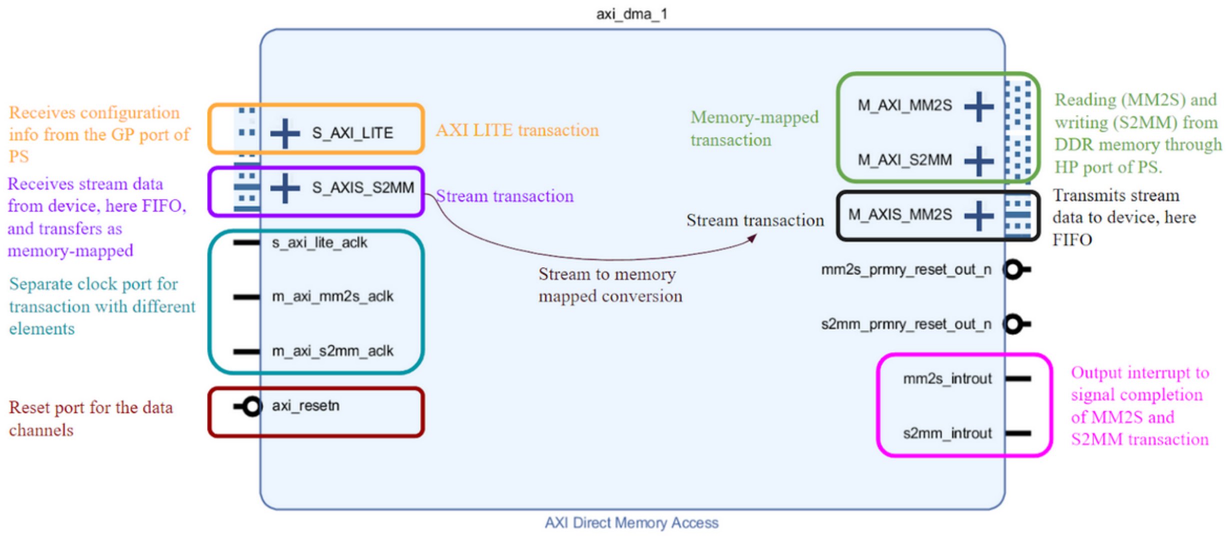
ELD Lab 11 Handout

Introduction to Lab:
In this lab we will be using FFT IP to perform the 8 point FFT. To perform data transfers between PS and PL, we will be using AXI DMA. First some data will be stored in the DDR memory of the float complex data type. This data will be accessed by the DMA via the HP port of the Zynq Processing system. The DMA will then write this data to the FFT IP in the PL part. Then the DMA will read this data from the FFT, and write it back to the DDR memory again.

Note: Float complex data type concatenates the real and imaginary part together with the upper 32 bits being imaginary and the lower 32 bits being real part of the complex number.

We will also perform the 8 Point FFT in PS using a testbench and then we will compare the software and hardware results.

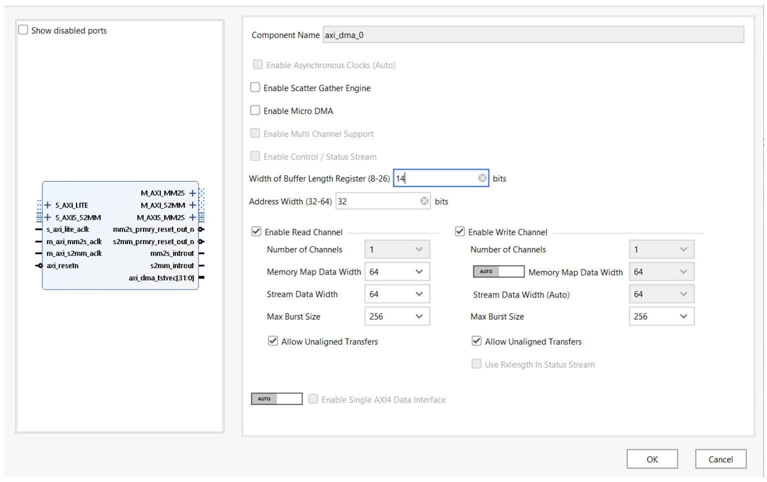
Before starting with the implementation, let’s revise the various ports of DMA IP


axi_dma_1
Receives configuration info from the GP port of PS
S_AXI_LITE
AXI LITE transaction
Receives stream data from device, here FIFO, and transfers as memory-mapped
S_AXIS_S2MM
Stream transaction
Separate clock port for transaction with different elements
s_axi_lite_aclk
m_axi_mm2s_aclk
m_axi_s2mm_aclk
Reset port for the data channels
axi_resetn
Stream to memory mapped conversion
Memory-mapped transaction
M_AXI_MM2S
M_AXI_S2MM
Reading (MM2S) and writing (S2MM) from DDR memory through HP port of PS.
Stream transaction
M_AXIS_MM2S
Transmits stream data to device, here FIFO
mm2s_prmry_reset_out_n
s2mm_prmry_reset_out_n
mm2s_introut
s2mm_introut
Output interrupt to signal completion of MM2S and S2MM transaction
AXI Direct Memory Access

Steps:

1. Open Vivado 2019.1. Create New Project
2. Click Next.
3. Select Zybo Z7-10 board from the board’s tab in the default part window
4. Create project by clicking on Finish

5. In the project , now click on the block design
6. Add the Zynq Processing System IP
7. Open the customization window by double clicking on the IP and make the following changes.
    a. In the PS-PL section, expand **HP Slave AXI Interface**, and check the **S AXI HP Interface** to add a slave HP port to the Zynq IP. This port will be used by the DMA to access the DDR memory.
    b. In the Interrupts section, expand the **Fabric Interrupts**, further expand the **PL-PS Interrupt Ports**, and check the **IRQ_F2P[15:0]**. This will add an interrupt port to the Zynq IP which will receive the interrupt signals from the DMA to indicate completion of the data transfers (only required if we want interrupt mode).
    c. Click on Ok.
8. Now add AXI Direct Memory Access IP, and add it to the design
9. Open its configuration window and make the following changes
    a. Deselect the “Enable Scatter/Gather mode”
    b. Increase the maximum burst size to 256
    c. Increase the Memory Map data width and stream data width to 64
        i. This is because now we are sending complex values which are of 64 bits (32 bits imaginary and 32 bits real concatenated)



