, download the project file at: https://github.com/CaptainPenguins/SimpleAudioRecognition
2. Place the project file in a directory path with no spaces
3. Open Vivaldo, cd to *.../SimpleAudioRecognition/VivadoProject/Nexys-Video-DMA-BT-FFT-Full-Precision-Test/proj* directory
4. Type *source create_project.tcl*
5. Add the path for the following two IP repository
    1) Digilent IPs (download from https://github.com/Digilent/vivado-library/releases)
    2) The repo folder inside the project (see picture below)



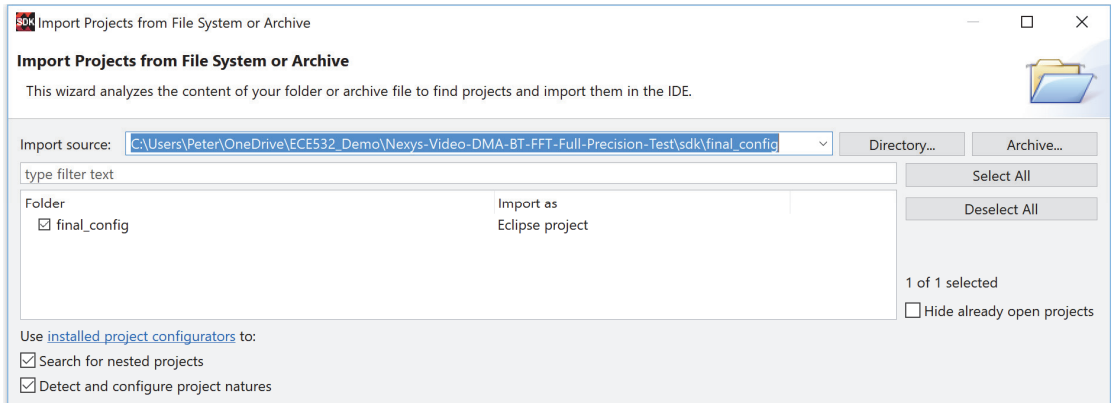
6. Make sure the Bluetooth module looks like the following

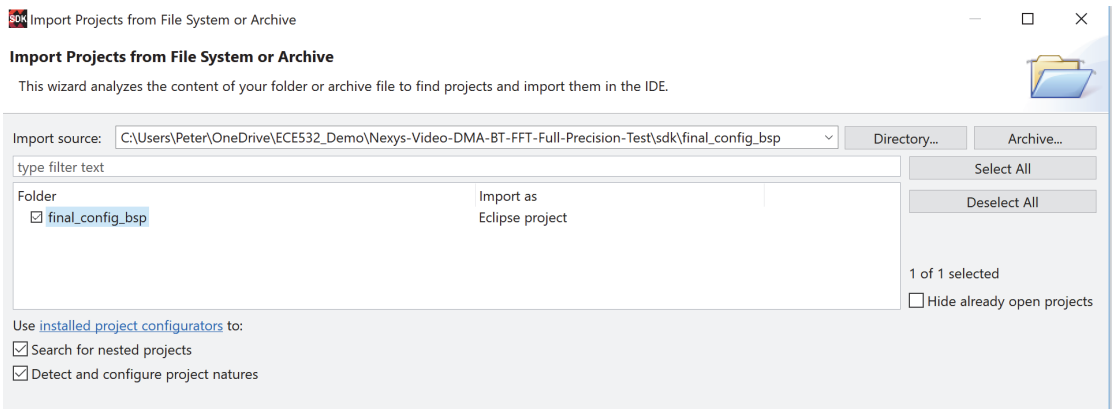


7. Validate the block design. If you see the “locked-ip” error, close and reopen Vivaldo to try again
8. Create HDL wrapper of the block design and generate bit-stream (if you encounter an error, try generating bit-stream again)

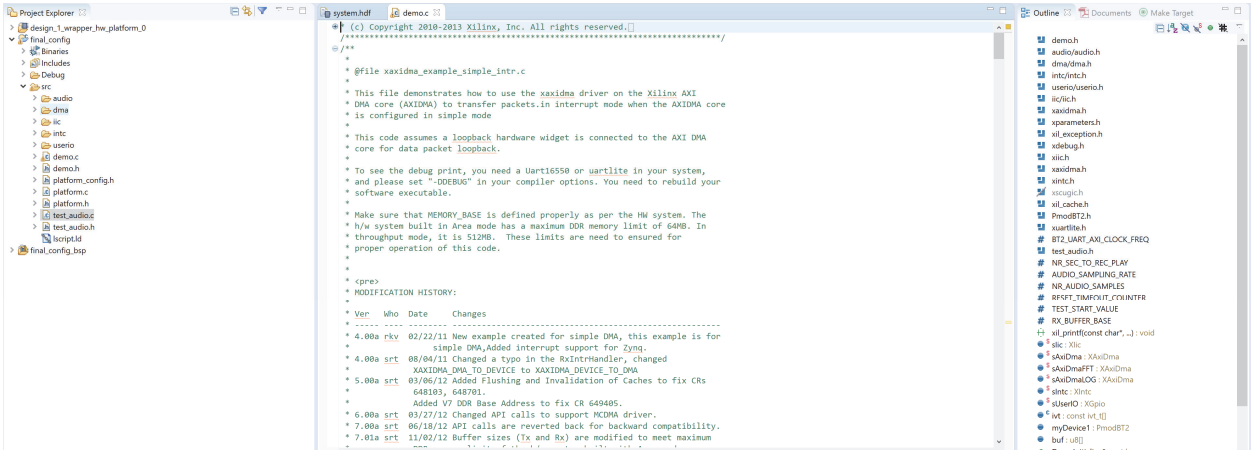


9. Export hardware and launch SDK
10. In the SDK window, go to File -> Import Project from File System
11. Import the two project folders inside the sdk folder

