INDRAPRASTHA INSTITUTE of INFORMATION TECHNOLOGY DELHI

Department
of
Electronics & Communication Engineering

Embedded Logic Design(ECE270)

Dr. Sumit J Darak

Lab_12

Mohammad Shariq

2020220

17-12-2021

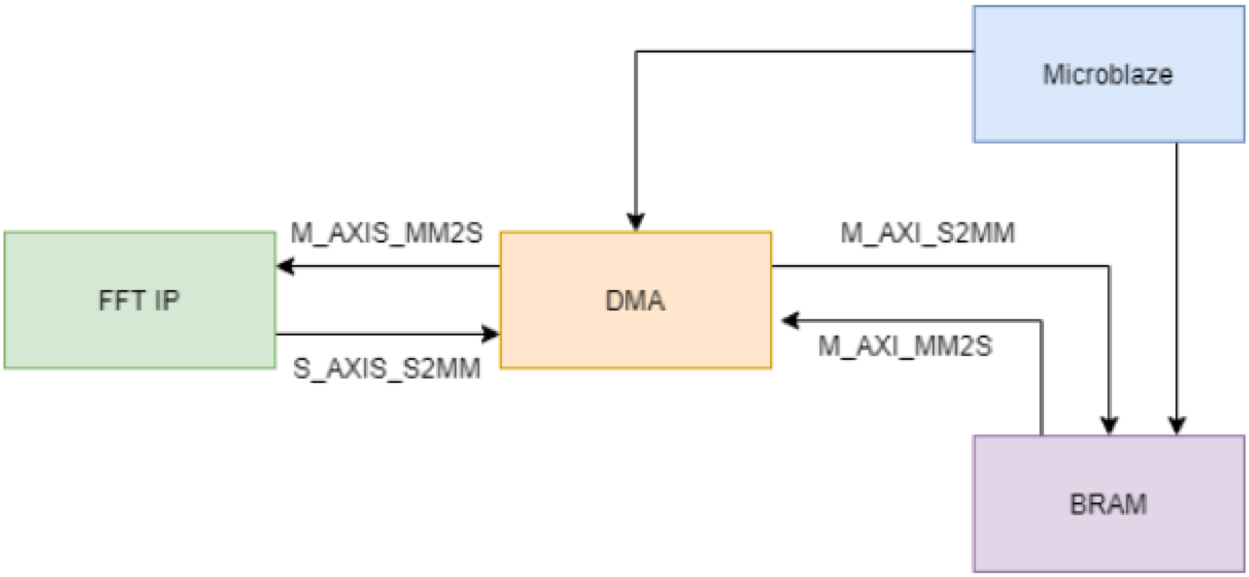
OBJECTIVE:

- In this lab, we will be using a Microblaze processor. The processor will store some data inside the BRAM and configure the DMA to perform data transfer between BRAM and FFT IP. The DMA will read the data from the BRAM via the port M_AXI_MM2S and send it to the FFT IP via the port M_AXIS_MM2S.

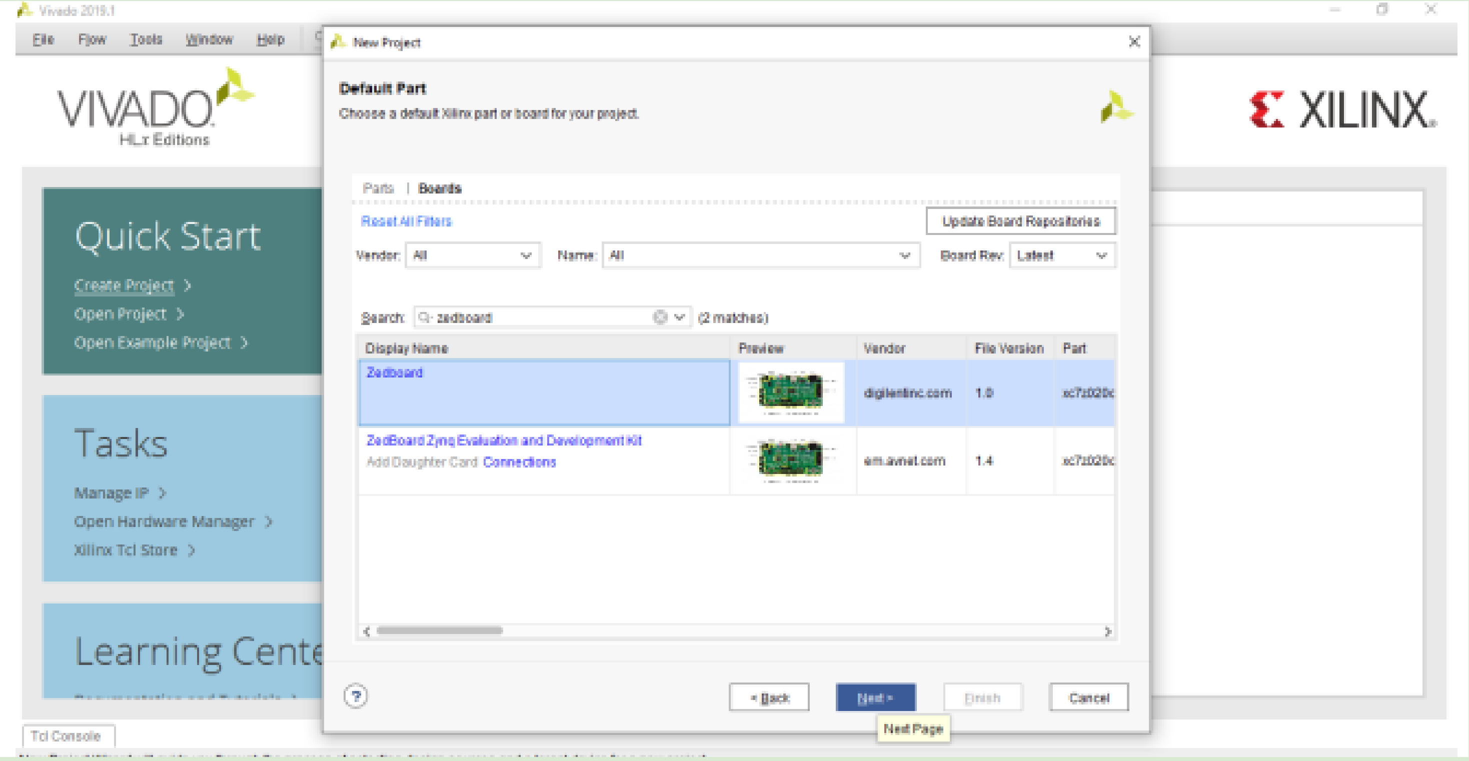
- The FFT IP after processing initiates the transfer, the DMA receives the processed data from FFT IP via the S_AXIS_S2MM port. The DMA writes the processed data into the BRAM via the port M_AXI_S2MM.

Theory:

Block Diagram



Observations:



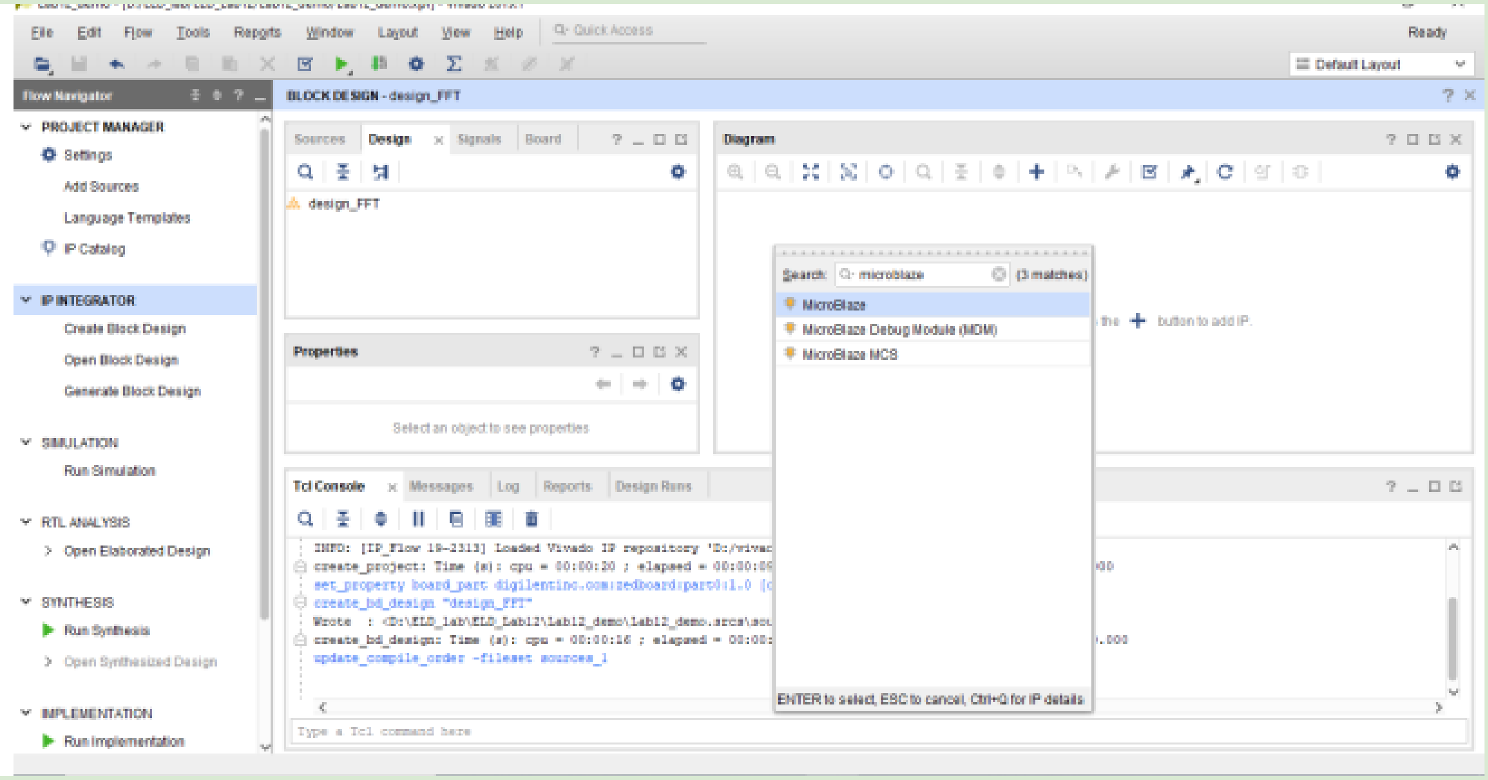
Default Part

Choose a default Xilinx part or board for your project.