Component Name axi_dma_0

Enable Asynchronous Clocks (Auto)

Enable Scatter Gather Engine

Enable Micro DMA

Enable Multi Channel Support

Enable Control / Status Stream

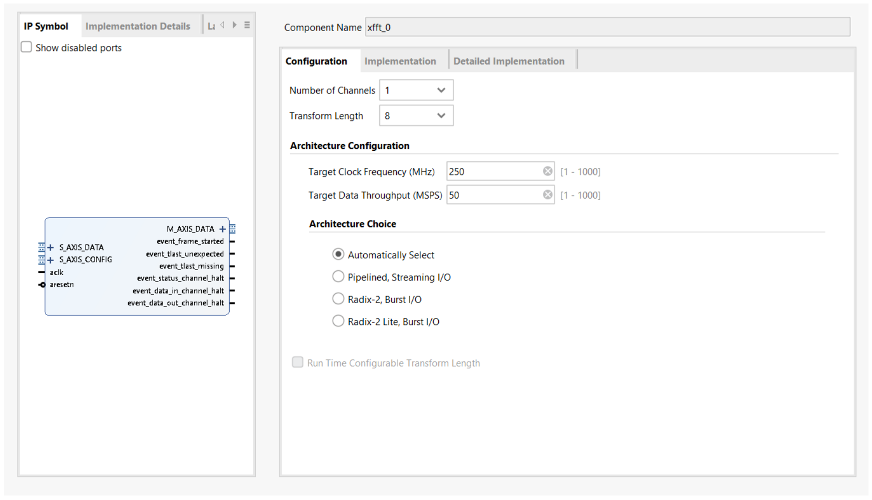
Width of Buffer Length Register (8-26) 14 bits

Address Width (32-64) 32 bits

[✓] Enable Read Channel		[✓] Enable Write Channel	
Number of Channels	1	Number of Channels	1
Memory Map Data Width	64	Memory Map Data Width	64
Stream Data Width	64	Stream Data Width (Auto)	64
Max Burst Size	256	Max Burst Size	256
[✓] Allow Unaligned Transfers		[✓] Allow Unaligned Transfers	
		Use Rxlength In Status Stream	

Enable Single AXI4 Data Interface

10. Click on run block automation. This will enable the DDR and FIXED_IO ports of the Zynq IP for external access
11. Then click on Run Connection automation
    a. This will add a processor system reset IP which will provide a reset signal for all the IP blocks added in the design.
    b. An AXI Interconnect IP will be added which will receive the configuration information of the DMA from the processor like its base address which will be used to access the DMA. The AXI Interconnect then passes this information to the DMA through its S_AXI_LITE port.
12. Now add the Fast Fourier Transform IP
13. Open its configuration window and make the following changes
    a. Change the transform length to 8 since we are performing 8-point FFT



Component Name xfft_0

Number of Channels 1

Transform Length 8

Architecture Configuration

Target Clock Frequency (MHz) 250 [1 - 1000]

Target Data Throughput (MSPS) 50 [1 - 1000]