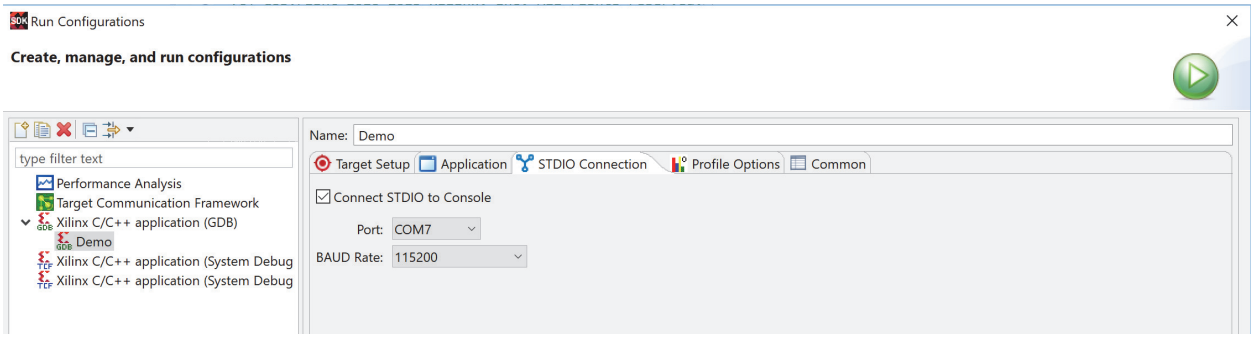




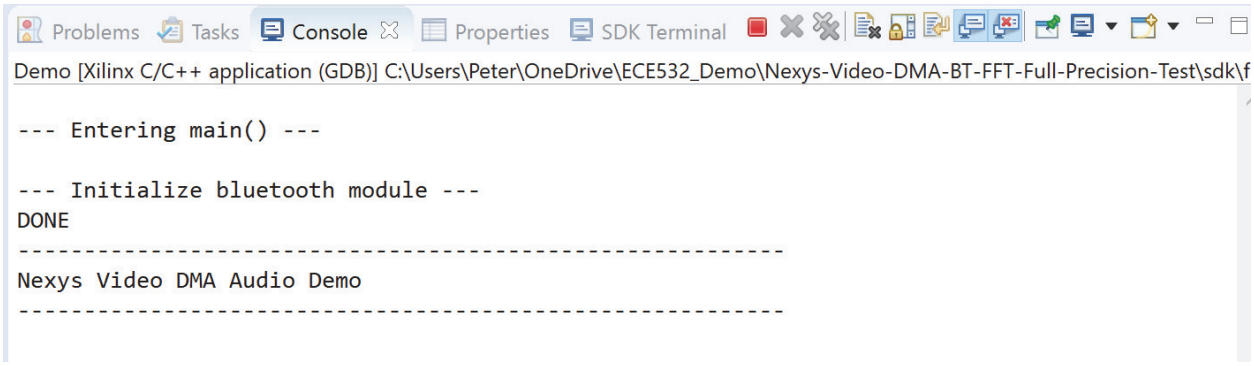
12. You should see the following in SDK window



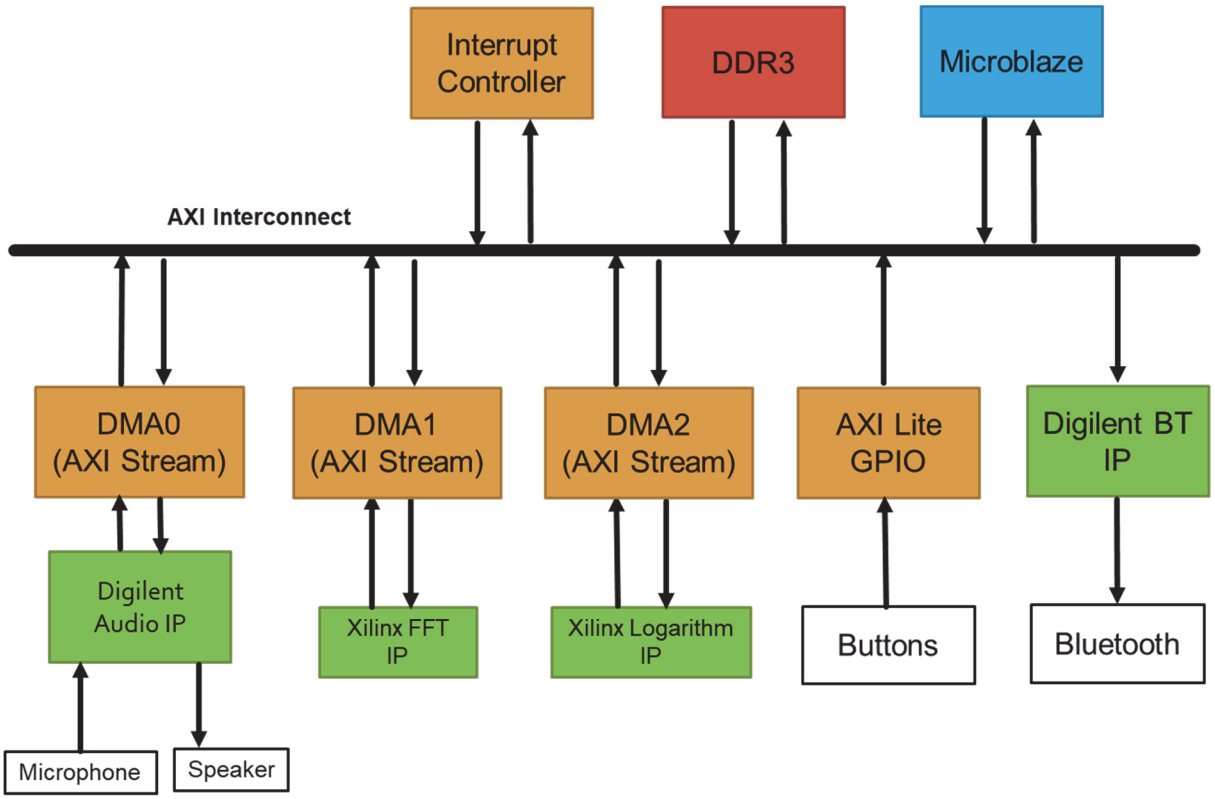
13. Program the FPGA
14. Run GDB debugger configuration with outputs to console