Vendor: All   Name: All   Board Rev: Latest

Search: zedboard (2 matches)

Display Name	Preview	Vendor	File Version	Part
Zedboard		digilentinc.com	1.0	xc7z020c
ZedBoard Zynq Evaluation and Development Kit Add Daughter Card Connections		em.avnet.com	1.4	xc7z020c



Search: microblaze (3 matches)

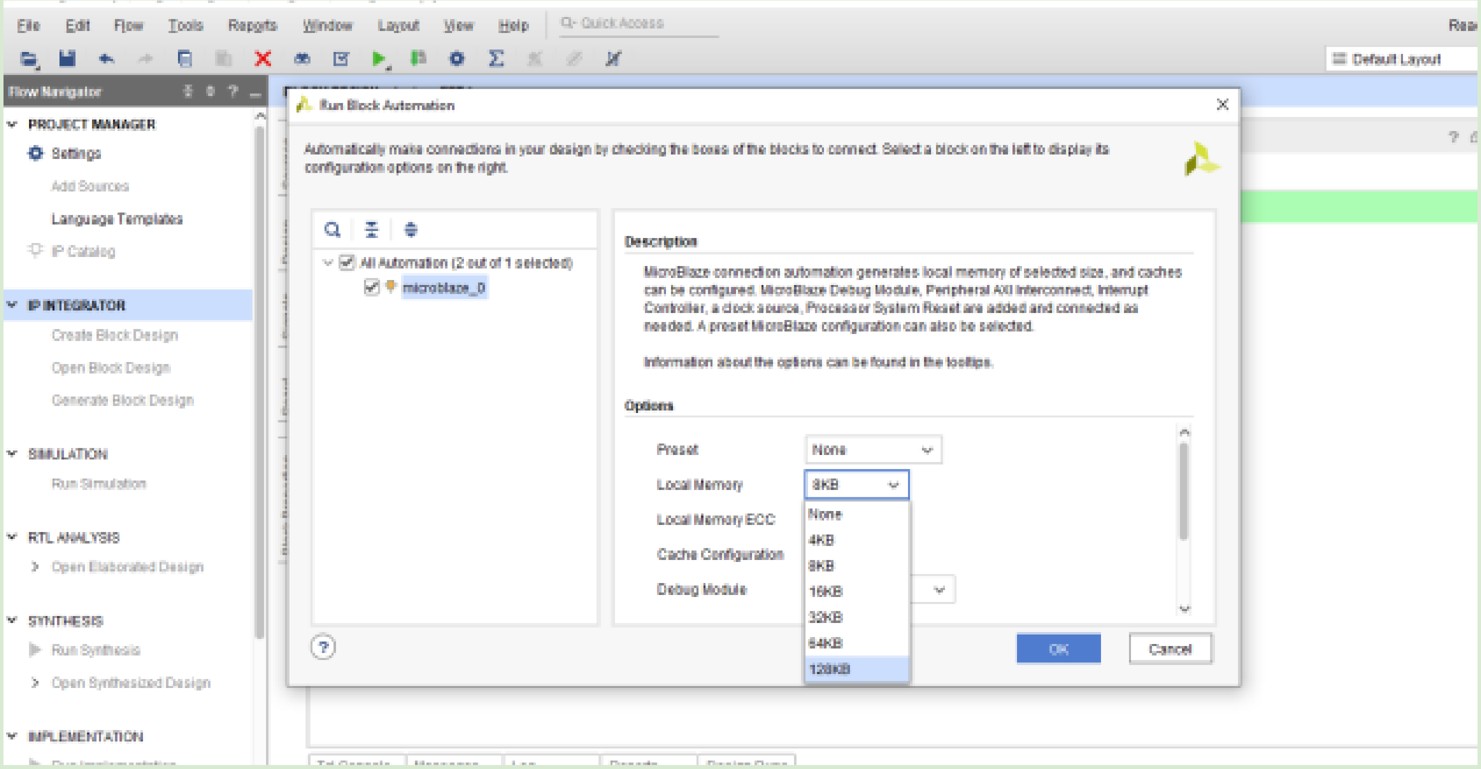
- MicroBlaze
- MicroBlaze Debug Module (MDM)
- MicroBlaze MCS

ENTER to select, ESC to cancel, Ctrl+Q for IP details

```
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'D:/vivac
create_project: Time (s): cpu = 00:00:20 ; elapsed = 00:00:0
set_property board_part digilentinc.com:zedboard:part0:1.0 [c
create_bd_design "design_FFT"
Wrote  : <D:\ELD_lab\ELD_Lab12\lab12_demo\lab12_demo.srcs\sou
create_bd_design: Time (s): cpu = 00:00:16 ; elapsed = 00:00:
update_compile_order -fileset sources_1
```

Type a Tcl command here

Click on + sign on the block design window to add new IP. Search for Microblaze and add it to the block design.



Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

☑ All Automation (2 out of 1 selected)

☑ microblaze_0

Description

MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI Interconnect, Interrupt Controller, a clock source, Processor System Reset are added and connected as needed. A preset MicroBlaze configuration can also be selected.

Information about the options can be found in the tooltips.

Options

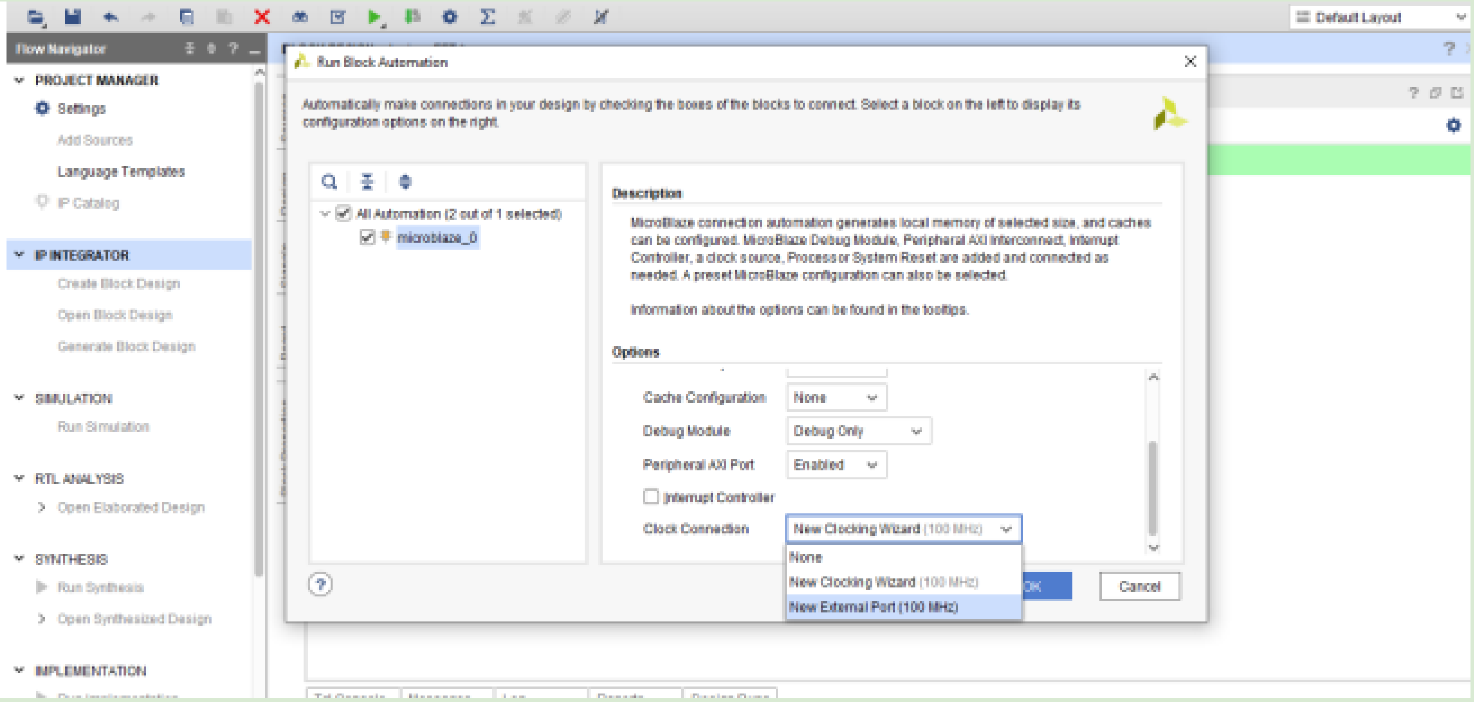
Preset: None

Local Memory: 8KB

Local Memory ECC

Cache Configuration

Debug Module



Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

☑ All Automation (2 out of 1 selected)

☑ microblaze_0

Description

MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI Interconnect, Interrupt Controller, a clock source, Processor System Reset are added and connected as needed. A preset MicroBlaze configuration can also be selected.

Information about the options can be found in the tooltips.