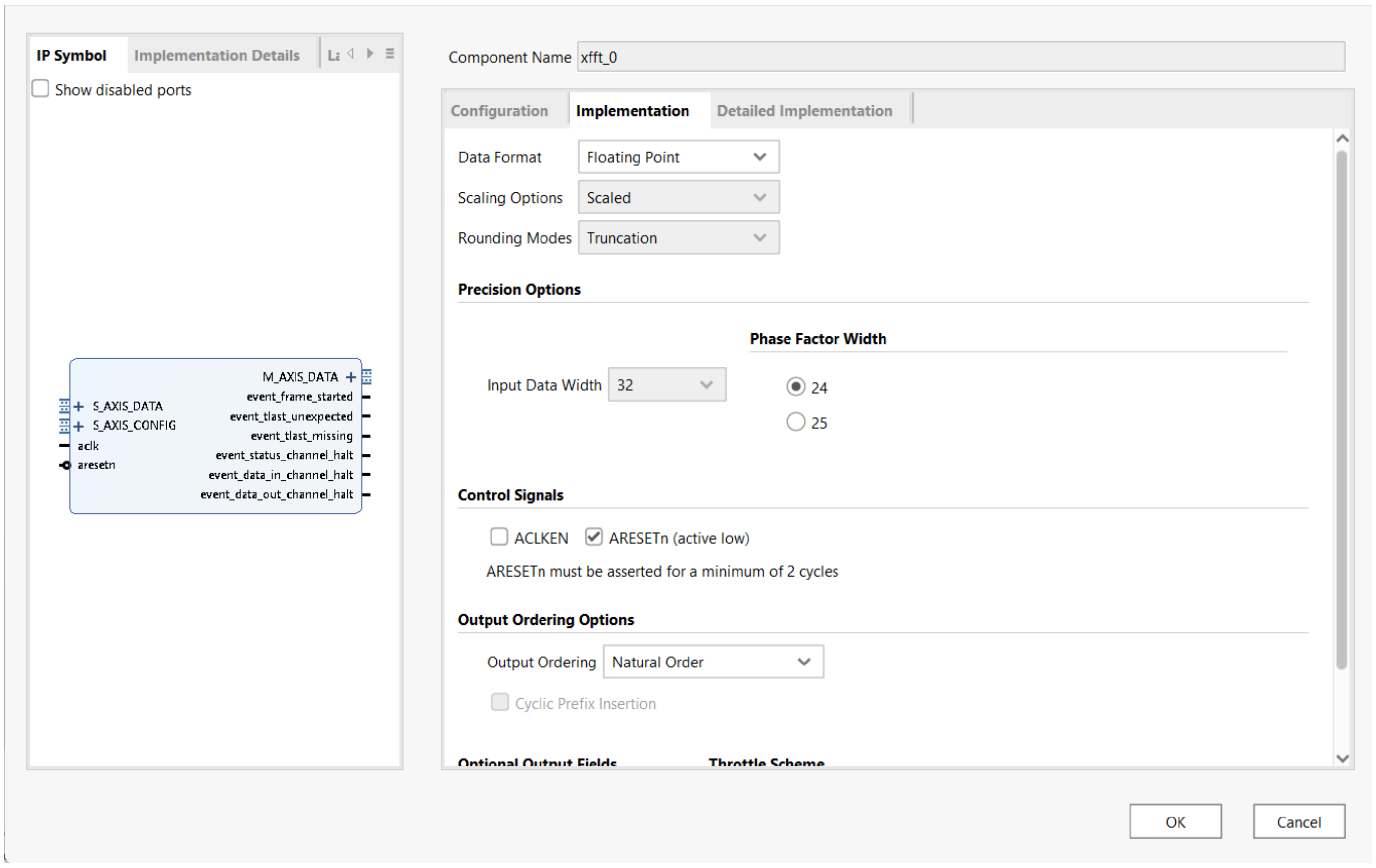
Architecture Choice

[✓ Automatically Select]
Pipelined, Streaming I/O
Radix-2, Burst I/O
Radix-2 Lite, Burst I/O

Run Time Configurable Transform Length

b. Make sure that the data format is floating point and the output ordering is set to Natural Order.
c. Check the ARESETn (active low)

Note: In the implementation details we can see that input data is of 64 bits with the upper 32 bits as imaginary and the lower 32 bits as real numbers.



Component Name xfft_0

Data Format: Floating Point
Scaling Options: Scaled
Rounding Modes: Truncation

Precision Options

Input Data Width: 32

Phase Factor Width

[✓ 24] 25

Control Signals

☐ ACLKEN ☑ ARESETn (active low)

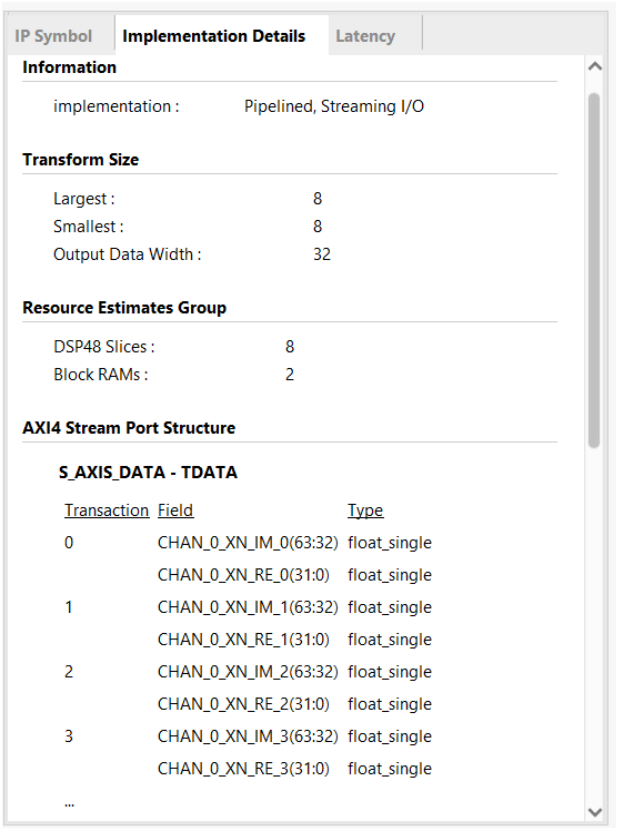
ARESETn must be asserted for a minimum of 2 cycles

Output Ordering Options

Output Ordering: Natural Order

☐ Cyclic Prefix Insertion

Optional Output Fields Throttle Scheme



Information

implementation :	Pipelined, Streaming I/O

Transform Size

Largest :	8
Smallest :	8
Output Data Width :	32

Resource Estimates Group

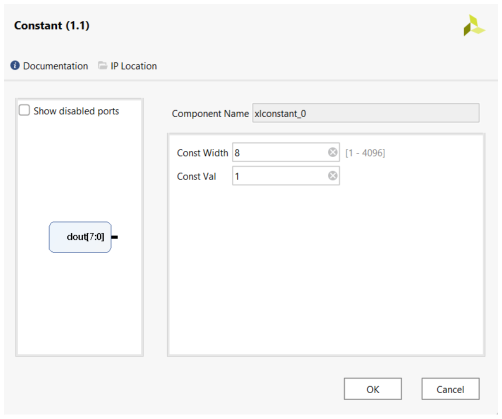
DSP48 Slices :	8
Block RAMs :	2

AXI4 Stream Port Structure

S_AXIS_DATA - TDATA

Transaction	Field	Type
0	CHAN_0_XN_IM_0(63:32)	float_single
	CHAN_0_XN_RE_0(31:0)	float_single
1	CHAN_0_XN_IM_1(63:32)	float_single
	CHAN_0_XN_RE_1(31:0)	float_single
2	CHAN_0_XN_IM_2(63:32)	float_single
	CHAN_0_XN_RE_2(31:0)	float_single
3	CHAN_0_XN_IM_3(63:32)	float_single
	CHAN_0_XN_RE_3(31:0)	float_single
...		

14. Now manually connect the M_AXIS_DATA of FFT IP to the S_AXIS_S2MM of AXI DMA IP and S_AXIS_DATA of FFT IP to the M_AXIS_MM2S of the AXI DMA IP.
    a. Connect the aclk of FFT IP to the s_axi_lite_aclk of AXI DMA IP
    b. Connect the aresetn pin of FFT IP to the axi_aresetn pin of AXI DMA IP.
    c. For performing FFT s_axis_config_tdata and s_axis_config_tvalid are to be explicitly given a high values, so we connect these ports with constants .
    d. S_axis_config_tdata[7:0] is connected to a constant of width 8 and value 1

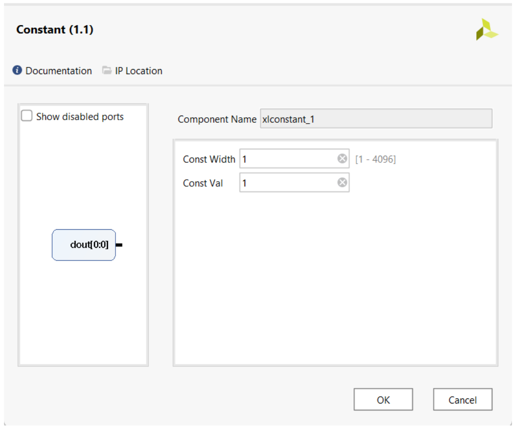

Constant (1.1)

