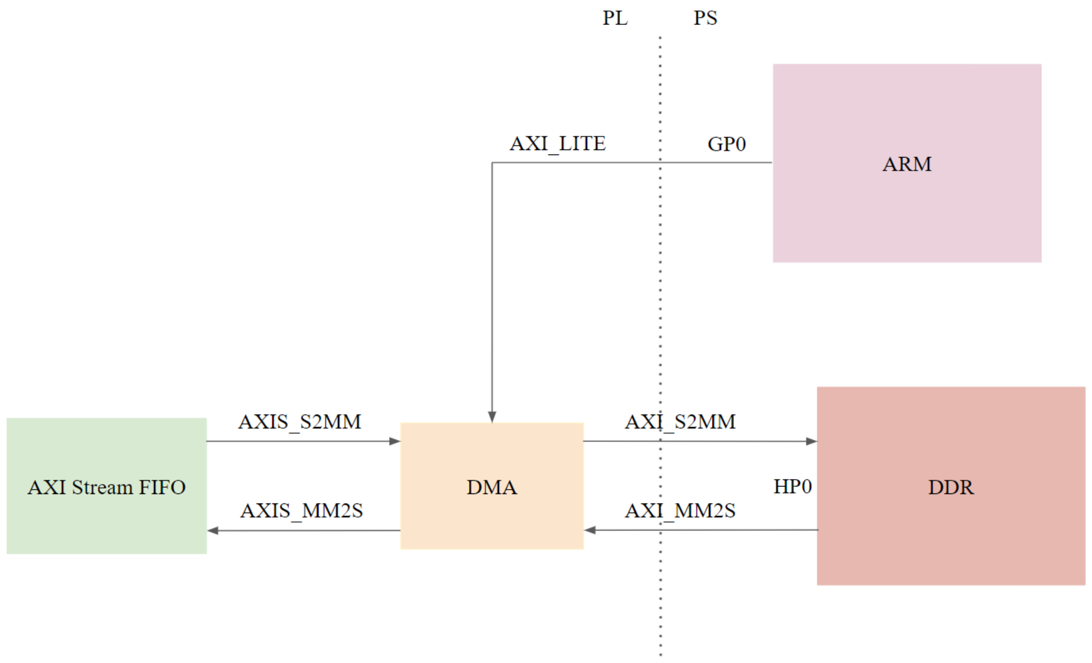
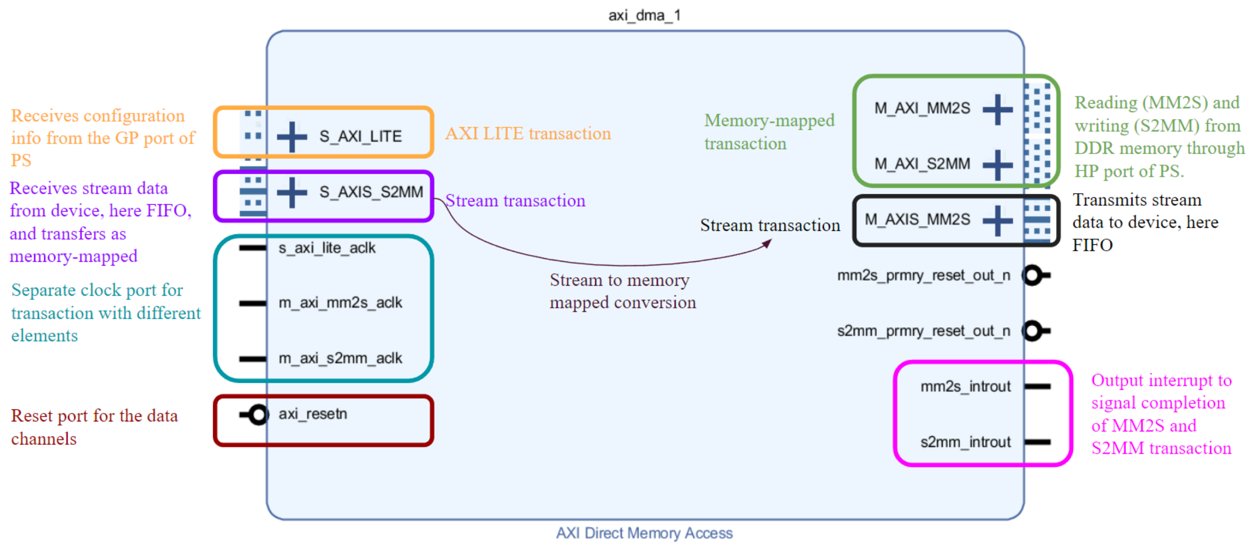
ELD Lab10 Handout