Options

Cache Configuration: None

Debug Module: Debug Only

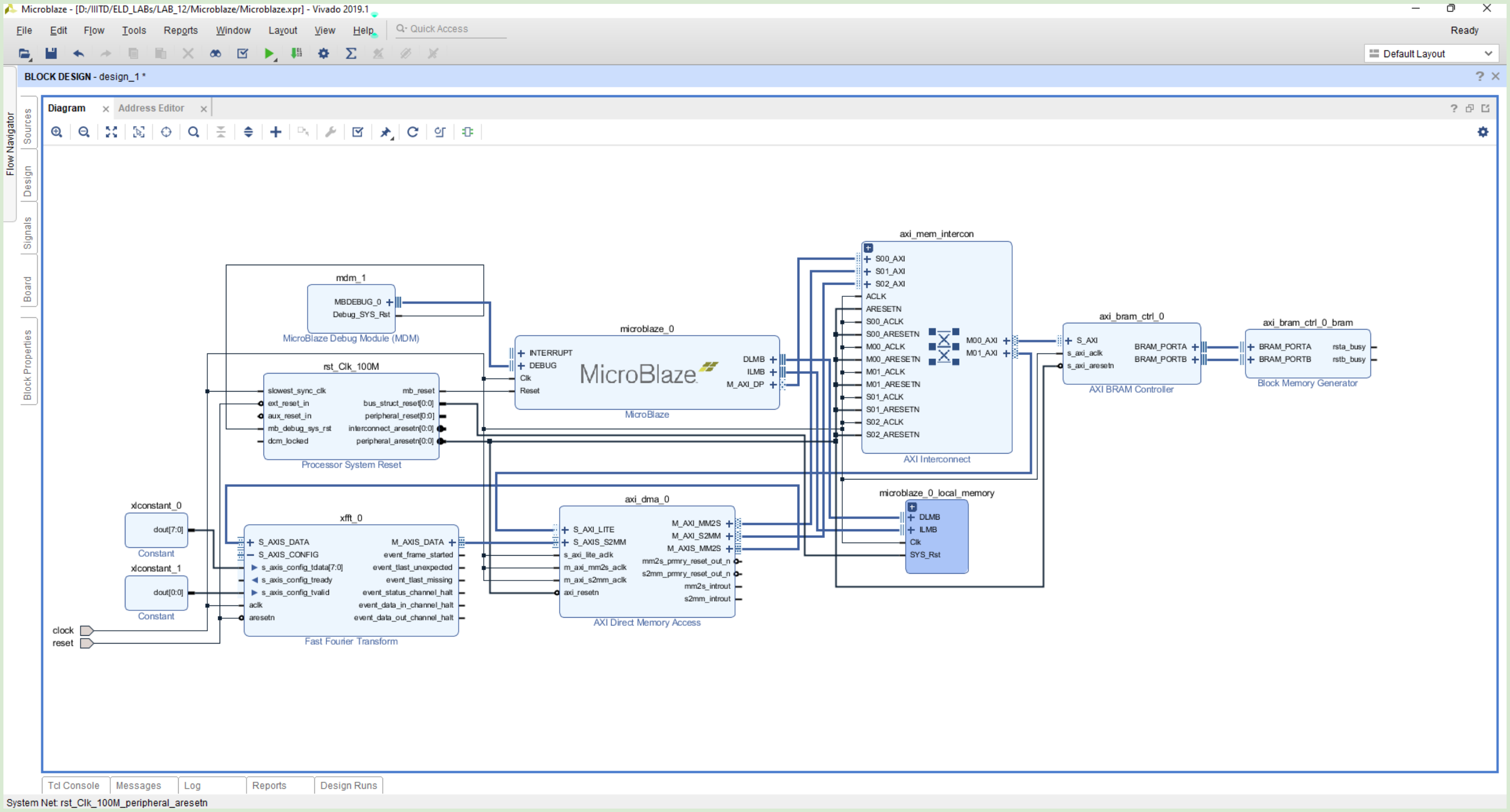
Peripheral AXI Port: Enabled

□ Interrupt Controller

Clock Connection: New Clocking Wizard (100 MHz)

After adding Microblaze to the block design, click on Run Block Automation. Choose the Local memory to be 128KB and the Clock Connection as New External Port (100MHz).Now click on OK.

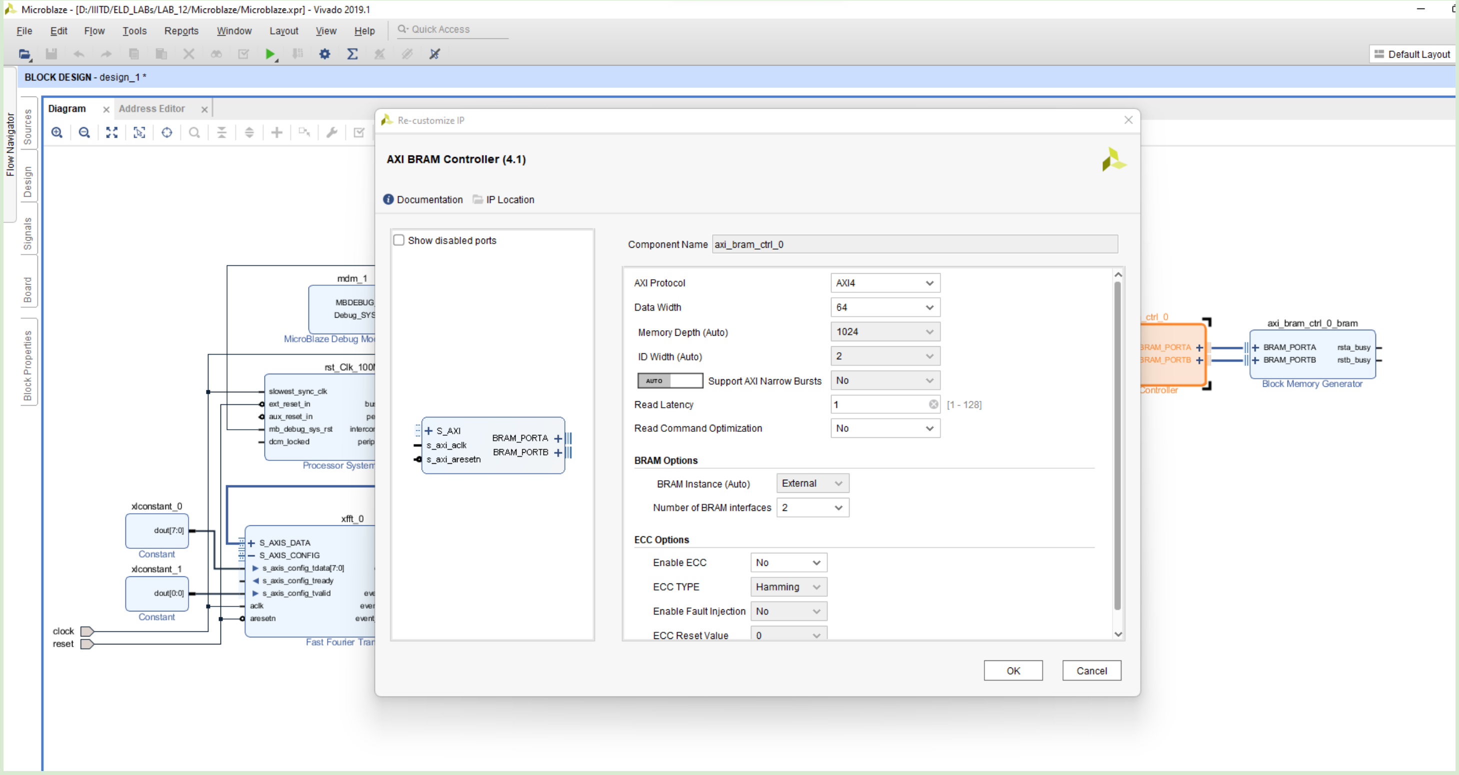
MicroBlaze_processing_System_Block_Design :



- *Select the Clk port and change its name to Clock (Same name should be used in the test bench) in External Port Properties.*

- *In the Processor System Reset, right click on the port ext_reset_in and select Make External. This will create an external port for the reset.*

- *In the Processor System Reset, right click on the port ext_reset_in and select Make External. This will create an external port for the reset.*



Re-customize IP

AXI BRAM Controller (4.1)

Documentation IP Location

□ Show disabled ports

Component Name axi_bram_ctrl_0

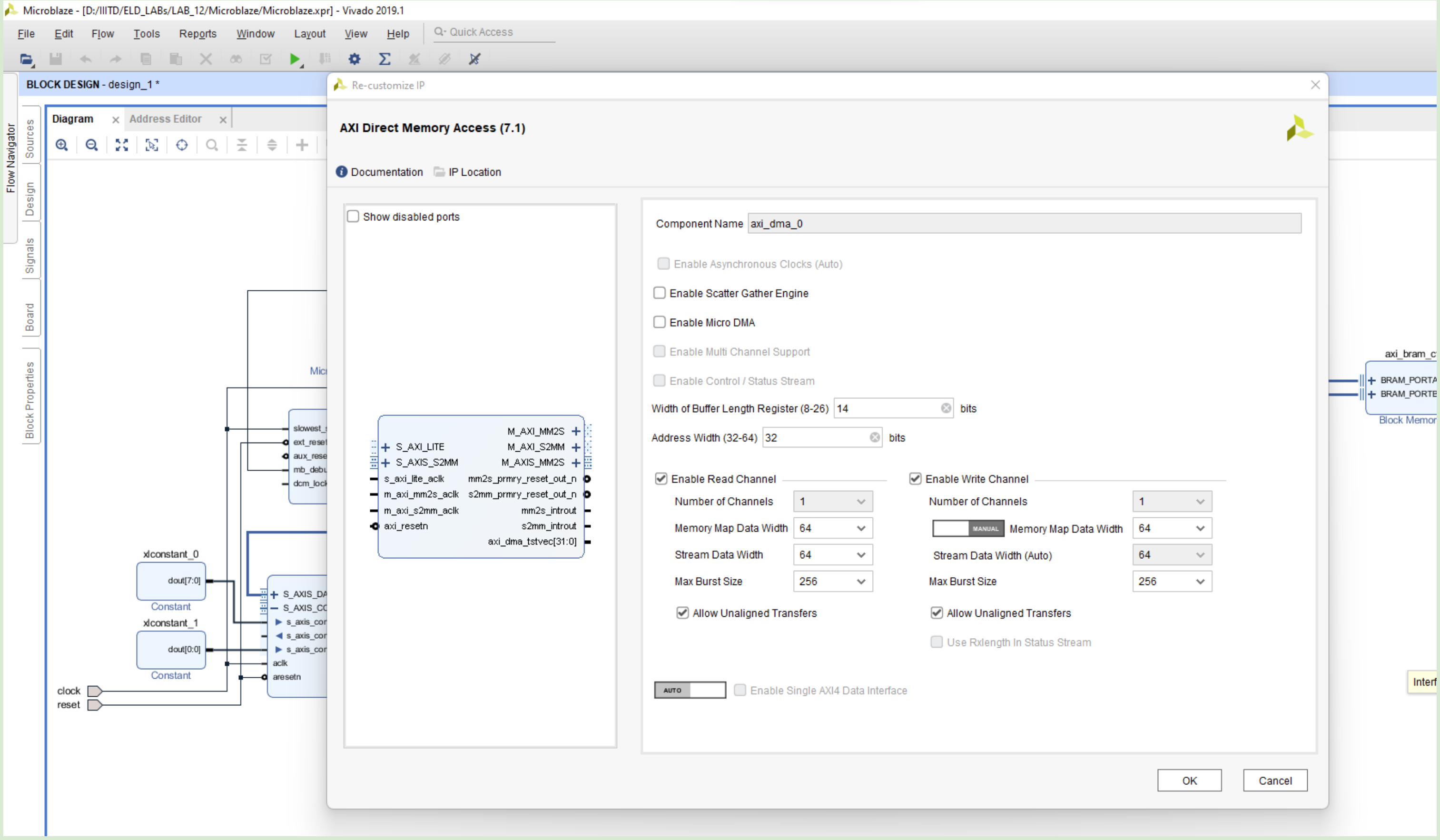
AXI Protocol AXI4
Data Width 64
Memory Depth (Auto) 1024
ID Width (Auto) 2
AUTO Support AXI Narrow Bursts No
Read Latency 1 [1 - 128]
Read Command Optimization No