15. Press the BTNU on the board to run the IP test script for all modules after you see the following message on the console



Block Diagram for the Hub Board



Description of the Testing Program (demo.c)

1. After the button BTNU is pressed, the button will generate the interrupt to trigger the testing routine.
2. FFT test sequence in the file test_audio.c is loaded into DDR3 memory
3. A full precision 512-point FFT is performed on the test sequence using DMA1
4. The real and imaginary part of the FFT result is printed in the console
5. The float-point IP configured as a logarithm module is tested using the DMA2
6. The integer part of the logarithm result is printed in the console
7. Bluetooth sends the character ‘A’

2. Bluetooth Module

Initial Setup

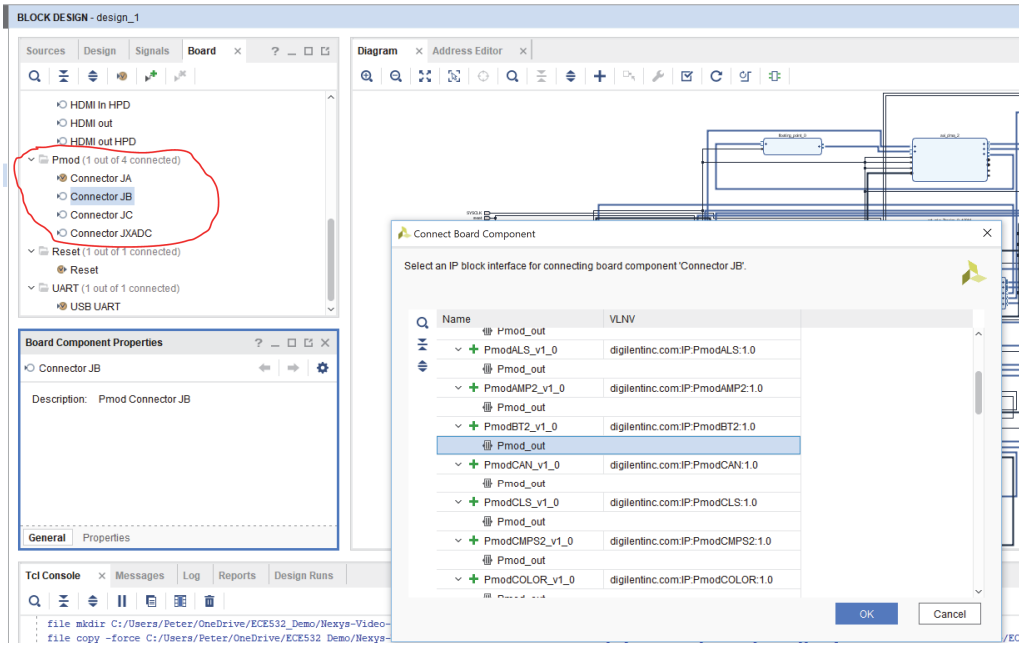
1. Plug the Bluetooth module on JA
2. Run the test routine once
3. Pair the Bluetooth module to another device

Connecting 2 Bluetooth PMOD

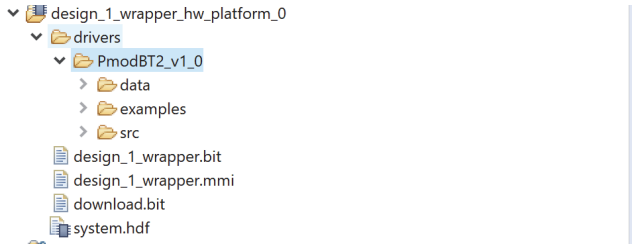
1. Follow the tutorial at: https://reference.digilentinc.com/learn/programmable-logic/tutorials/auto-connect-2-bt2_s/start
2. **Make sure to turn both Bluetooth module on at the same time to auto-connect!**

Setting Up Bluetooth PMOD

1. Make sure you set up the Digilent IP repository in Part 1
2. Right click on the PMOD connector that you want (JA in this case), and select the BT2 PMOD



3. Validate design and generate bitstream
4. Open SDK and make sure that the xparameters.h contains the newly added Bluetooth PMOD
5. Notice the location of the Bluetooth driver with sample code



APIs for the Bluetooth Module

The following code sets up the required library files for the Bluetooth PMOD:

```
/************************** Include from Bluetooth Configuration **************/
#include "xil_cache.h"
#include "PmodBT2.h"
#include "xuartlite.h"
#define BT2_UART_AXI_CLOCK_FREQ XPAR_CPU_M_AXI_DP_FREQ_HZ
```

The following APIs initialize the Bluetooth PMOD and enables the Bluetooth PMOD to send / receive a single character stored in buf.

```
PmodBT2 myDevice1;
u8 buf[1];

void DemoInitialize()
{
    //EnableCaches();
    BT2_Begin (
        &myDevice1,
        XPAR_PMODBT2_0_AXI_LITE_GPIO_BASEADDR,
        XPAR_PMODBT2_0_AXI_LITE_UART_BASEADDR,
        BT2_UART_AXI_CLOCK_FREQ,
        115200
    );
}

void Bluetooth_send(u8* buf)
{

    xil_printf("Initialized PmodBT2 Demo\r\n");
    xil_printf("Sent from BT1 on JA: %c\r\n", (char) buf[0]);
    BT2_SendData(&myDevice1, buf, 1);
}

void Bluetooth_recieve(u8* buf)
{

    xil_printf("Starts receiving\r\n");
    int n = BT2_RecvData(&myDevice1, buf, 1);
    //Pooling for data
    while (n == 0){
        n = BT2_RecvData(&myDevice1, buf, 1);
    }
    xil_printf("Receive from BT1 on JA: %c\r\n", (char) buf[0]);
}
```