Intro to lab:
In this lab, we will be using AXI DMA (Direct Memory Access) to perform data transfers between PS and PL. We will first have some data stored in the DDR memory. 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 a data FIFO IP in the PL part. Then the DMA will read this data from the FIFO, and write it to the DDR memory again.
The DMA can be used in two modes: 1) interrupt mode or 2) polling mode. In the interrupt mode, the DMA generates the interrupt signals to confirm the completion of the data transaction (from/to DDR). Whereas in the polling mode, we just wait for the DMA buses to be idle by checking the halt and idle register values. In this lab, we will be using the polling mode.

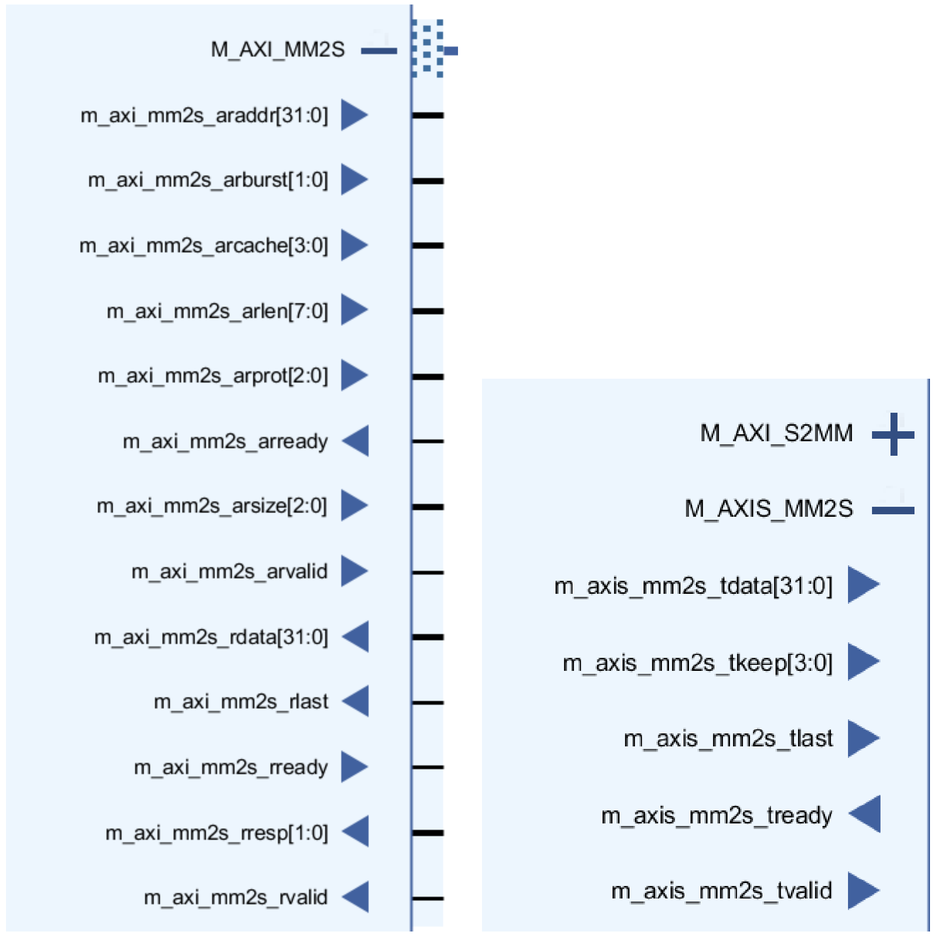
Design Overview:
The connections between the various elements of the design can be illustrated in the figure below.



A brief explanation of various ports in DMA IP in the vivado is given below.