BRAM Options
BRAM Instance (Auto) External
Number of BRAM interfaces 2

ECC Options
Enable ECC No
ECC TYPE Hamming
Enable Fault Injection No
ECC Reset Value 0

OK Cancel

- *Add AXI BRAM Controller to the block design and double click on it to customize it. Choose the Data Width to be 64 bits and click on OK.*



Re-customize IP

AXI Direct Memory Access (7.1)

Documentation IP Location

□ Show disabled ports

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
Number of Channels 1
Memory Map Data Width 64
Stream Data Width 64
Max Burst Size 256
☑ Allow Unaligned Transfers

☑ Enable Write Channel
Number of Channels 1
MANUAL Memory Map Data Width 64
Stream Data Width (Auto) 64
Max Burst Size 256
☑ Allow Unaligned Transfers
□ Use Rxlength In Status Stream

AUTO □ Enable Single AXI4 Data Interface

OK Cancel

- Add the AXI Direct Memory Access to the block design and double click on it to re-customize it. Make the following changes.
  a) Deselect the Enable Scatter Gather Engine. This special mode of DMA is used to read from non-contiguous memory locations.

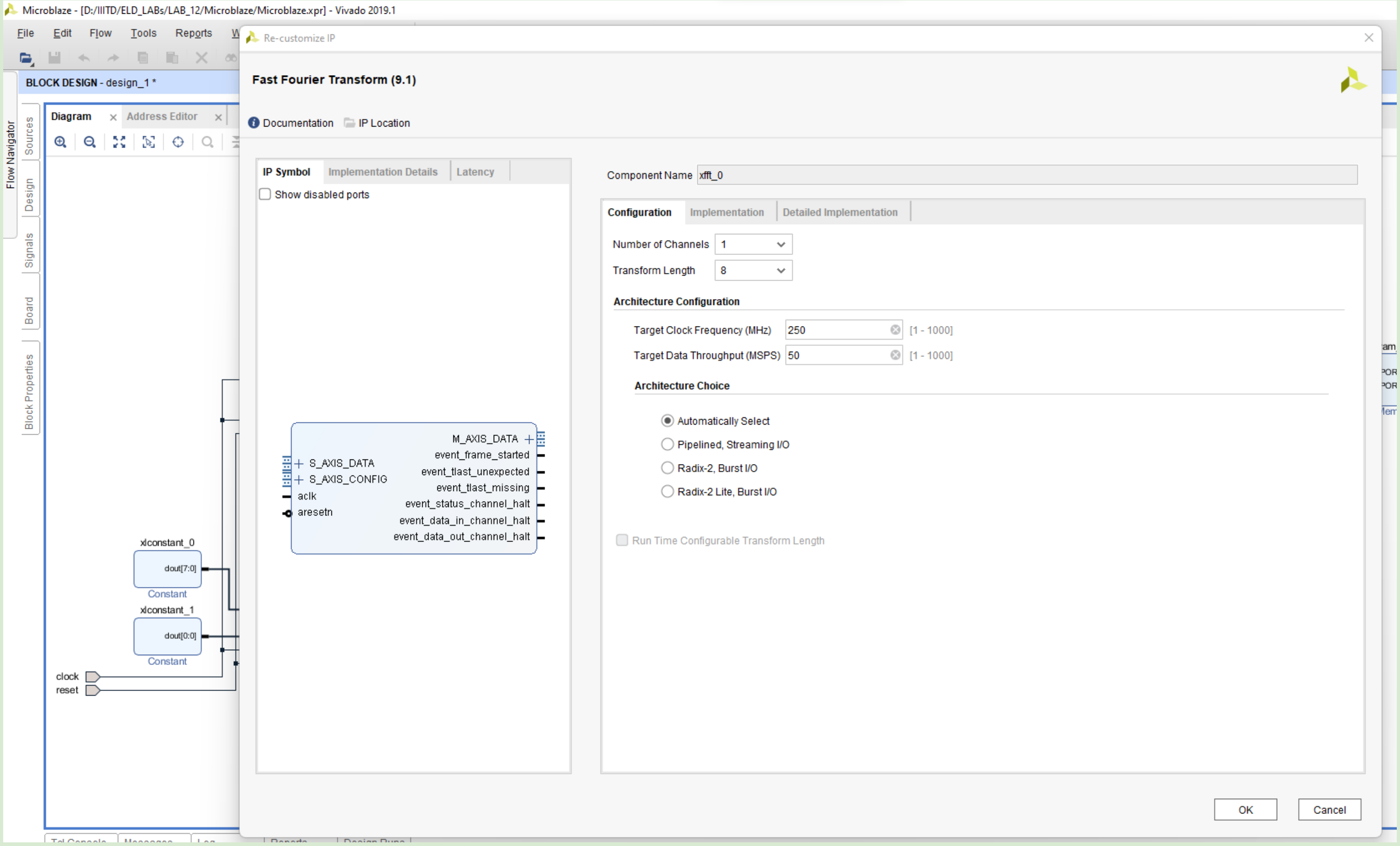
b) In the Enable Read Channel, increase the Memory Map Data Width and Stream Data Width to 64. Select Max Burst Size of 256.

c) In the Enable Write Channel, increase the Memory Map Data Width to 64. Select Max Burst Size of 256. Click on Ok.

- Click on Run Connection Automation, in the Run Connection Automation window select All Automation and click on OK. Two slave ports are added to the AXI interconnect.

- The AXI DMA will perform read and write operations from the Block Memory Generator via AXI interconnect. The DMA port M_AXI_MM2S is for reading the data and the port M_AXI_S2MM is for writing the data into the Block Memory Generator. The DMA gets configured by the Microblaze through the S_AXI_LITE port.

- ***FFT IP***



Microblaze - [D:/IIITD/ELD_LABs/LAB_12/Microblaze/Microblaze.xpr] - Vivado 2019.1

Re-customize IP

Fast Fourier Transform (9.1)

Component Name: xfft_0

Configuration

Number of Channels: 1

Transform Length: 8

Architecture Configuration

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

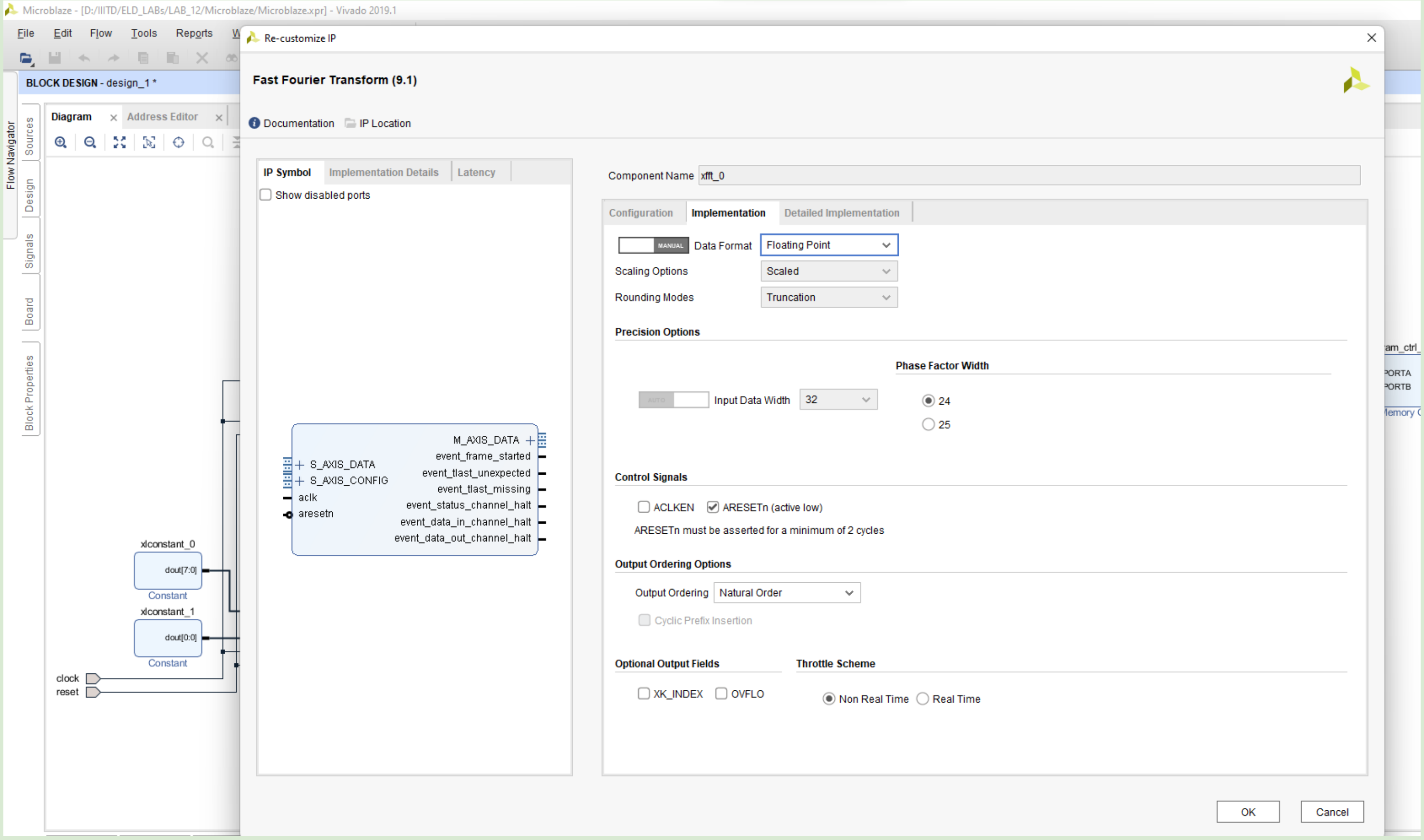
Target Data Throughput (MSPS): 50 [1 - 1000]