3. DMA Module

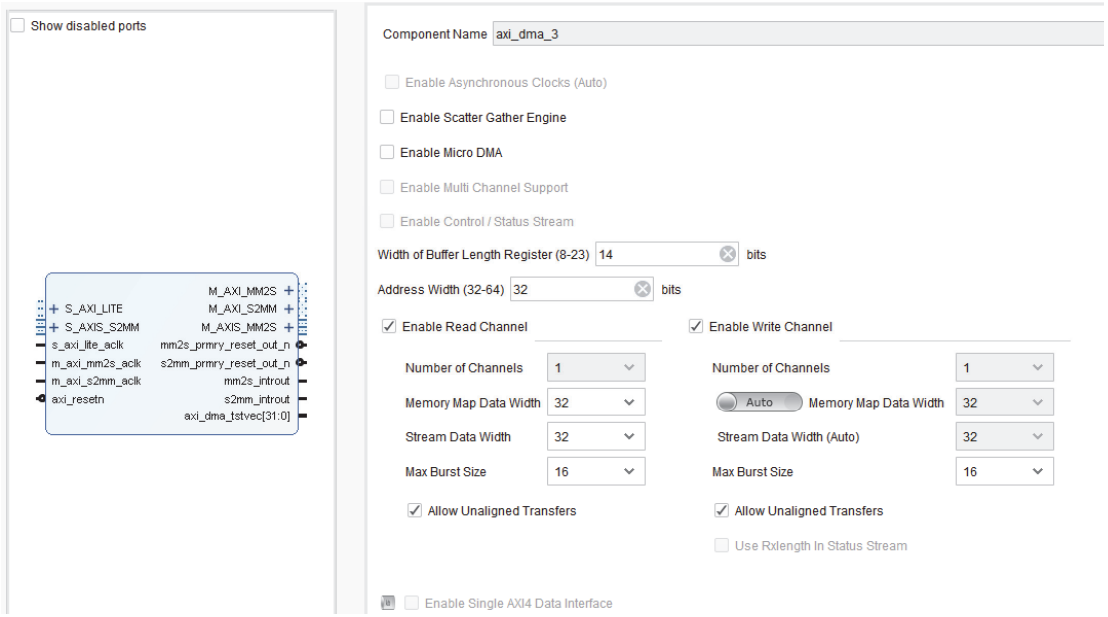
DMA is the best module for streaming data between the memory and the AXI-stream enabled IPs. Ideally, you should use a separate DMA for each AXI-stream enabled IP.

Documentation

https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf

Initial Setup

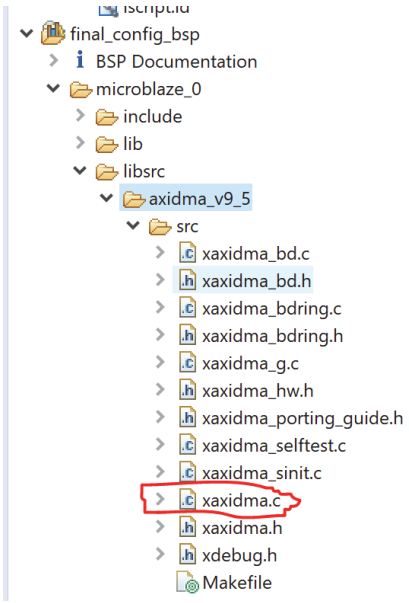
1. Open block design
2. Add AXI Direct Memory Access block
3. Configure the DMA in the following way. The Stream Data Width for the read channel should match the input data width of the AXI-Stream IP. The Stream Data Width for the write channel will automatically adapt to the output data width of the AXI-Stream IP.



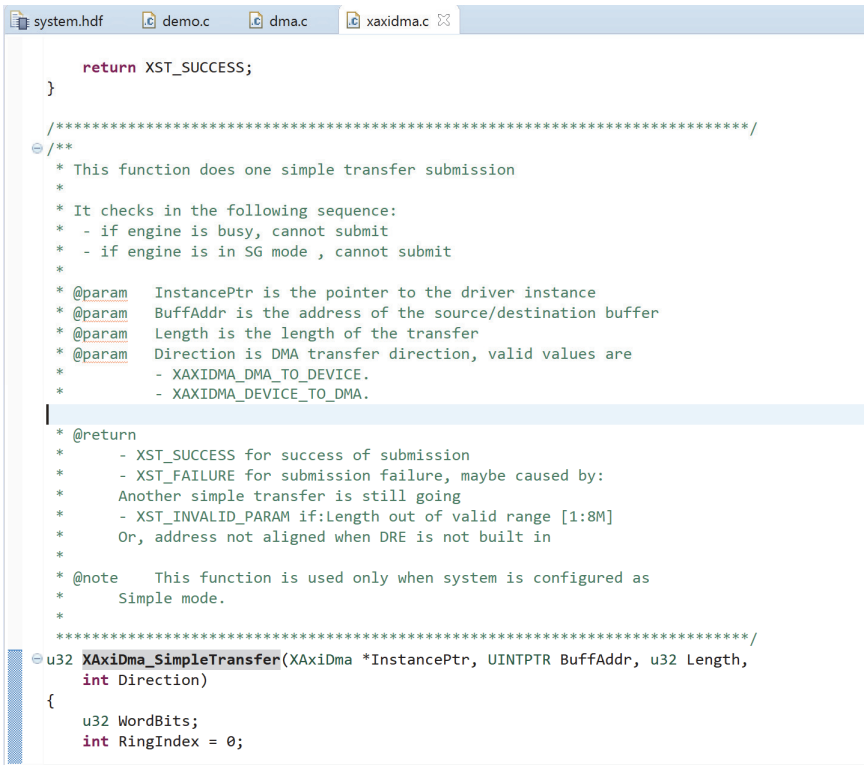
4. Run Connection automation
5. Connect the input and the output of the AXI-Stream IP to the port AXIS_MM2S and AXIS_S2MM
6. See tutorial about the DMA concept: http://www.fpgadeveloper.com/2014/08/using-the-axi-dma-in-vivado.html

APIs for DMA

When the DMA is added to the block diagram and bit-stream is regenerated, Xilinx DMA driver is automatically added to the project BSP (see below). The most important file for DMA control is xaxidma.c, which contains APIs that can be used to stream data bidirectionally using the DMA.