Component Name xlconstant_0

Const Width 8 [1 - 4096]

Const Val 1

    e. S_axis_config_tvalid is connected to a constant of width 1 and value 1

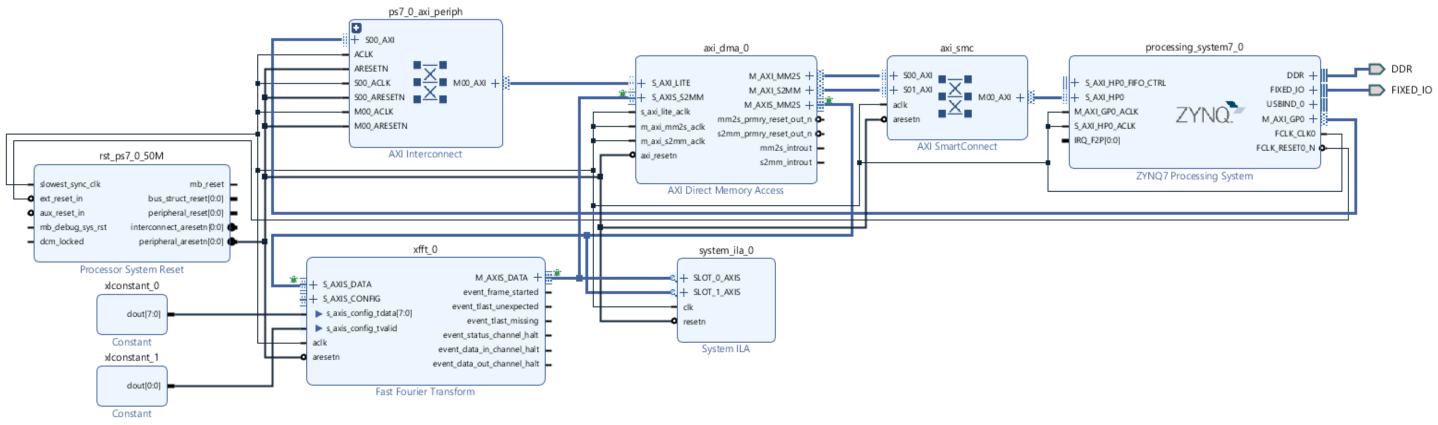

Constant (1.1)

Component Name xlconstant_1

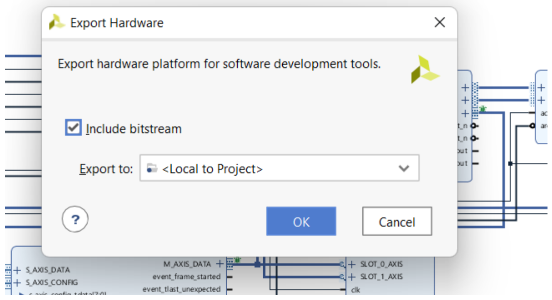
Const Width 1 [1 - 4096]

Const Val 1

15. Finally run the connection automation.
16. Debug the S_AXIS DATA and M_AXIS_DATA of FFT IP by right clicking and then run connection automation.
17. System ILA will be added automatically.
18. The block design is now complete.
19. Validate your design to check for any missing or incorrect connections.
20. Final Design should look like this.



21. Then in the sources tab, right-click on your block design file and select Create HDL Wrapper. Click on OK. This step will generate the corresponding Verilog files of your design.
22. Now generate the bitstream
23. Once the bitstream is generated, "Export Hardware" and while doing so, include the bitstream.

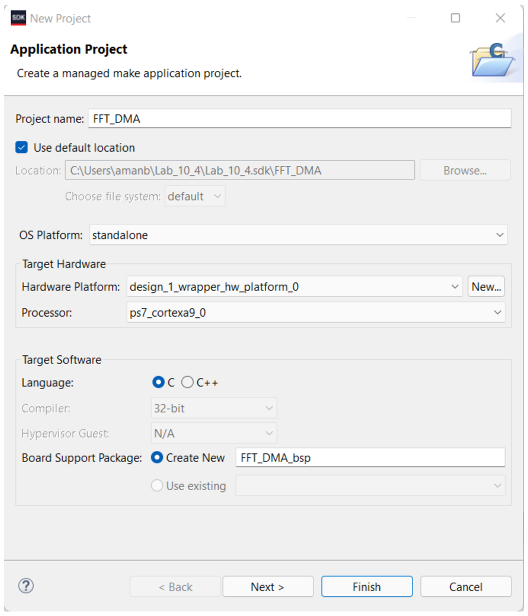

Export Hardware

Export hardware platform for software development tools.

☑ Include bitstream

Export to: <Local to Project>

24. Click on Launch SDK and wait for it to open.
25. Now create a New Application project.


New Project

Application Project

Create a managed make application project.

Project name: FFT_DMA

[✓] Use default location

Location: C:\Users\amanb\Lab_10_4\Lab_10_4.sdk\FFT_DMA

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++

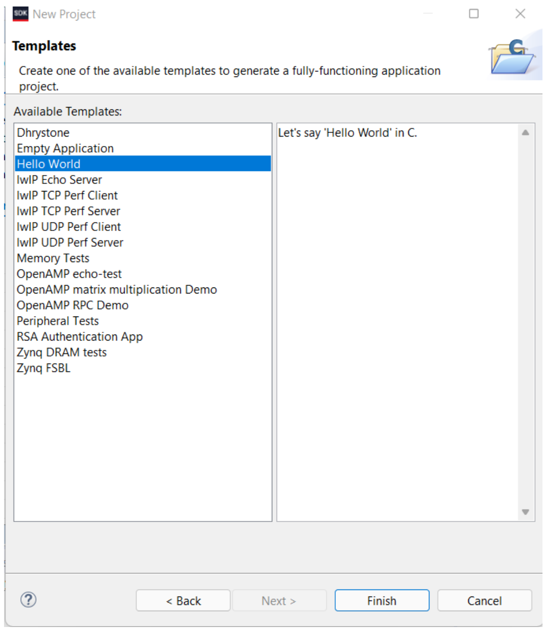
Compiler: 32-bit

Hypervisor Guest: N/A

Board Support Package: [✓ Create New] FFT_DMA_bsp

Use existing

26. Click Next and select the Hello World template from the available template palette.


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

Let's say 'Hello World' in C.