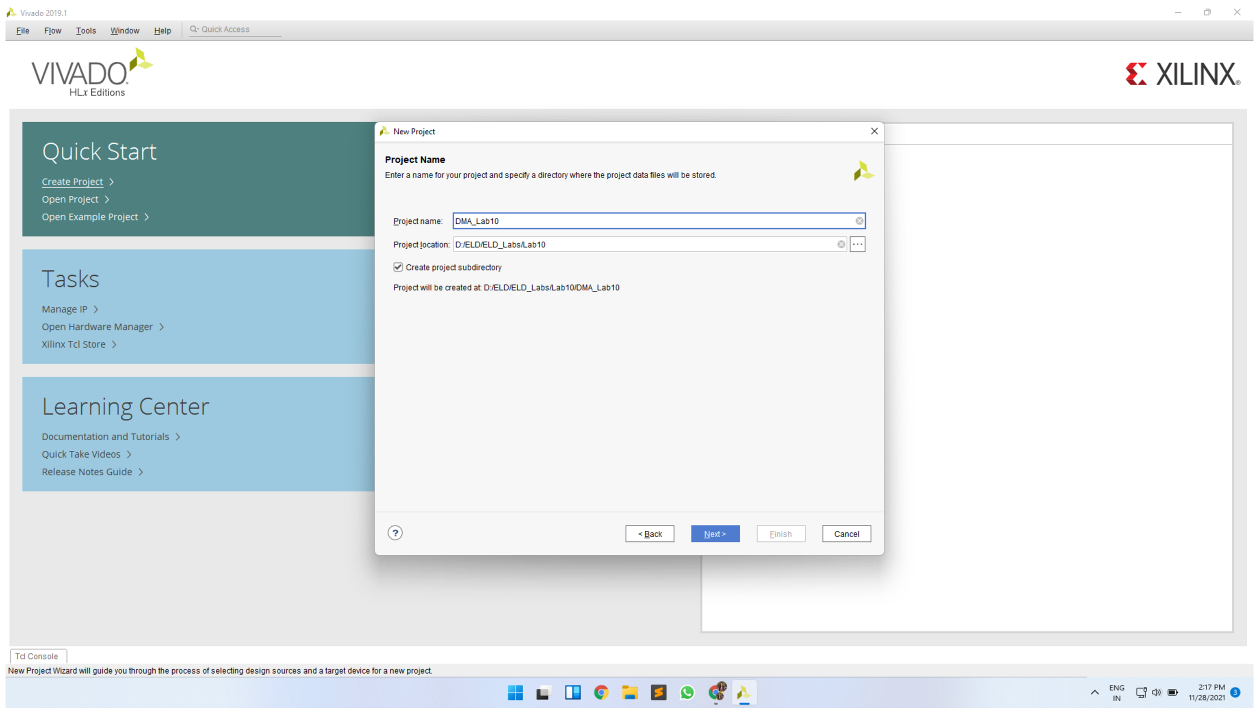

axi_dma_1
Receives configuration info from the GP port of PS
S_AXI_LITE
AXI LITE transaction
Receives stream data from device, here FIFO, and transfers as memory-mapped
S_AXIS_S2MM
Stream transaction
Separate clock port for transaction with different elements
s_axi_lite_aclk
m_axi_mm2s_aclk
m_axi_s2mm_aclk
Reset port for the data channels
axi_resetn
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
Stream to memory mapped conversion
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

The comparison of ports in M_AXI_MM2S and M_AXI***S***_MM2S is shown in the figure below. In the AXI, we have read address, read address burst, data burst, data length, etc parameters as well apart from the data, valid and ready channels.



Steps:

1. Open Vivado 2019.1. Create New Project.



Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: DMA_Lab10

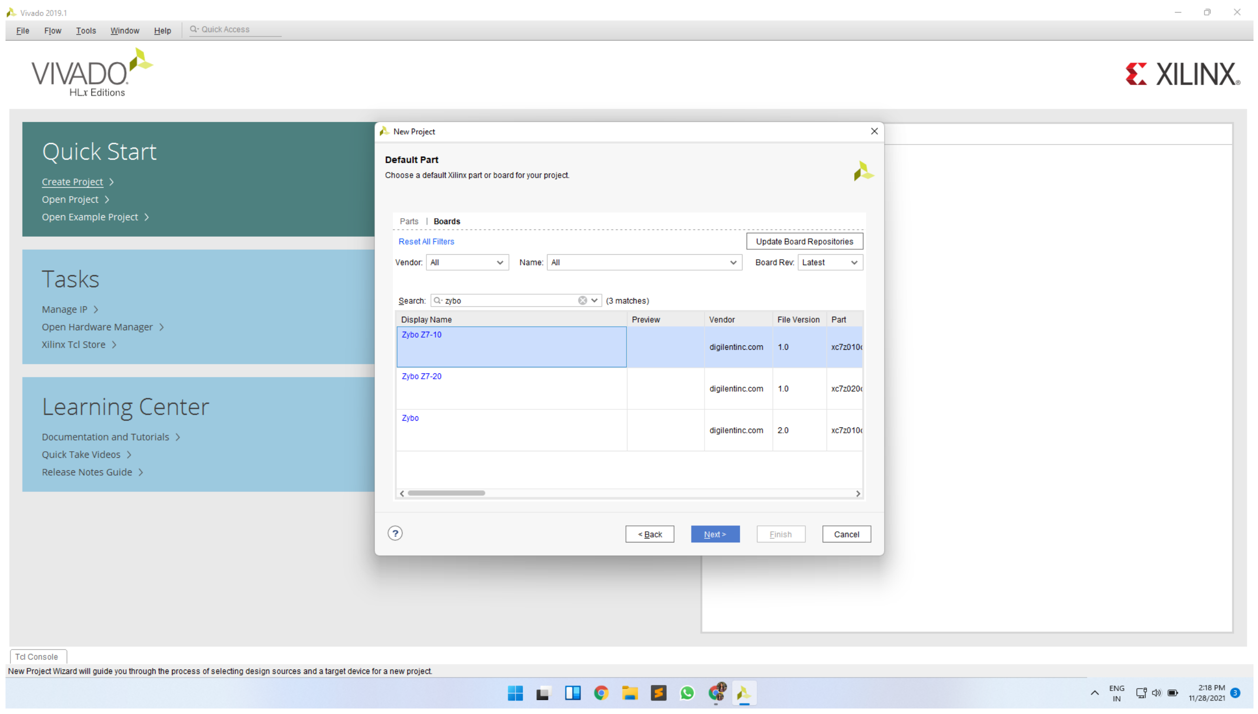
Project location: D:/ELD/ELD_Labs/Lab10

[✓] Create project subdirectory

Project will be created at: D:/ELD/ELD_Labs/Lab10/DMA_Lab10

New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.

2. Click Next.
3. Select Zybo Z7-10 board from the boards' tab in the default part window.



Default Part

Choose a default Xilinx part or board for your project.