In particular, the XAxiDma_simpleTransfer is most used in my design. See my code in demo.c on how to initialize DMAs.



4. FFT Module

Using the FFT Module can greatly speed up FFT computation compared with the software C code implementation. The Xilinx FFT IP supports three modes of operations.

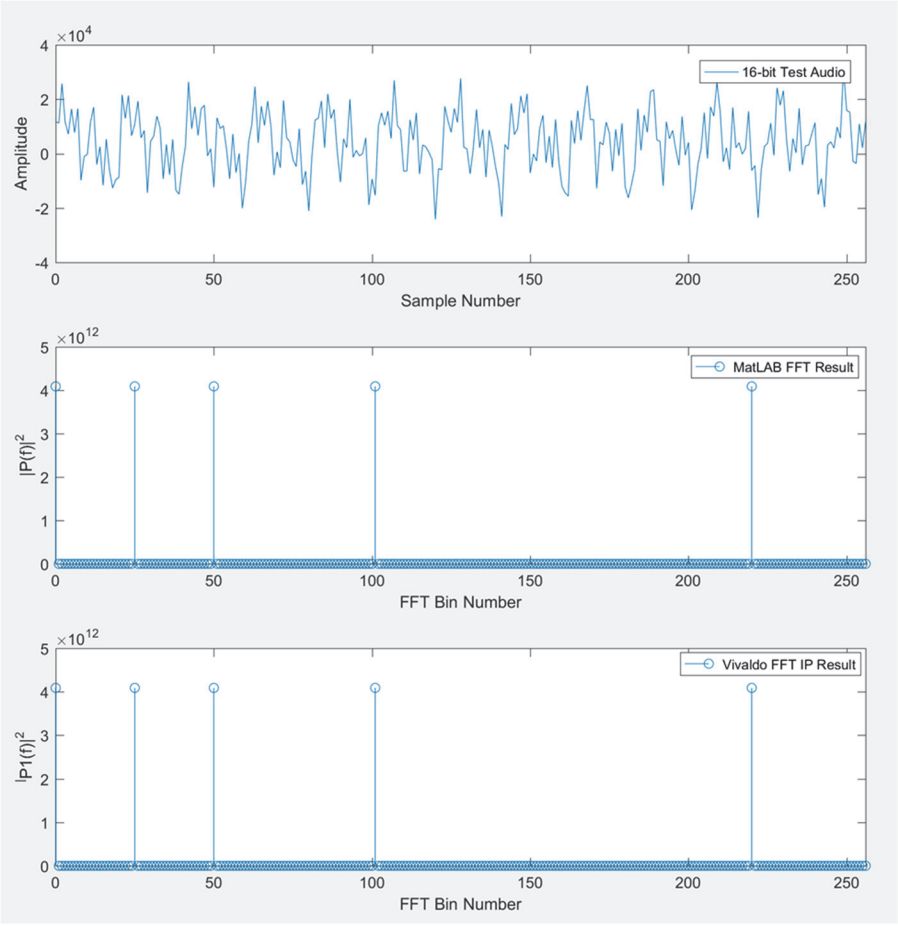
Documentation

https://www.xilinx.com/support/documentation/ip_documentation/xfft/v9_0/pg109-xfft.pdf

Three Modes of FFT Operation

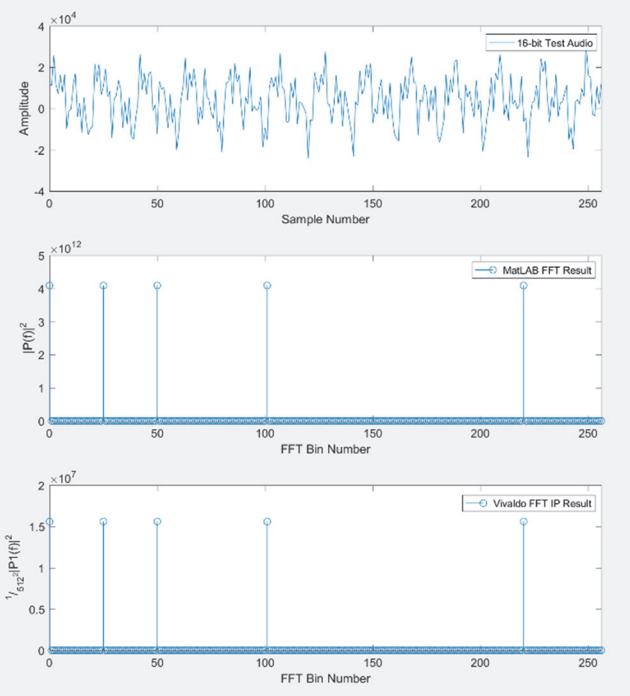
The three FFT computational modes for a fixed test sequence in test_audio.c is showcased below with comparison to MATLAB FFT results. **For most applications, I recommend using the full precision unscaled arithmetic mode**.