27. Open the helloworld.c file from FFT_DMA/src folder
28. Add the following code

```
#include <stdio.h>
#include <stdbool.h>
#include "xaxidma.h"
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include <stdlib.h>
```

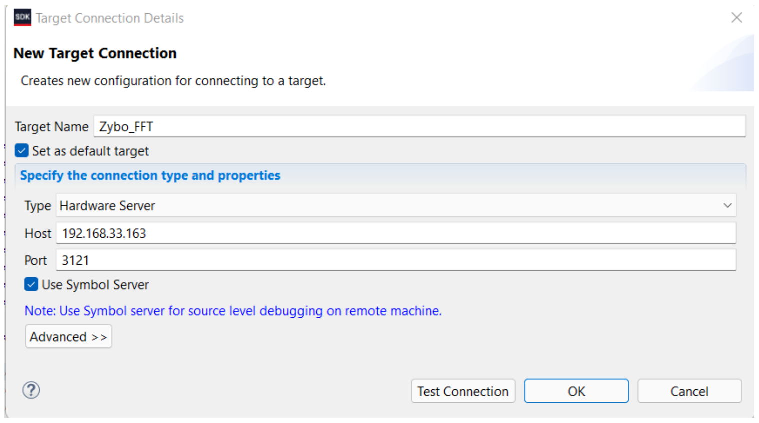
```c
#include <complex.h>
#include <time.h>
#include <xtime_l.h>
#include <math.h>

#define N 8

//vector which contains the positions of data shuffle
const int rev8_in[N] = {0, 4, 2, 6, 1, 5, 3, 7};

const float complex W[N/2] = {1-0*I, 0.7071067811865476-0.7071067811865475*I,
0.0-1*I,  -0.7071067811865475-0.7071067811865476*I};

void bitreverse(int rev8[N] , float complex dataIn[N], float complex dataOut[N])
{
    bit_reversal: for (int i = 0 ; i < N ; i++)
    {
        dataOut[i] = dataIn[rev8[i]];
    }
}

void FFT_stages(float complex FFT_input[N], float complex FFT_output[N])
{
    float complex temp1[N], temp2[N];
    stage1: for (int i = 0 ; i < N ; i=i+2)
    {
        temp1[i] = FFT_input[i] + FFT_input[i+1];
        temp1[i+1] = FFT_input[i] - FFT_input[i+1];
    }

    stage2: for (int i = 0 ; i < N ; i=i+4)
    {
        for (int j = 0; j < 2; ++j)
        {
                temp2[i+j] = temp1[i+j] + W[2*j]*temp1[i+j+2];
                temp2[i+2+j] = temp1[i+j] - W[2*j]*temp1[i+j+2];
        }

    }

    stage3: for (int i = 0 ; i < N/2 ; i++)
    {
        FFT_output[i] = temp2[i] + W[i]*temp2[i+4];
        FFT_output[i+4] = temp2[i] - W[i]*temp2[i+4];
    }

}
```

```
XAxiDma AxiDma;

int init_DMA() //this function initializes the DMA instance with required
configurations
{
    XAxiDma_Config *CfgPtr;

    int status;

    CfgPtr = XAxiDma_LookupConfig(XPAR_AXI_DMA_0_DEVICE_ID); //extracts the
configuration settings of DMA
    //the XPAR_AXI_DMA_0_DEVICE_ID is defined in xparameters.h file

    if (!CfgPtr)
    {
        xil_printf("No config found for %d\r\n", XPAR_AXI_DMA_0_DEVICE_ID);
        return XST_FAILURE;
    }

    status = XAxiDma_CfgInitialize(&AxiDma, CfgPtr); //initializes the DMA instance
with above settings

    if (status != XST_SUCCESS)
    {
        xil_printf("DMA Initialization Failed. Return Status: %d\r\n", status);
        return XST_FAILURE;
    }

    if(XAxiDma_HasSg(&AxiDma)) //check whether DMA is in scatter gather mode. If
yes, return error
    {
        xil_printf("Device configuration as SG mode\r\n");
        return XST_FAILURE;
    }

    return XST_SUCCESS;

    }

u32 checkIdle(u32 baseAddress, u32 offset) //this function is defined to check
whether DMA is busy or not
    { //offset = 0x4 to check for DMA to FIFO transaction channel

    //offset = 0x34 to check for FIFO to DMA transaction channel
        u32 status;
        status = (XAxiDma_ReadReg(baseAddress, offset)) &
XAXIDMA_IDLE_MASK;//XAXIDMA_IDLE_MASK;

        return status;
    }

int main()
{

    // to store the time at which certain processes starts and ends
    XTime tprocessorStart , tprocessorend;
    XTime tfpgaStart , tfpgaend;

      const float complex FFT_input[N] =
{11+23*I,32+10*I,91+94*I,15+69*I,47+96*I,44+12*I,96+17*I,49+58*I};// input to the
FFT in PS and PL

      float complex FFT_output_PS[N];
      float complex FFT_rev[N];

      //-------------------- PS PART -------------------------------//
      XTime_GetTime(&tprocessorStart); // getting the time before FFT is Done in
PS
      bitreverse(rev8_in , FFT_input, FFT_rev);
      FFT_stages(FFT_rev, FFT_output_PS);
      XTime_GetTime(&tprocessorend); // getting the time after FFT is done in PS


    xil_printf("\r\n***Entering main function***\r\n");


    int status_dma = init_DMA(); //initializing the DMA

    if (status_dma != XST_SUCCESS)
    {
        xil_printf("Couldn't initialize DMA\r\n");
        return XST_FAILURE;
    }


    float complex RX_PNTR[N]; // to store the output from the PL