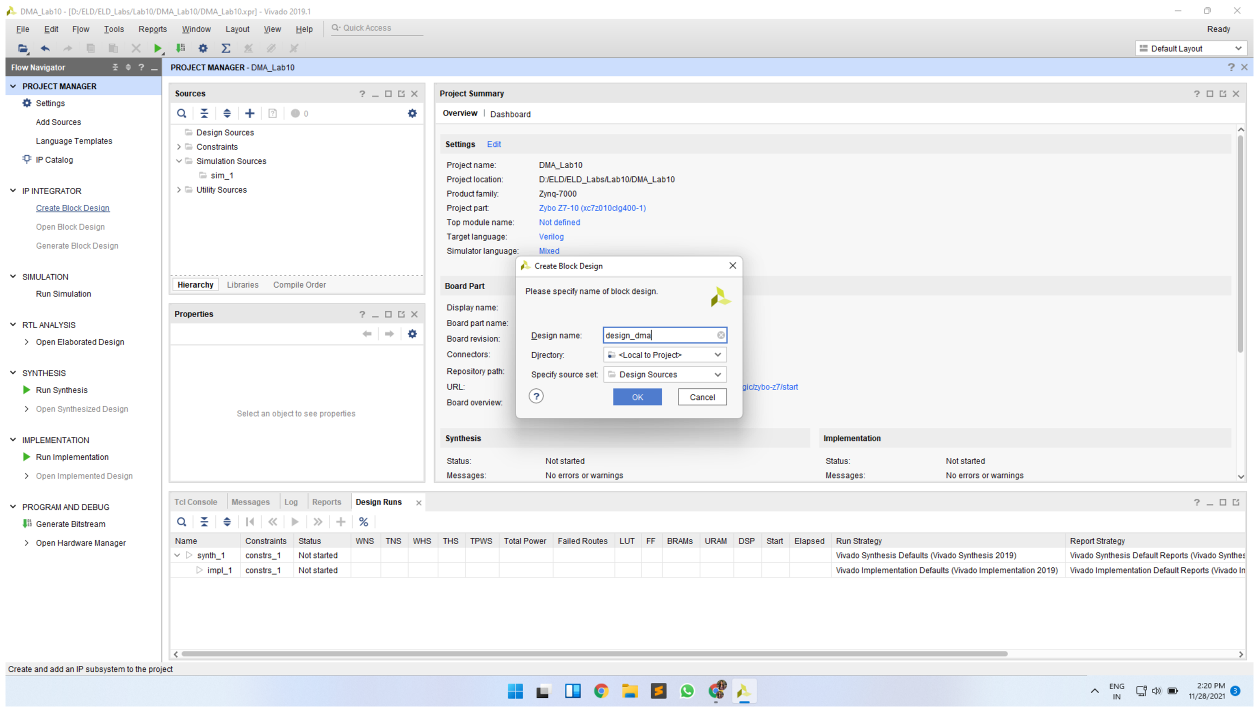
Vendor: All Name: All Board Rev: Latest

Search: zybo (3 matches)

Display Name	Preview	Vendor	File Version	Part
Zybo Z7-10		digilentinc.com	1.0	xc7z010
Zybo Z7-20		digilentinc.com	1.0	xc7z020
Zybo		digilentinc.com	2.0	xc7z010

New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.

4. Again click on Finish to create the project.
5. In the project, now click on create block design and name it as **design_dma**.


DMA_Lab10 - [D:/ELD/ELD_Labs/Lab10/DMA_Lab10/DMA_Lab10.xpr] - Vivado 2019.1
File Edit Flow Tools Reports Window Layout View Help
Ready
Default Layout
Flow Navigator
PROJECT MANAGER - DMA_Lab10
Create Block Design
OK
Cancel

Project Summary

Settings

Project name:	DMA_Lab10
Project location:	D:/ELD/ELD_Labs/Lab10/DMA_Lab10
Product family:	Zynq-7000
Project part:	Zybo Z7-10 (xc7z010clg400-1)
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

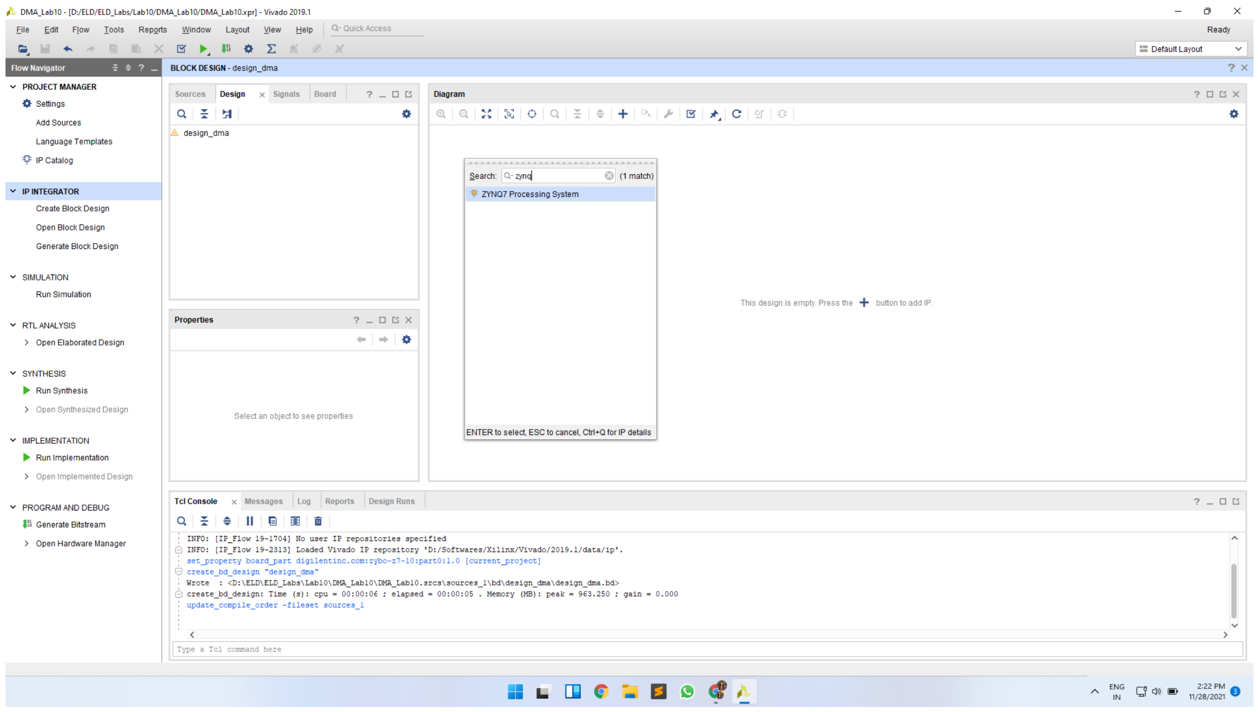
Please specify name of block design.