1. Full precision unscaled arithmetic
    - Results matches very closely with MATLAB (error less than 0.1%)
    - Input requires 16-bits fixed-point representation
    - Output contains real and imaginary part, 32-bits each
    - No scaling schedule required



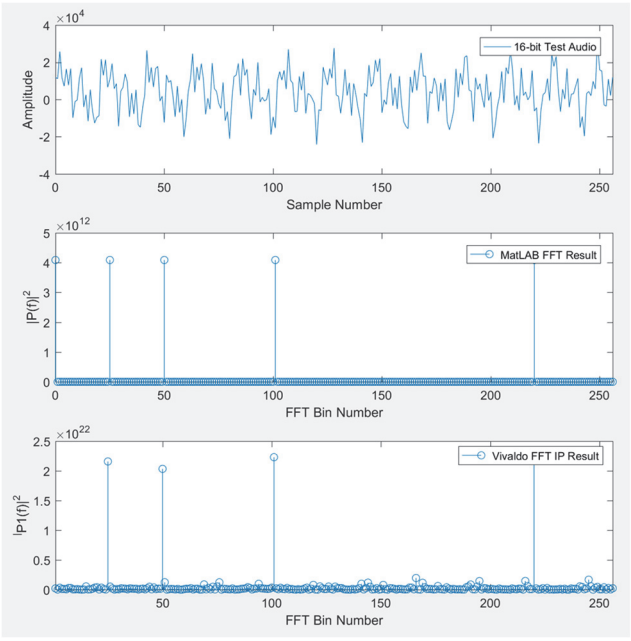
2. Scaled fixed-point (user provide scaling schedule)

    - Input requires 16-bits fixed-point representation
    - Output contains real and imaginary part, 16-bits each using the recommended scaling schedule. Thus, we lose the resolution of the lower 16-bits compared with the fully unscaled 32-bits outputs.
    - Potentially large relative errors compared with MATLAB



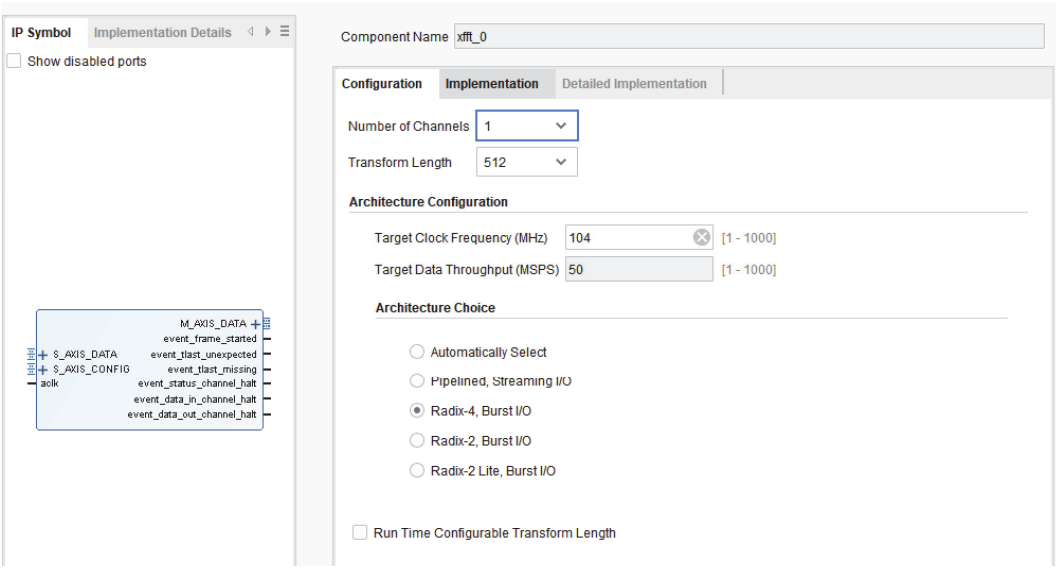
3. Block floating-point (automatically scaled)

    - Input requires 32-bits single precision floating-point number
    - Uncontrollable scaling of the output (input dependent and unknown to user)
    - Presence of computational “noise” in the output spectrum (see datasheet for more explanation)