    bool err_flag = false; //error flag to check for mismatch between transmitted
and received data
```

```
    Xil_DCacheFlushRange((UINTPTR)FFT_input, (sizeof(float complex )*N)); //cache
flush to write back the data in the DDR memory
    Xil_DCacheFlushRange((UINTPTR)RX_PNTR, (sizeof(float complex)*N));

    //status = 2 indicates idle and status = 0 indicates idle
    xil_printf("\nDMA status before transfer\r\nDMA to Device: %d, Device to
DMA:%d\r\n", checkIdle(XPAR_AXI_DMA_0_BASEADDR, 0x4),
checkIdle(XPAR_AXI_DMA_0_BASEADDR,0x34));
    xil_printf("\rStarting Data Transfer------------->>>>>>>>>\r\n");

    int status_transfer;

    XTime_GetTime(&tfpgaStart); // getting the time before FFT is Done in PL

    status_transfer = XAxiDma_SimpleTransfer(&AxiDma, (UINTPTR)RX_PNTR,
(sizeof(float complex)*N) , XAXIDMA_DEVICE_TO_DMA);
    //transferring data from DDR to FIFO.

    if (status_transfer != XST_SUCCESS)
    {
        xil_printf("Writing data to FFT_IP via DMA failed\r\n");
    }

    xil_printf("DMA status between transfer\nDMA to Device status: %d, Device to
DMA status: %d\r\n", checkIdle(XPAR_AXI_DMA_0_BASEADDR,
0x4),checkIdle(XPAR_AXI_DMA_0_BASEADDR, 0x34));


    status_transfer = XAxiDma_SimpleTransfer(&AxiDma, (UINTPTR)FFT_input,
(sizeof(float complex)*N), XAXIDMA_DMA_TO_DEVICE);
    //receiving data from FIFO in DDR.
    if (status_transfer != XST_SUCCESS)
    {
        xil_printf("Reading data from FFT via DMA failed\r\n");
    }
        xil_printf("DMA status after transfer\nDMA to Device status: %d, Device to
DMA status: %d\r\n", checkIdle(XPAR_AXI_DMA_0_BASEADDR,
0x4),checkIdle(XPAR_AXI_DMA_0_BASEADDR, 0x34));

    XTime_GetTime(&tfpgaend); // getting the time before FFT is Done in PL

    int status = checkIdle(XPAR_AXI_DMA_0_BASEADDR, 0x4); //DMA to device

    while(status != 2)
    {
        status = checkIdle(XPAR_AXI_DMA_0_BASEADDR, 0x4);
        //xil_printf("status value : %d \n" , status);
```

```c
    }
        status = checkIdle(XPAR_AXI_DMA_0_BASEADDR, 0x34); //Device to DMA

    while(status != 2)
    {
        status = checkIdle(XPAR_AXI_DMA_0_BASEADDR, 0x34);

    }

    xil_printf("DMA status after transfer\nDMA to Device status: %d, Device to DMA
status: %d\r\n", checkIdle(XPAR_AXI_DMA_0_BASEADDR,
0x4),checkIdle(XPAR_AXI_DMA_0_BASEADDR, 0x34));
    xil_printf("\nComparing software FFT outputs and hardware FFT outputs via DMA
\r\n");


    int j = 0;
    printf("size of float complex : %d \n" , sizeof(float complex));
    for (j = 0 ; j < N ; j++)
    {
        printf("PS Output : %f + I%f , PL output : %f + I%f \n " ,
crealf(FFT_output_PS[j]), cimagf(FFT_output_PS[j]) , crealf(RX_PNTR[j]) ,
cimagf(RX_PNTR[j]));
        //printing the output of PS and PL

        float diff1 = abs(crealf(FFT_output_PS[j]) - crealf(RX_PNTR[j]));
        float diff2 = abs(cimagf(FFT_output_PS[j]) - cimagf(RX_PNTR[j]));

        if ( diff1 >= 0.0001 && diff2 >= 0.0001)
        {

              err_flag = true; //error flag asserted for any mismatch
              break;
        }
    }


    if (err_flag)
        xil_printf("Data Mismatch found at %d. Transmitted Data: %d. Received Data:
%d\r\n", j , FFT_output_PS[j], RX_PNTR[j]);
    else
        xil_printf("\nDMA ran successfully!! :)");

    xil_printf("\n-----------TIME COMPARISION--------------------\n");


    // here we are calculating the time it took for PS and PL to execute a data
    float time_processor = 0;
    time_processor = (float)1.0 * (tprocessorend - tprocessorStart) /
(COUNTS_PER_SECOND/1000000);
    xil_printf("time for PS : %d\n" , time_processor);

    float time_fpga = 0;
    time_fpga = (float)1.0 * (tfpgaend - tfpgaStart) / (COUNTS_PER_SECOND/1000000);
    xil_printf("time for PL : %d\n" , time_fpga);

    return XST_SUCCESS;

}
```