Design name: design_dma
Directory: <Local to Project>
Specify source set: Design Sources

Synthesis Status: Not started Messages: No errors or warnings
Implementation Status: Not started Messages: No errors or warnings

Create and add an IP subsystem to the project

6. Click on + sign on the block design window to add a new IP (or press Ctrl+I for the same). Search for Zynq Processing System IP.


DMA_Lab10 - [D:/ELD/ELD_Labs/Lab10/DMA_Lab10/DMA_Lab10.xpr] - Vivado 2019.1
BLOCK DESIGN - design_dma
Diagram
Search: zynq (1 match)
ZYNQ7 Processing System
This design is empty. Press the + button to add IP.

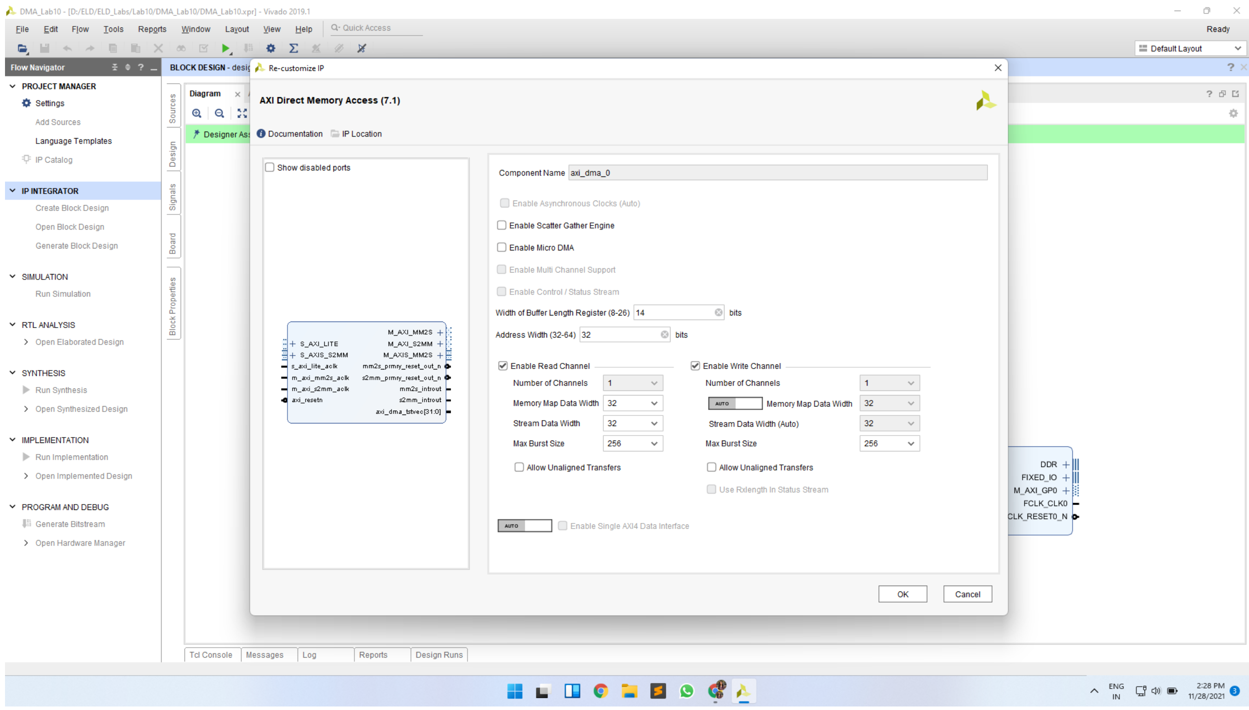
ENTER to select, ESC to cancel, Ctrl+Q for IP details

```
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'D:/Softwares/Xilinx/Vivado/2019.1/data/ip'.
set_property board_part digilentinc.com:zybo-z7-10:part0:1.0 [current_project]
create_bd_design "design_dma"
Wrote  : <D:\ELD\ELD_Labs\Lab10\DMA_Lab10\DMA_Lab10.srcs\sources_1\bd\design_dma\design_dma.bd>
create_bd_design: Time (s): cpu = 00:00:06 ; elapsed = 00:00:05 . Memory (MB): peak = 963.250 ; gain = 0.000
update_compile_order -fileset sources_1
```