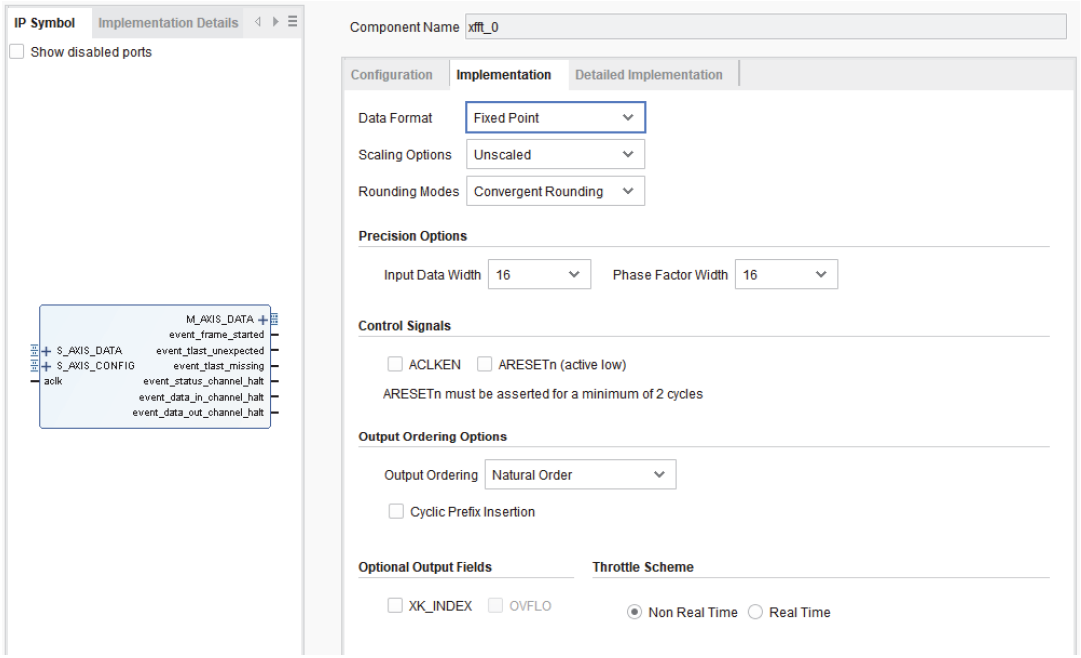


Initial Setup for Full Precision Unscaled Arithmetic Mode

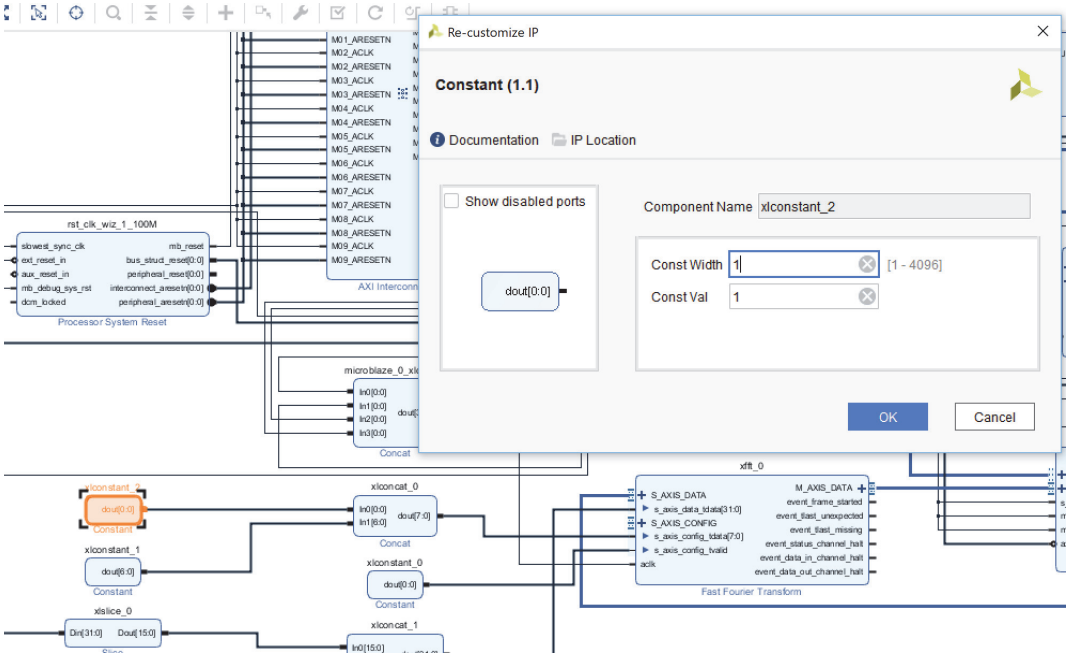
1. Open block design and add the FFT IP
2. Configure the FFT module shown below. Since I am bursting 512 input data at a time, I am using the radix-4 FFT architecture



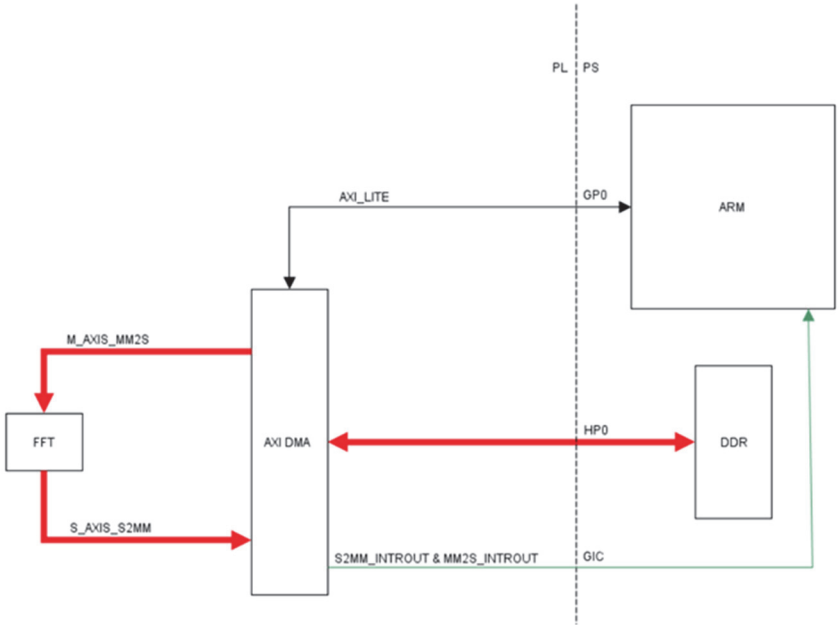
3. Since I am using fixed-point unscaled arithmetic mode and I would like to get the output in natural order, I configure the FFT implementation tab in the following way.



4. To configure the FFT module computing forward FFT operation, configure the first bit of the a_axis_config_data[7:0] to 1.



5. Add a new DMA and connect this IP to it (see Part 3)



API for the FFT IP

```
/********************** Peter's FFT Function ************************************/
int FFT(u32 *TxBufferPtr, u32 *RxBufferPtr)
{
     *  This function computes a 512-point FFT using by sending consecutive input data

    // Sending 512 input data from TxbufferPtr to the FFT IP
    xil_printf("\r\nFFT Start");
    int Status;
    Status = XAxiDma_SimpleTransfer(&sAxiDmaFFT,(u32) TxBufferPtr,
                4*512, XAXIDMA_DMA_TO_DEVICE);

    if (Status != XST_SUCCESS) {
        xil_printf("\r\nFFT Write Failed");
        return XST_FAILURE;
    }
    while ((XAxiDma_Busy(&sAxiDmaFFT,XAXIDMA_DMA_TO_DEVICE))) {
            /* Wait */
    }

    // Reading 512 output data from FFT IP and store them in RxBufferPtr
    Status = XAxiDma_SimpleTransfer(&sAxiDmaFFT,(u32) RxBufferPtr,
                2*4*512, XAXIDMA_DEVICE_TO_DMA);
    if (Status != XST_SUCCESS) {
        xil_printf("\r\nFFT Read Failed");
        return XST_FAILURE;
    }

    while ((XAxiDma_Busy(&sAxiDmaFFT,XAXIDMA_DEVICE_TO_DMA))) {
            /* Wait */
    }

    //Flush cache
    microblaze_flush_dcache();
    microblaze_invalidate_dcache();

    return XST_SUCCESS;
}
```