29. Add the Hardware server
    a. Add a new target connection


Target Connection Details

New Target Connection

Creates new configuration for connecting to a target.

Target Name: Zybo_FFT

☑ Set as default target

Specify the connection type and properties

Type: Hardware Server

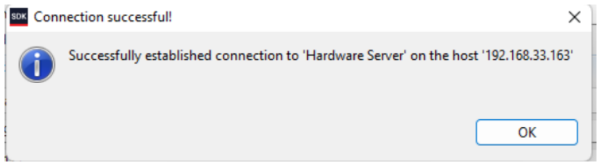
Host: 192.168.33.163

Port: 3121

☑ Use Symbol Server

Note: Use Symbol server for source level debugging on remote machine.

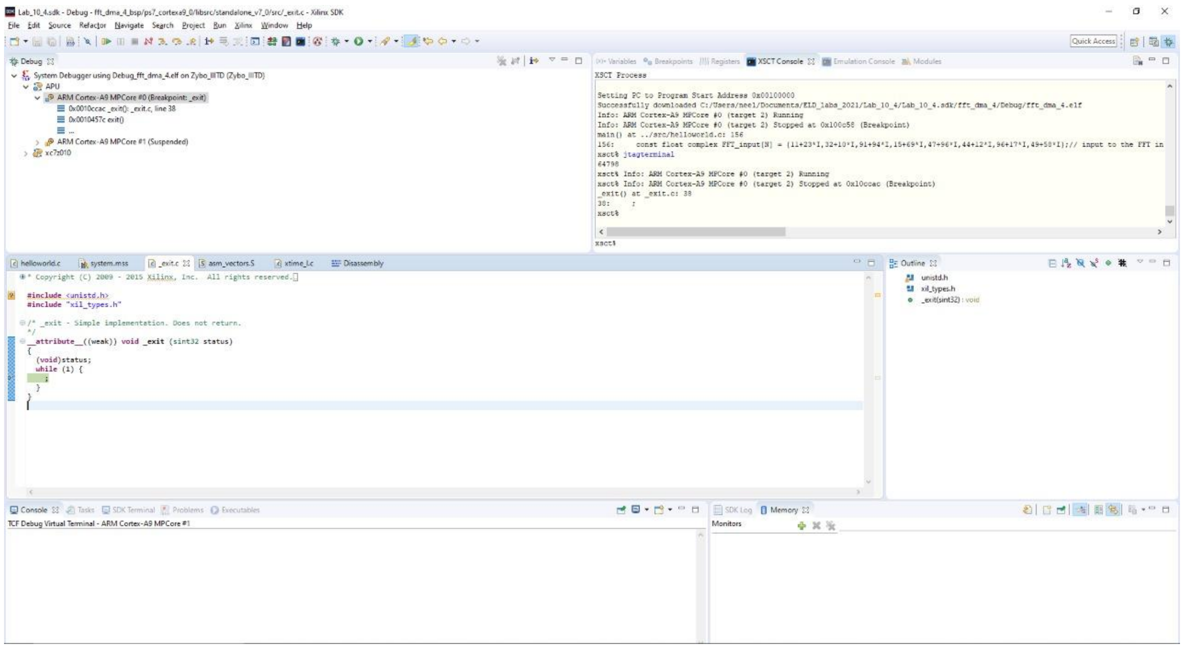
    b. Test the connection.


Connection successful!

Successfully established connection to 'Hardware Server' on the host '192.168.33.163'

30. Now from the system.mss file. Select the Modify BSP's settings
    a. Change the STDIN and STDOUT to ps7_coresight_comp_0 (to enable JTAG interface instead of default UART). Click on Ok.
    b. Click ok.
31. Right click on the project FFT_DMA , go to Debug As and then select Debug Configurations.
32. Double click the Xilinx C/C++ Application (System Debugger) to define the debug configuration for the current run. Check reset the entire system and program FPGA. Leave other settings unchanged.
33. Once the system debugger is launched, type jtagterminal in XSCT Console. This will open the terminal for displaying output messages.

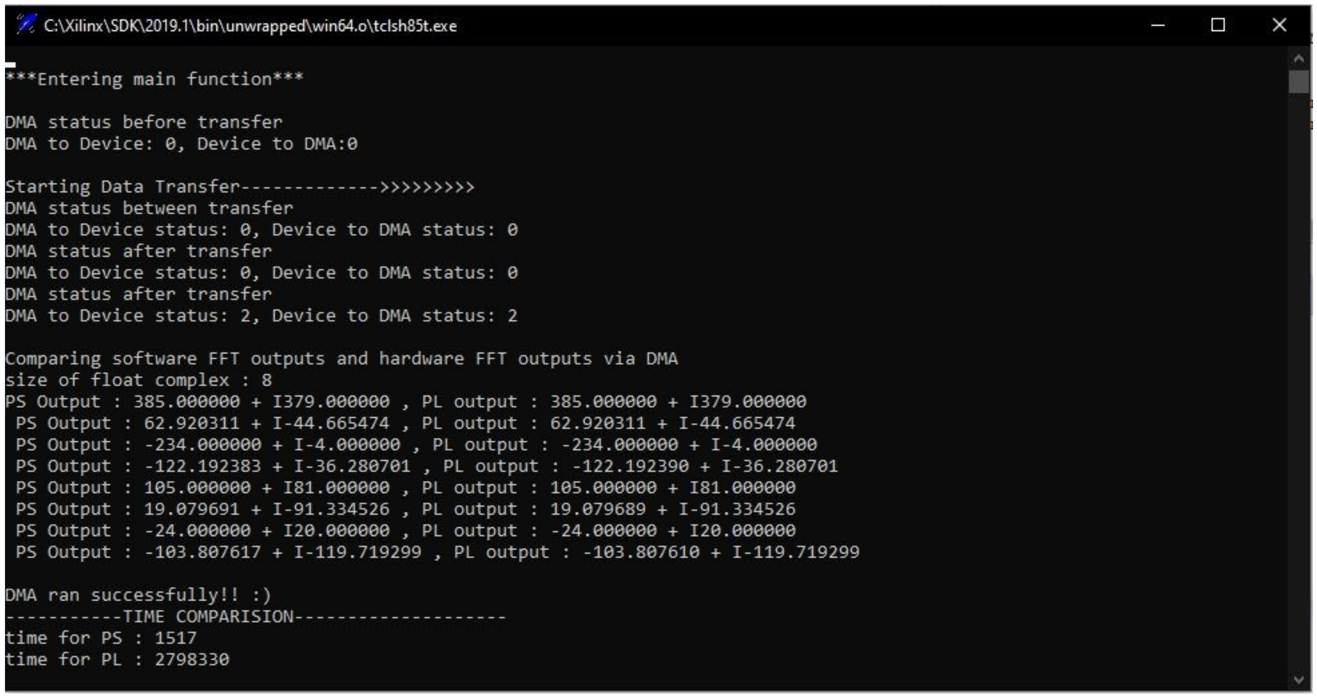
34. Click on the resume button.



```
XSCT Process

Setting PC to Program Start Address 0x00100000
Successfully downloaded C:/Users/neel/Documents/ELD_labs_2021/Lab_10_4/Lab_10_4.sdk/fft_dma_4/Debug/fft_dma_4.elf
Info: ARM Cortex-A9 MPCore #0 (target 2) Running
Info: ARM Cortex-A9 MPCore #0 (target 2) Stopped at 0x100c58 (Breakpoint)
main() at ../src/helloworld.c: 156
156:     const float complex FFT_input[N] = {11+23*I,32+10*I,91+94*I,15+69*I,47+96*I,44+12*I,96+17*I,49+58*I};// input to the FFT in
xsct% jtagterminal
64798
xsct% Info: ARM Cortex-A9 MPCore #0 (target 2) Running
xsct% Info: ARM Cortex-A9 MPCore #0 (target 2) Stopped at 0x10ccac (Breakpoint)
_exit() at _exit.c: 38
38:     ;
xsct%
```

```c
/* Copyright (C) 2009 - 2015 Xilinx, Inc.  All rights reserved.

#include <unistd.h>
#include "xil_types.h"

/* _exit - Simple implementation. Does not return.
*/
__attribute__((weak)) void _exit (sint32 status)
{
  (void)status;
  while (1) {
	;
  }
}
```