Architecture Choice

● Automatically Select
○ Pipelined, Streaming I/O
○ Radix-2, Burst I/O
○ Radix-2 Lite, Burst I/O

□ Run Time Configurable Transform Length

Now add the Fast Fourier Transform IP and open its configuration window and Change the transform length to 8 since we are performing 8-point FFT



Fast Fourier Transform (9.1)

Component Name xfft_0

Configuration | **Implementation** | Detailed Implementation

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

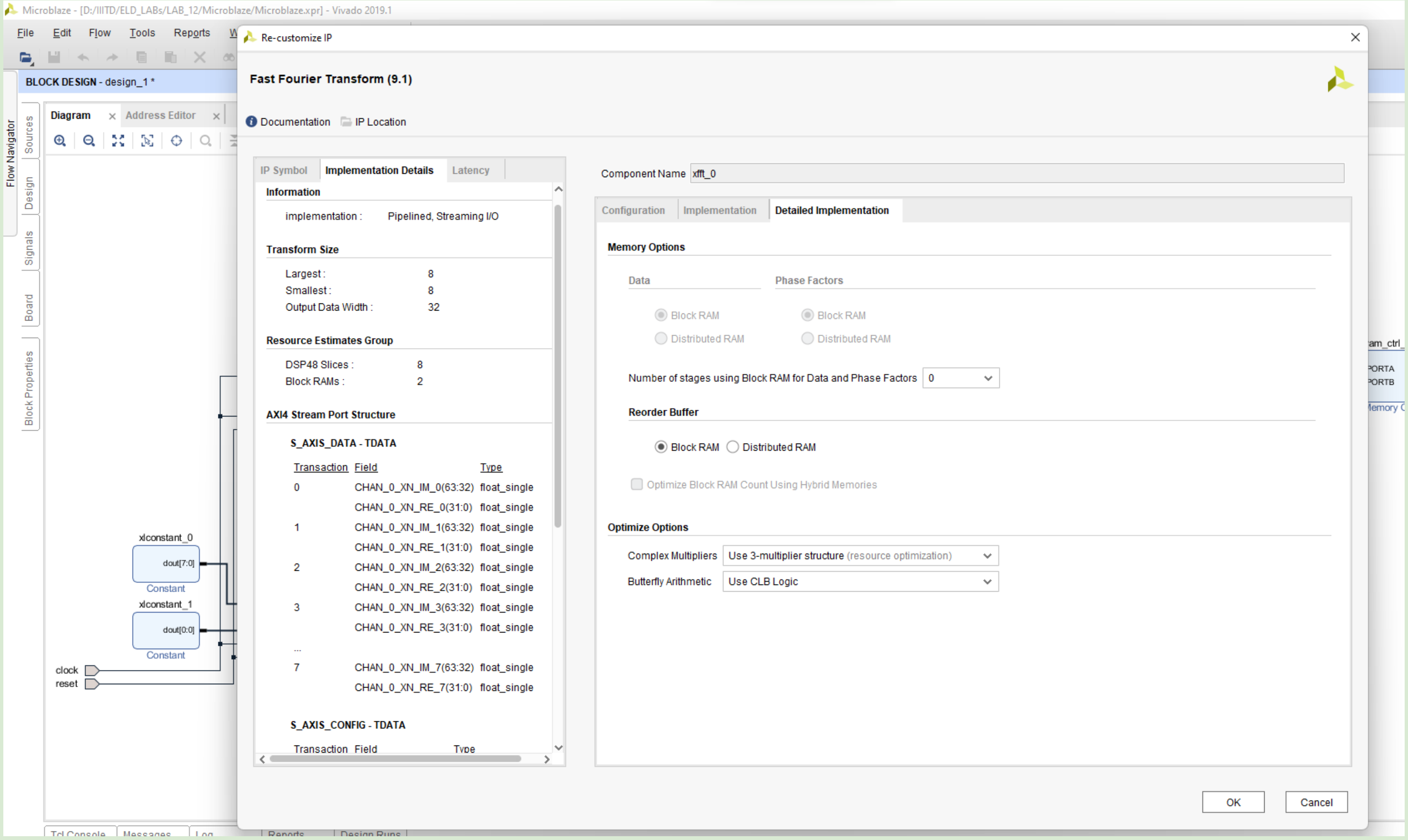
Optional Output Fields

□ XK_INDEX □ OVFLO

Throttle Scheme

● Non Real Time ○ Real Time

Under the implementation tab, change the Data Format to Floating Point. Add the control signal ARESETn. Select the Output Ordering as Natural Order. Click on OK.



Fast Fourier Transform (9.1)

Implementation Details

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
...		
7	CHAN_0_XN_IM_7(63:32)	float_single
	CHAN_0_XN_RE_7(31:0)	float_single

S_AXIS_CONFIG - TDATA

Transaction Field Type

Component Name xfft_0

Configuration | Implementation | **Detailed Implementation**

Memory Options

Data: ● Block RAM ○ Distributed RAM

Phase Factors: ● Block RAM ○ Distributed RAM

Number of stages using Block RAM for Data and Phase Factors: 0

Reorder Buffer

● Block RAM ○ Distributed RAM

□ Optimize Block RAM Count Using Hybrid Memories

Optimize Options

Complex Multipliers: Use 3-multiplier structure (resource optimization)

Butterfly Arithmetic: Use CLB Logic

- ***FFT IP connections***

a) Manually connect the aclk and aresetn of FFT IP to the common Clock port and Reset port respectively.

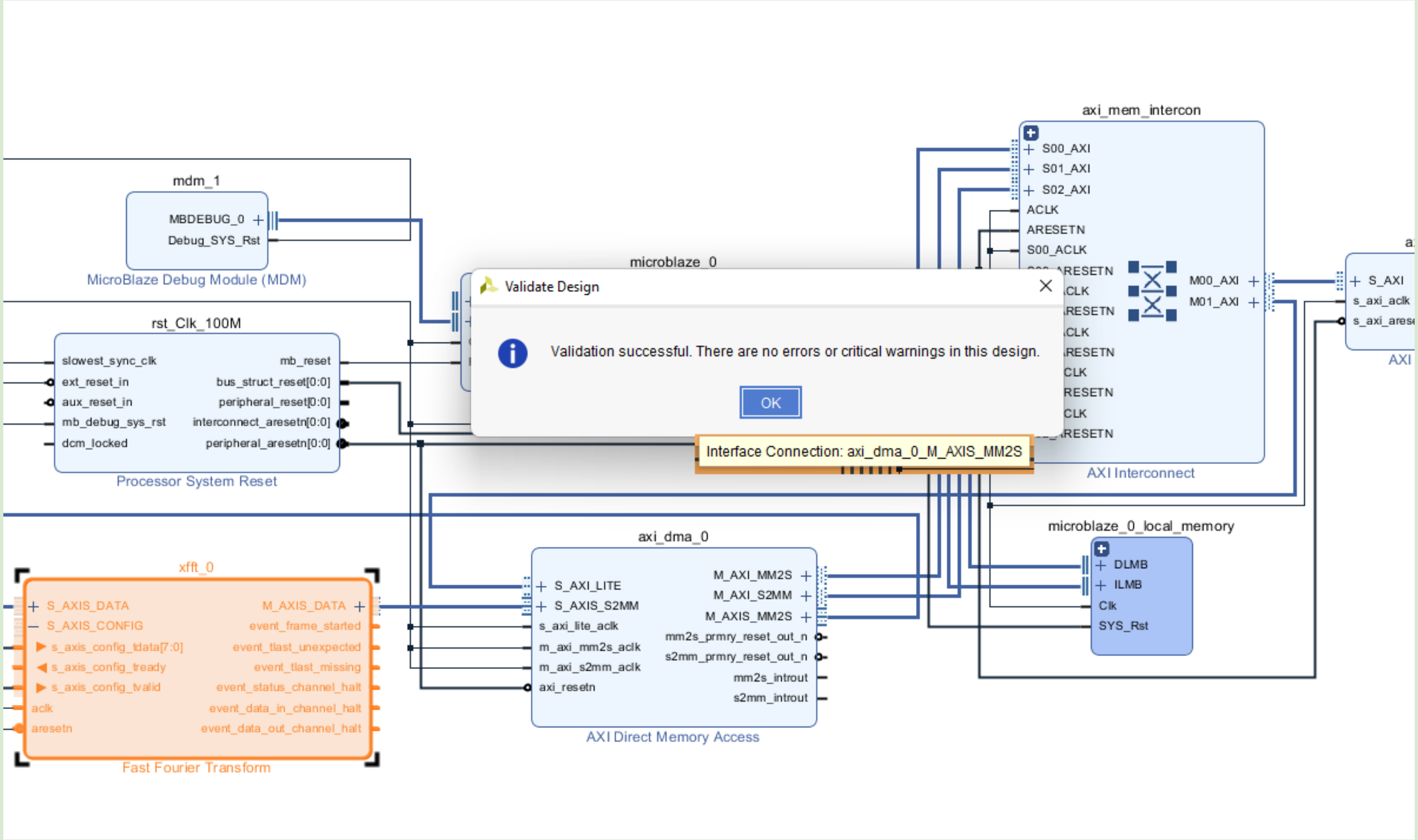
b) For Configuration of the FFT IP, the S_axis_config_tdata and S_axis_config_tvalid should be manually connected to constants of value 1.

c) S_axis_config_tdata [7:0] is connected to a constant of width 8 and value 1.
d) S_axis_config_tvalid is connected to a constant of width 1 and value 1.

e) Manually connect the M_AXIS_MM2S port of the DMA IP to the S_AXIS_DATA port of the FFT IP. The DMA IP transfers the data read from the BRAM to the FFT IP.

f) Manually connect the S_AXIS_S2MM port of the DMA IP to the M_AXIS_DATA port of the FFT IP. The FFT IP transfers the data after processing to the DMA.