5. Floating-point IP

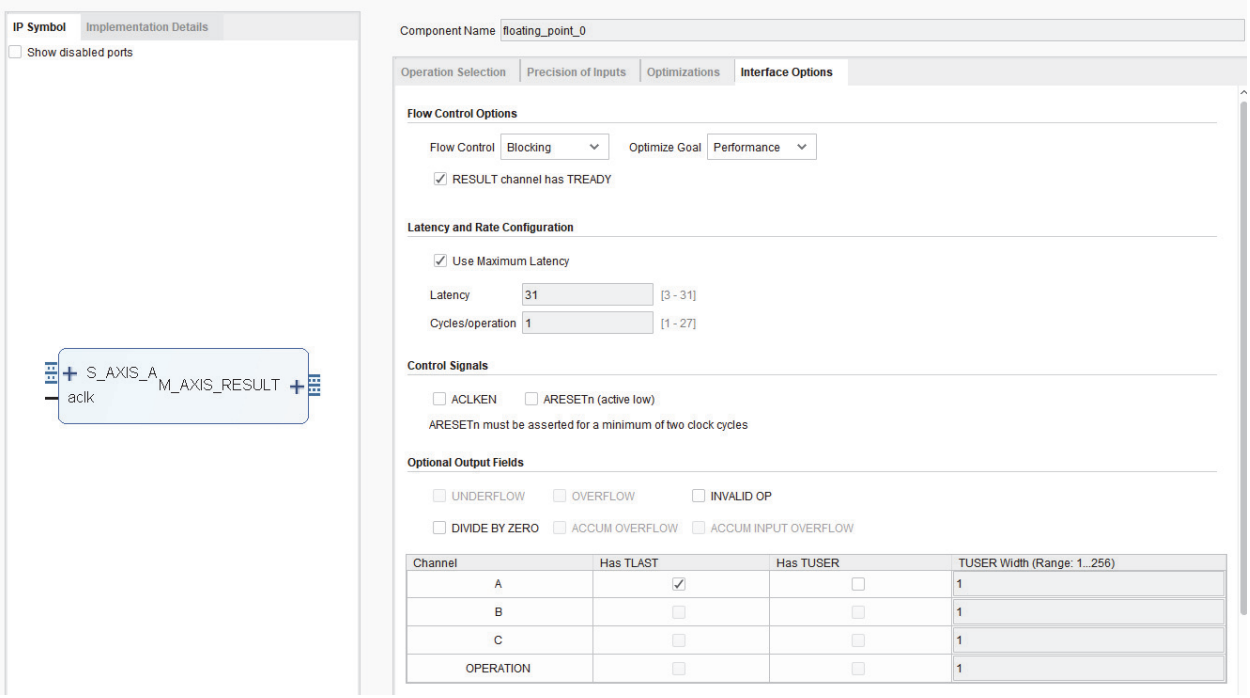
Floating point operation in Microblaze is extremely slow. Using the Floating-point IP can speed up several commonly used floating point operations. In this section, I will show use how to configure the Floating-point IP for logarithm operation. Since the Floating-point IP contains AXI-Stream interface, we need to interface it with DMA.

Documentation

https://www.xilinx.com/support/documentation/ip_documentation/floating_point/v7_0/pg060-floating-point.pdf

Initial Setup

6. Open block design and add the Floating-point IP
7. Choose the desired operation (Logarithm in this case)
8. Precision of Inputs -> Single Precision
9. Optimizations -> DSP Slices -> Full Usage
10. Configure Interface Options as shown below (**choose maximum latency to ensure the results are consistently correct!**)



11. Add a new DMA and connect this IP to it (see Part 3)

APIs for Floating-point IP using DMA

Assume you have "length" number of data stored in the address TxBufferPtr and you want to use the Floating-point IP to compute the logarithm of these data and store them from the address RxBufferPtr, you can call the Log function shown below. It uses the DMA module to read and write data from the memory to the Floating-point IP. **Remember to flush the cache before and after each transfer!**

```
/*********************** Peter's LOG Function ************************************/
int LOG(float *TxBufferPtr, float *RxBufferPtr, u32 Length)
{
    /*
     *  This function computes the natural logarithm (ln) for an array of Length
     *  Both address pointers should point to float values
     */

    //Flush cache
    microblaze_flush_dcache();
    microblaze_invalidate_dcache();

    xil_printf("\r\nLOG Start");

    // Write input data from TxBufferPtr to Floating-point IP
    int Status;
    Status = XAxiDma_SimpleTransfer(&sAxiDmaLOG,(u32) TxBufferPtr,
                sizeof(float)*Length, XAXIDMA_DMA_TO_DEVICE);

    if (Status != XST_SUCCESS) {
        xil_printf("\r\nLOG Write Failed");
        return XST_FAILURE;
    }

    //Flush cache
    microblaze_flush_dcache();
    microblaze_invalidate_dcache();


    // Reading output data from Floating-point IP and store them in RxBufferPtr
    Status = XAxiDma_SimpleTransfer(&sAxiDmaLOG,(u32) RxBufferPtr,
                sizeof(float)*Length, XAXIDMA_DEVICE_TO_DMA);
    if (Status != XST_SUCCESS) {
        xil_printf("\r\nLOG Read Failed");
        return XST_FAILURE;
    }


    //Flush cache
    microblaze_flush_dcache();
    microblaze_invalidate_dcache();

    return XST_SUCCESS;
}
```