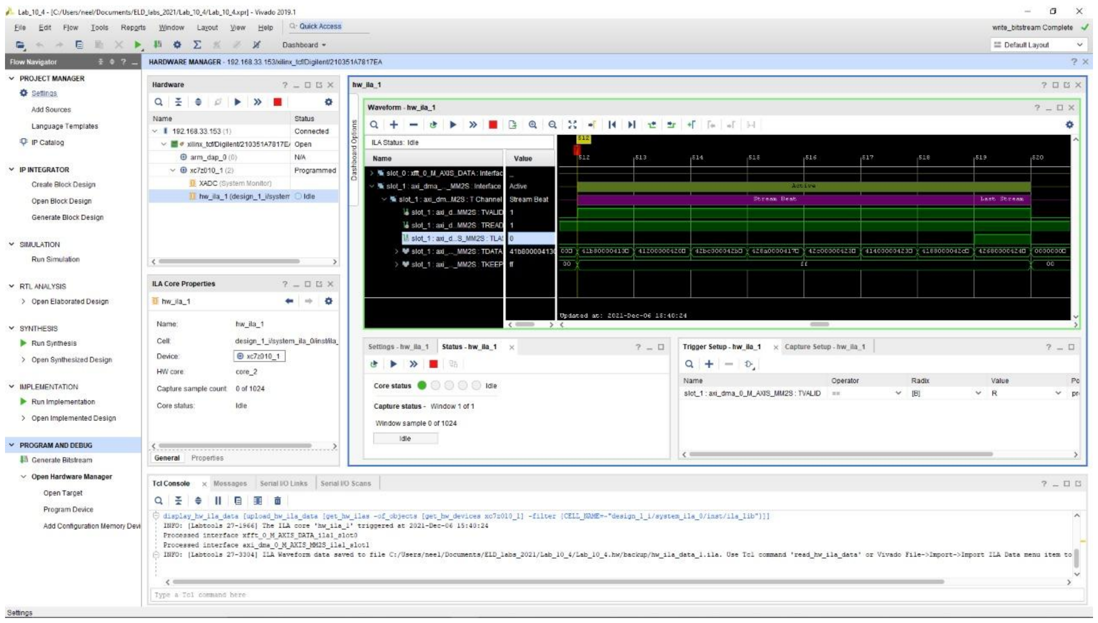
35. Terminal window will have these outputs



```
C:\Xilinx\SDK\2019.1\bin\unwrapped\win64.o\tclsh85t.exe

***Entering main function***

DMA status before transfer
DMA to Device: 0, Device to DMA:0

Starting Data Transfer------------->>>>>>>>>
DMA status between transfer
DMA to Device status: 0, Device to DMA status: 0
DMA status after transfer
DMA to Device status: 0, Device to DMA status: 0
DMA status after transfer
DMA to Device status: 2, Device to DMA status: 2

Comparing software FFT outputs and hardware FFT outputs via DMA
size of float complex : 8
PS Output : 385.000000 + I379.000000 , PL output : 385.000000 + I379.000000
 PS Output : 62.920311 + I-44.665474 , PL output : 62.920311 + I-44.665474
 PS Output : -234.000000 + I-4.000000 , PL output : -234.000000 + I-4.000000
 PS Output : -122.192383 + I-36.280701 , PL output : -122.192390 + I-36.280701
 PS Output : 105.000000 + I81.000000 , PL output : 105.000000 + I81.000000
 PS Output : 19.079691 + I-91.334526 , PL output : 19.079689 + I-91.334526
 PS Output : -24.000000 + I20.000000 , PL output : -24.000000 + I20.000000
 PS Output : -103.807617 + I-119.719299 , PL output : -103.807610 + I-119.719299

DMA ran successfully!! :)
-----------TIME COMPARISION--------------------
time for PS : 1517
time for PL : 2798330
```

Now to observe the AXI transactions, we will utilize the system ILA IP which was included when we added debug in step 16-17.

36. Now go back to Vivado and open the hardware manager.
37. Connect to the same Host IP as you did in SDK
38. Program the device
39. Now, the waveform window will open up
40. Add triggers for slot 1 and slot 2 for TVALID.


HARDWARE MANAGER - 192.168.33.153/xilinx_tcf/Digilent/210351A7817EA
Waveform - hw_ila_1
Trigger Setup - hw_ila_1
slot_1 : axi_dma_0_M_AXIS_MM2S : TVALID
Tcl Console

```
display_hw_ila_data [upload_hw_ila_data [get_hw_ilas -of_objects [get_hw_devices xc7z010_1] -filter {CELL_NAME=~"design_1_i/system_ila_0/inst/ila_lib"}]]
INFO: [Labtools 27-1966] The ILA core 'hw_ila_1' triggered at 2021-Dec-06 15:40:24
Processed interface xfft_0_M_AXIS_DATA_ila1_slot0
Processed interface axi_dma_0_M_AXIS_MM2S_ila1_slot1
INFO: [Labtools 27-3304] ILA Waveform data saved to file C:/Users/neel/Documents/ELD_labs_2021/Lab_10_4/Lab_10_4.hw/backup/hw_ila_data_1.ila. Use Tcl command 'read_hw_ila_data' or Vivado File->Import->Import ILA Data menu item to
```