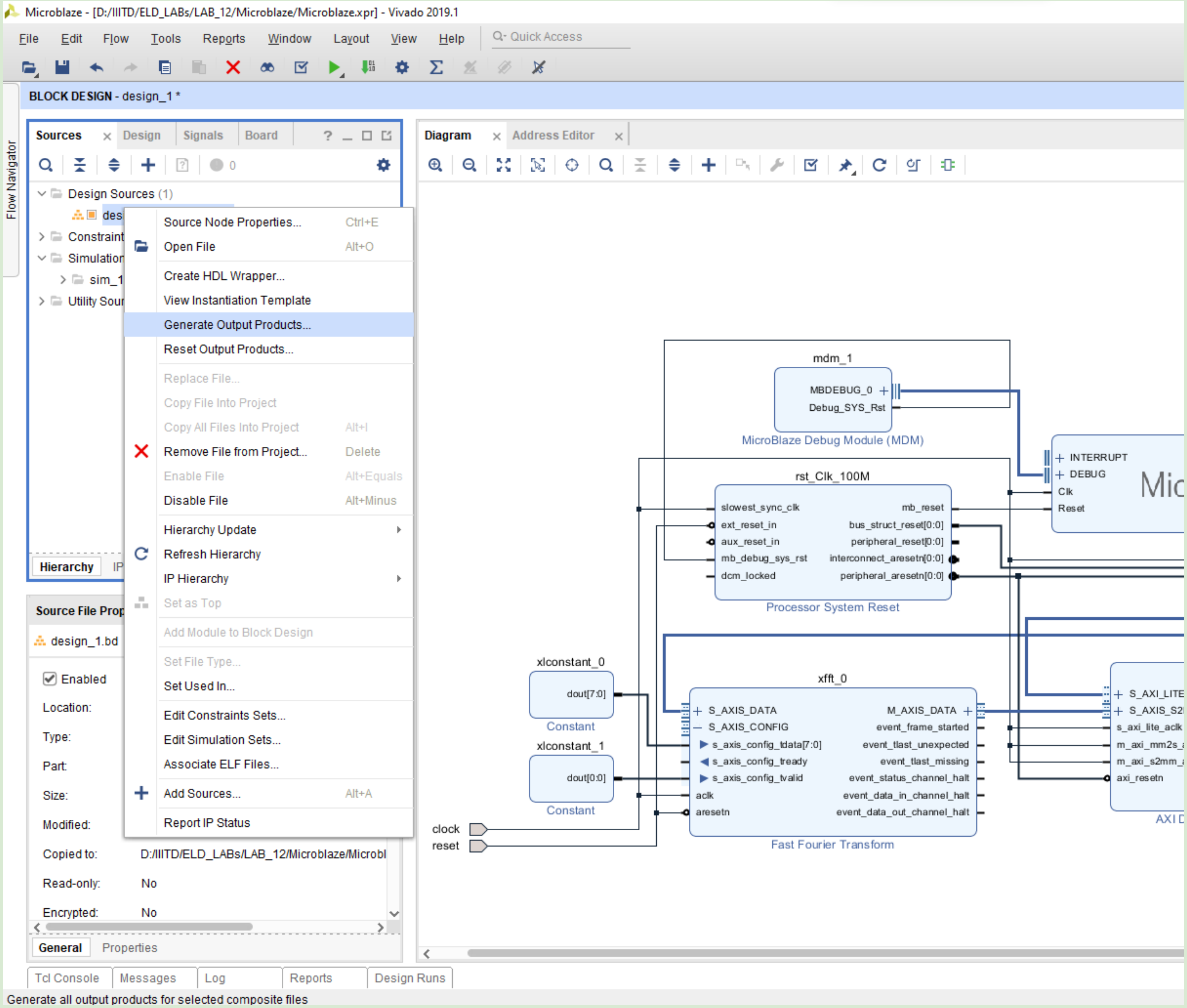
### Validate the design, generate output product, and create HDL wrapper. DO NOT ever miss these steps.



Validate Design

Validation successful. There are no errors or critical warnings in this design.

Interface Connection: axi_dma_0_M_AXIS_MM2S



Location:

Type:

Part:

Size:

Modified:

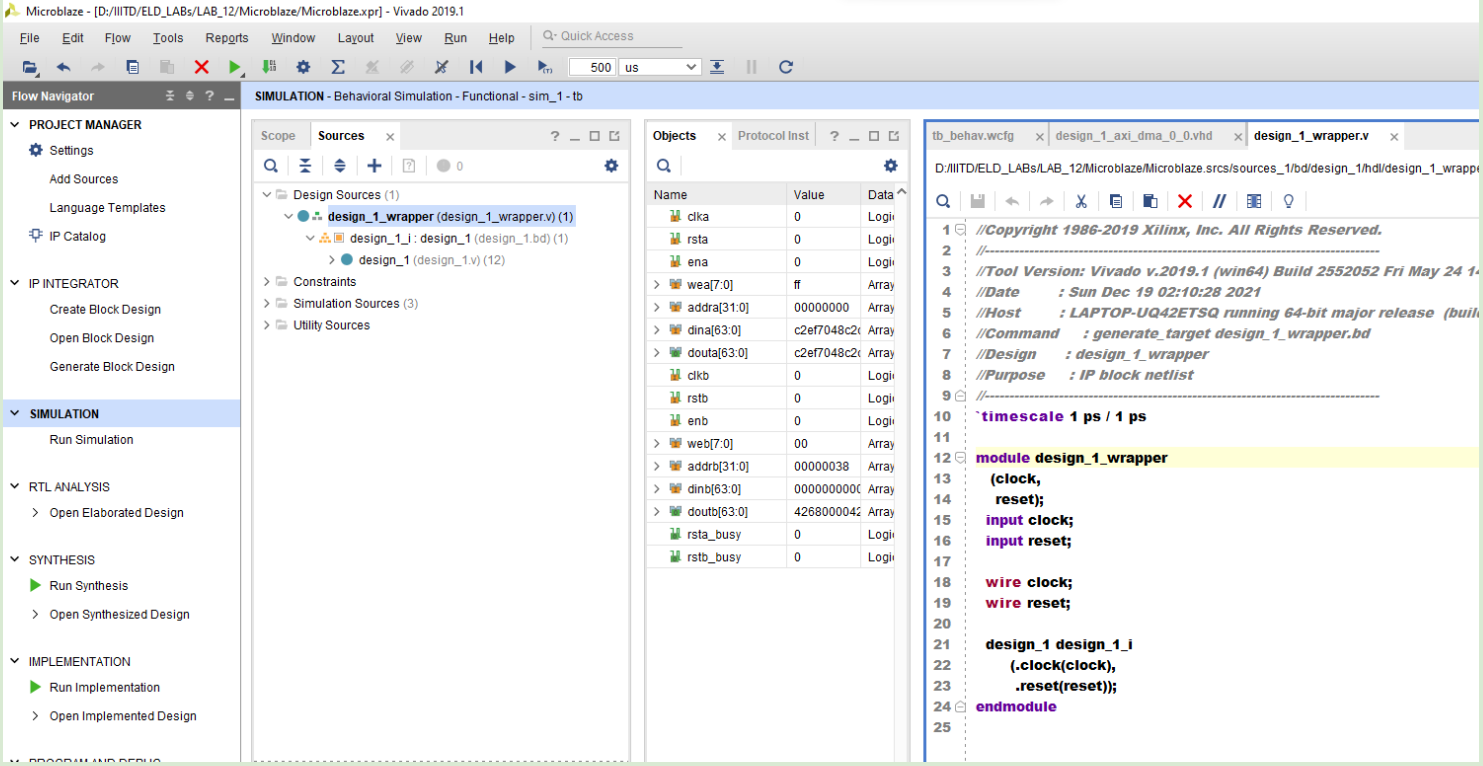
Copied to: D:/IIITD/ELD_LABs/LAB_12/Microblaze/Microbl

Read-only: No

Encrypted: No

Generate all output products for selected composite files

Auto_Generated_HDL_Wrapper



SIMULATION - Behavioral Simulation - Functional - sim_1 - tb

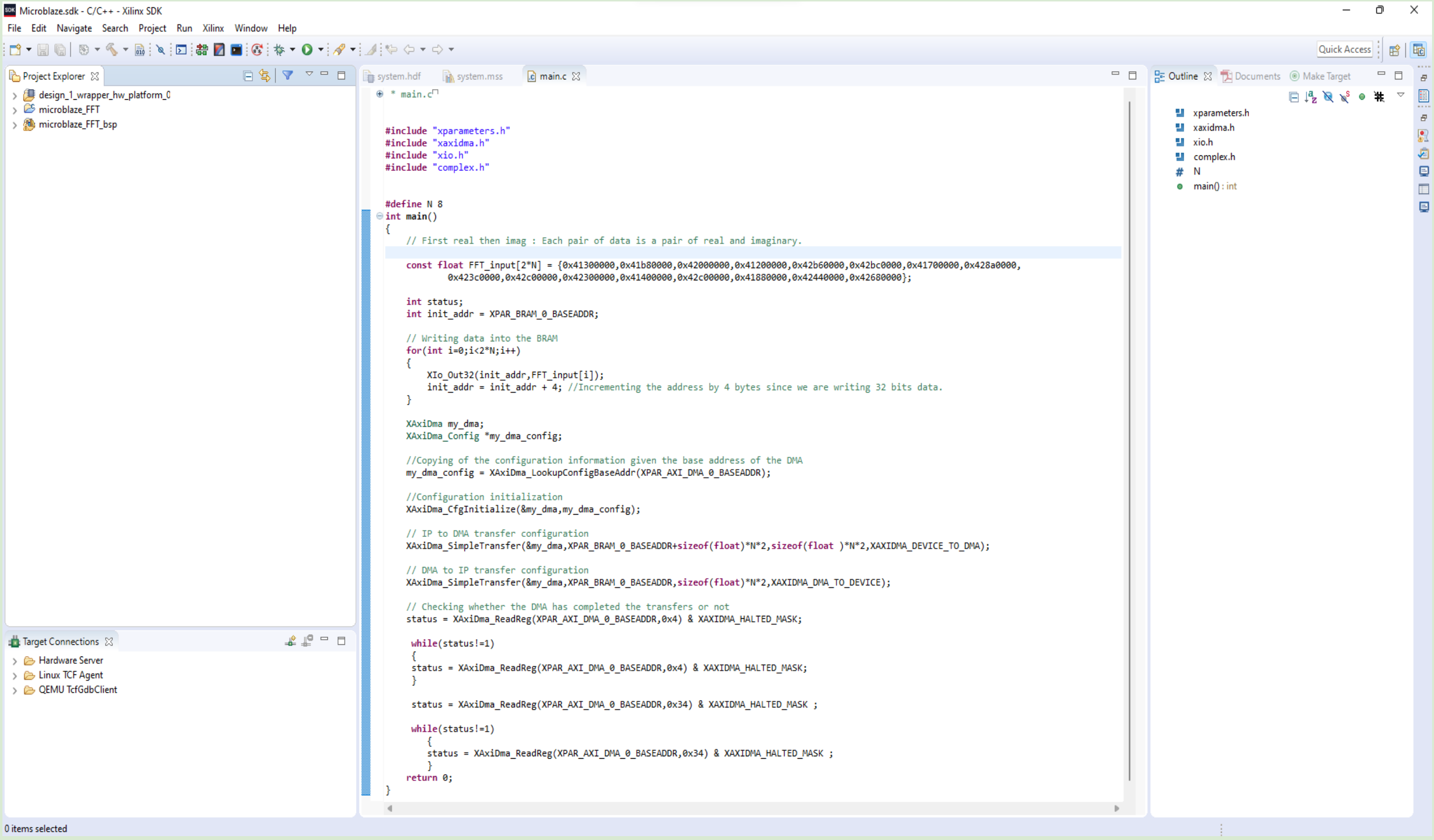
Design Sources (1)
- design_1_wrapper (design_1_wrapper.v) (1)
  - design_1_i : design_1 (design_1.bd) (1)
    - design_1 (design_1.v) (12)

Name	Value	Data
clka	0	Logi
rsta	0	Logi
ena	0	Logi
wea[7:0]	ff	Array
addra[31:0]	00000000	Array
dina[63:0]	c2ef7048c2	Array
douta[63:0]	c2ef7048c2	Array
clkb	0	Logi
rstb	0	Logi
enb	0	Logi
web[7:0]	00	Array
addrb[31:0]	00000038	Array
dinb[63:0]	000000000	Array
doutb[63:0]	426800042	Array
rsta_busy	0	Logi
rstb_busy	0	Logi

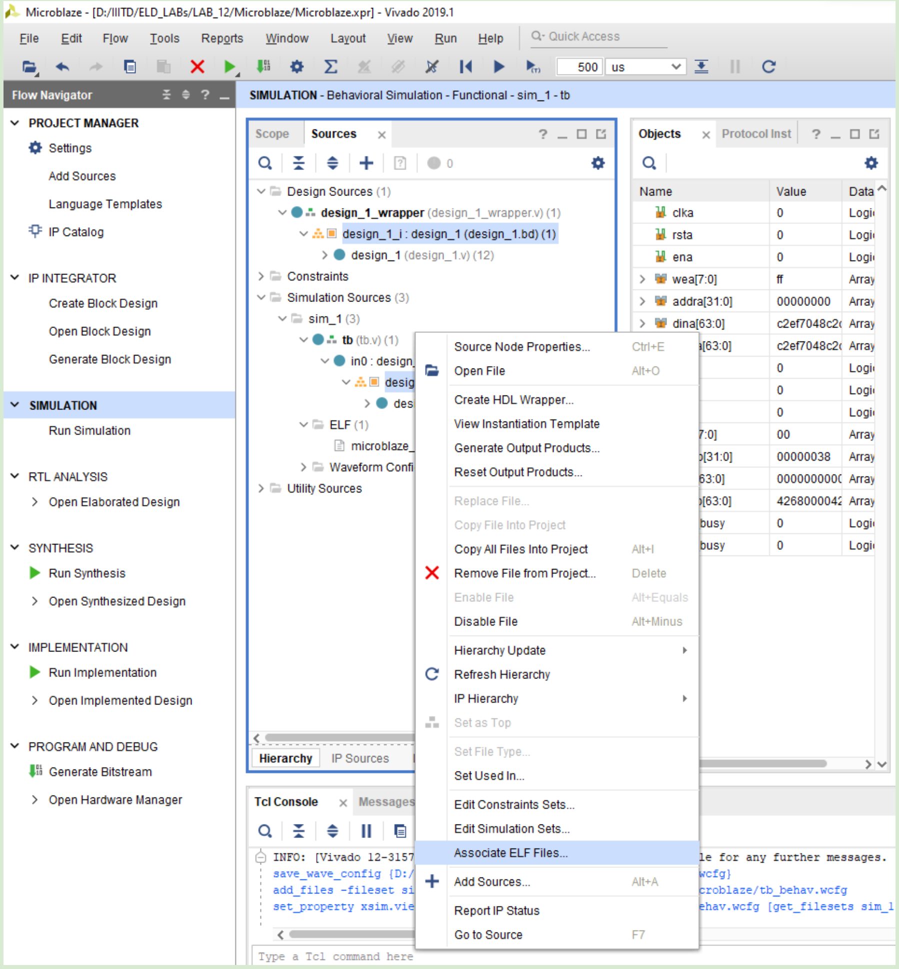
D:/IIITD/ELD_LABs/LAB_12/Microblaze/Microblaze.srcs/sources_1/bd/design_1/hdl/design_1_wrappe

```verilog
//Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
//--------------------------------------------------------------------------------
//Tool Version: Vivado v.2019.1 (win64) Build 2552052 Fri May 24 1
//Date        : Sun Dec 19 02:10:28 2021
//Host        : LAPTOP-UQ42ETSQ running 64-bit major release  (buil
//Command     : generate_target design_1_wrapper.bd
//Design      : design_1_wrapper
//Purpose     : IP block netlist
//--------------------------------------------------------------------------------
`timescale 1 ps / 1 ps

module design_1_wrapper
   (clock,
    reset);
  input clock;
  input reset;

  wire clock;
  wire reset;

  design_1 design_1_i
       (.clock(clock),
        .reset(reset));
endmodule
```

Launching SDK and program in C

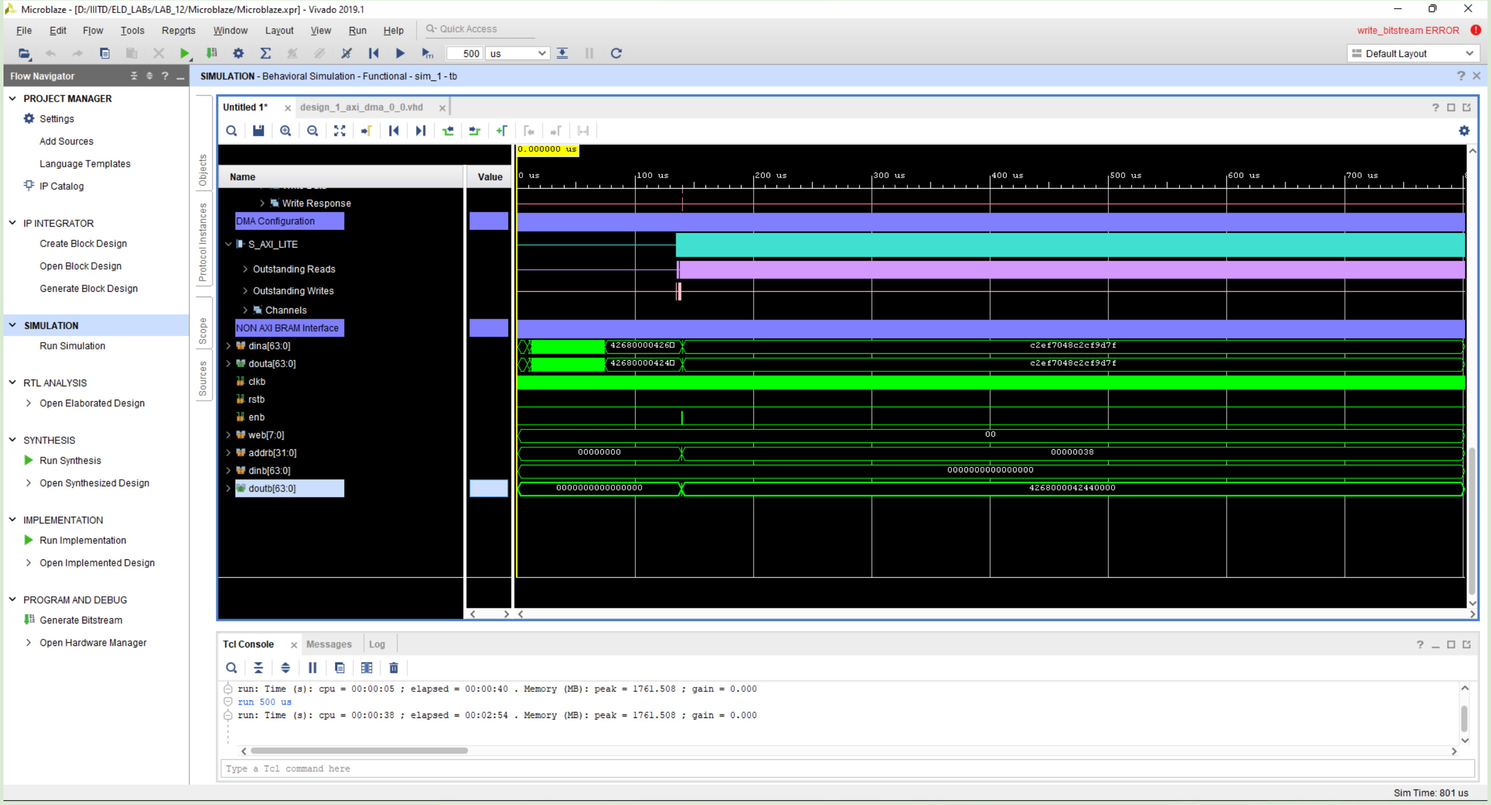


```c
 * main.c

#include "xparameters.h"
#include "xaxidma.h"
#include "xio.h"
#include "complex.h"


#define N 8
int main()
{
    // First real then imag : Each pair of data is a pair of real and imaginary.

    const float FFT_input[2*N] = {0x41300000,0x41b80000,0x42000000,0x41200000,0x42b60000,0x42bc0000,0x41700000,0x428a0000,
            0x423c0000,0x42c00000,0x42300000,0x41400000,0x42c00000,0x41880000,0x42440000,0x42680000};

    int status;
    int init_addr = XPAR_BRAM_0_BASEADDR;

    // Writing data into the BRAM
    for(int i=0;i<2*N;i++)
    {
        XIo_Out32(init_addr,FFT_input[i]);
        init_addr = init_addr + 4; //Incrementing the address by 4 bytes since we are writing 32 bits data.
    }

    XAxiDma my_dma;
    XAxiDma_Config *my_dma_config;

    //Copying of the configuration information given the base address of the DMA
    my_dma_config = XAxiDma_LookupConfigBaseAddr(XPAR_AXI_DMA_0_BASEADDR);

    //Configuration initialization
    XAxiDma_CfgInitialize(&my_dma,my_dma_config);

    // IP to DMA transfer configuration
    XAxiDma_SimpleTransfer(&my_dma,XPAR_BRAM_0_BASEADDR+sizeof(float)*N*2,sizeof(float )*N*2,XAXIDMA_DEVICE_TO_DMA);

    // DMA to IP transfer configuration
    XAxiDma_SimpleTransfer(&my_dma,XPAR_BRAM_0_BASEADDR,sizeof(float)*N*2,XAXIDMA_DMA_TO_DEVICE);

    // Checking whether the DMA has completed the transfers or not
    status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x4) & XAXIDMA_HALTED_MASK;

     while(status!=1)
     {
     status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x4) & XAXIDMA_HALTED_MASK;
     }

     status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) & XAXIDMA_HALTED_MASK ;

     while(status!=1)
        {
        status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) & XAXIDMA_HALTED_MASK ;
        }
    return 0;
}
```

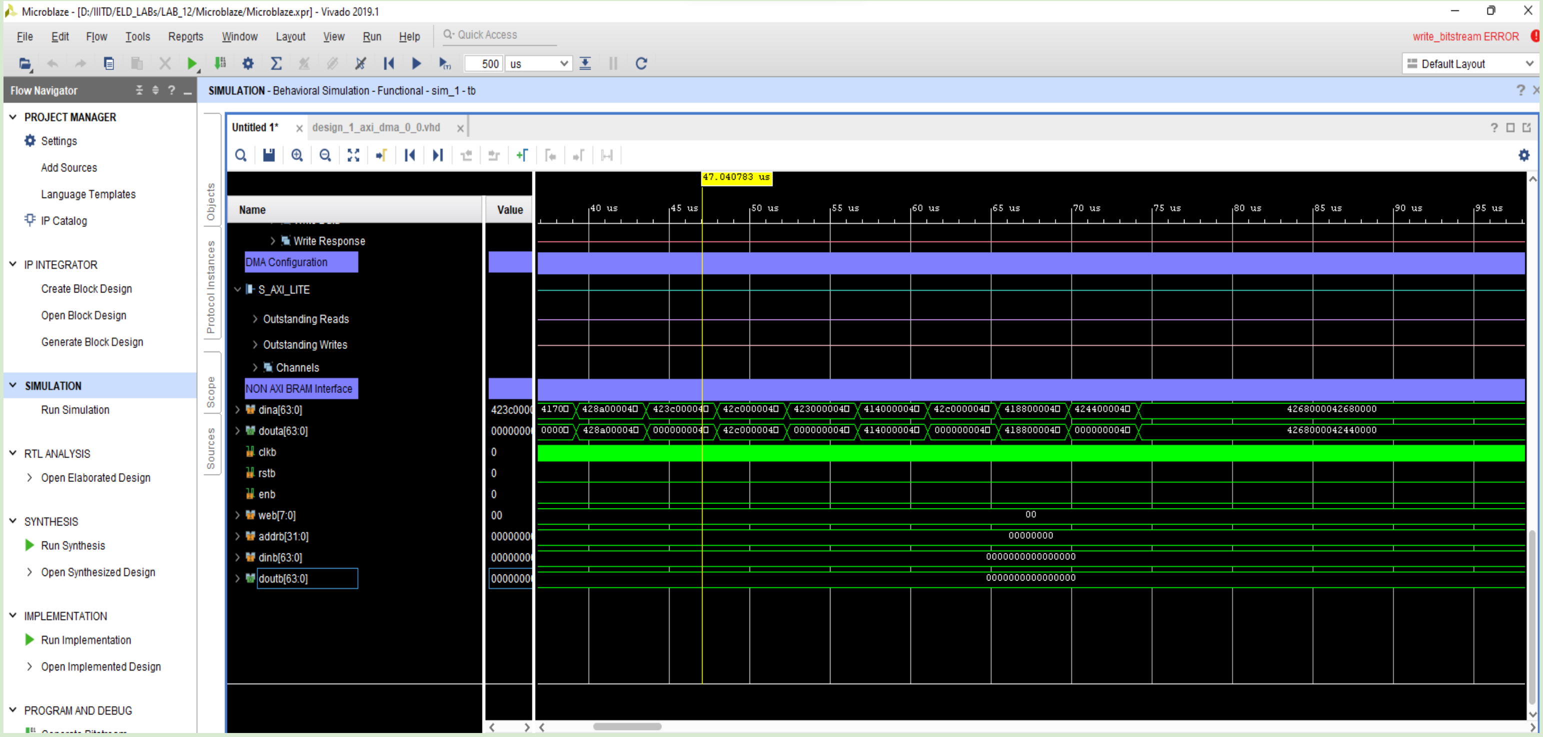


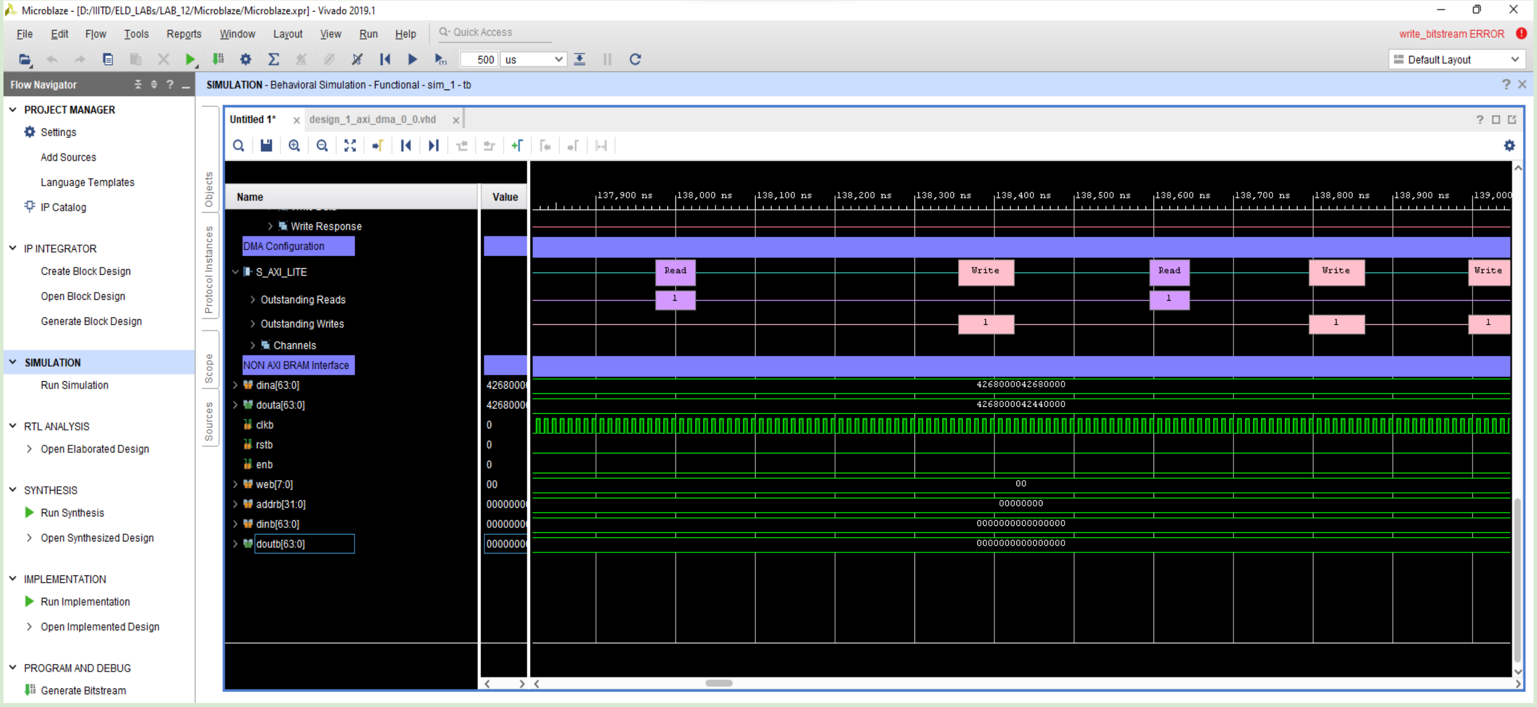
Open Vivado. Under the simulation sources, right click on the design and click on Associate ELF files. Click on the 3 dots to add the ELF file.

## running Behavioral Simulation



```
run: Time (s): cpu = 00:00:05 ; elapsed = 00:00:40 . Memory (MB): peak = 1761.508 ; gain = 0.000
run 500 us
run: Time (s): cpu = 00:00:38 ; elapsed = 00:02:54 . Memory (MB): peak = 1761.508 ; gain = 0.000
```





Conclusion:

Successfully created MIcroblaze PS block design in Vivado and wrote the C program following which ran simulations using a testbench to test the block design successful implementation.