7. Double click the IP to open the customization window. The following changes need to be made.

   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, press Ctrl+I (or right-click in the white space of block design and select Add IP), search for AXI Direct Memory Access IP, and add it to the design
9. Double click the AXI DMA IP to open its configuration window. The following changes need to be made to the IP.
   a. Unselect the Enable Scatter/Gather mode.
      i. This special mode of DMA is used to read from non-contiguous memory locations.
   b. Increase the maximum burst size to 256.
      i. This is done for maximized throughput of data transfers between DMA.



AXI Direct Memory Access (7.1)

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	32	Memory Map Data Width	32
Stream Data Width	32	Stream Data Width (Auto)	32
Max Burst Size	256	Max Burst Size	256
☐ Allow Unaligned Transfers		☐ Allow Unaligned Transfers	
		☐ Use Rxlength In Status Stream	

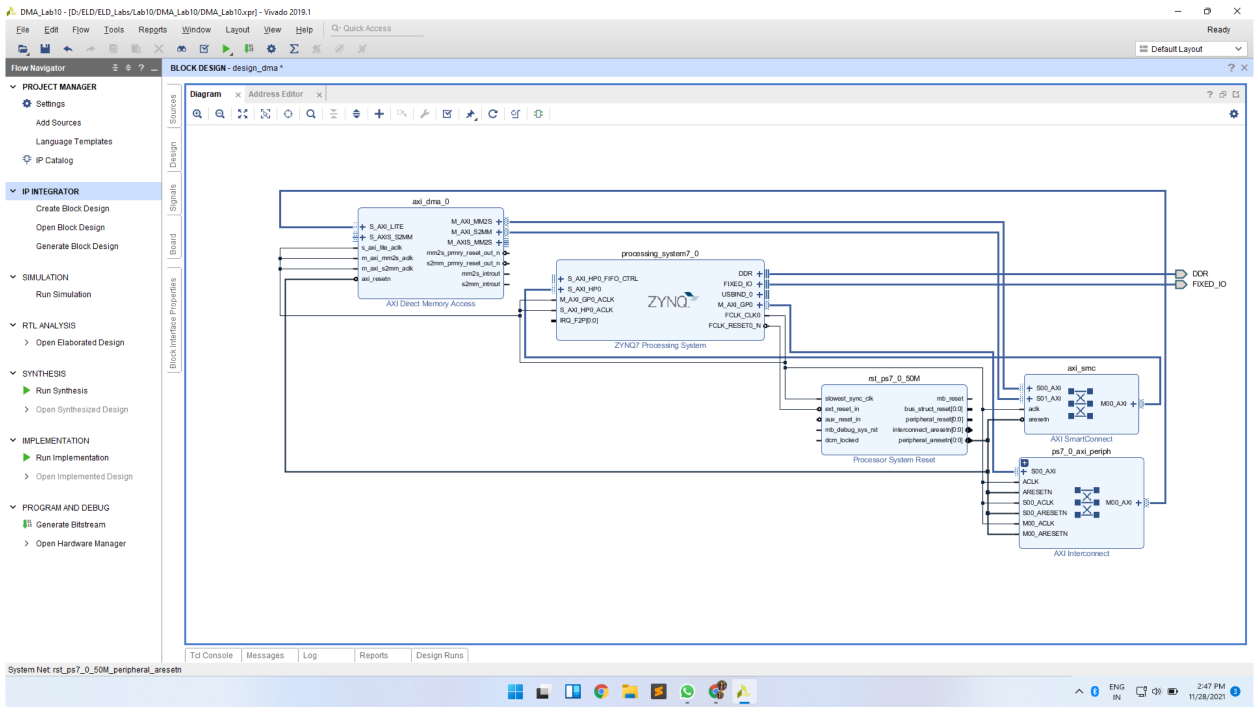
- [ ] Enable Single AXI4 Data Interface

10. Click on Run block automation. This will only 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.

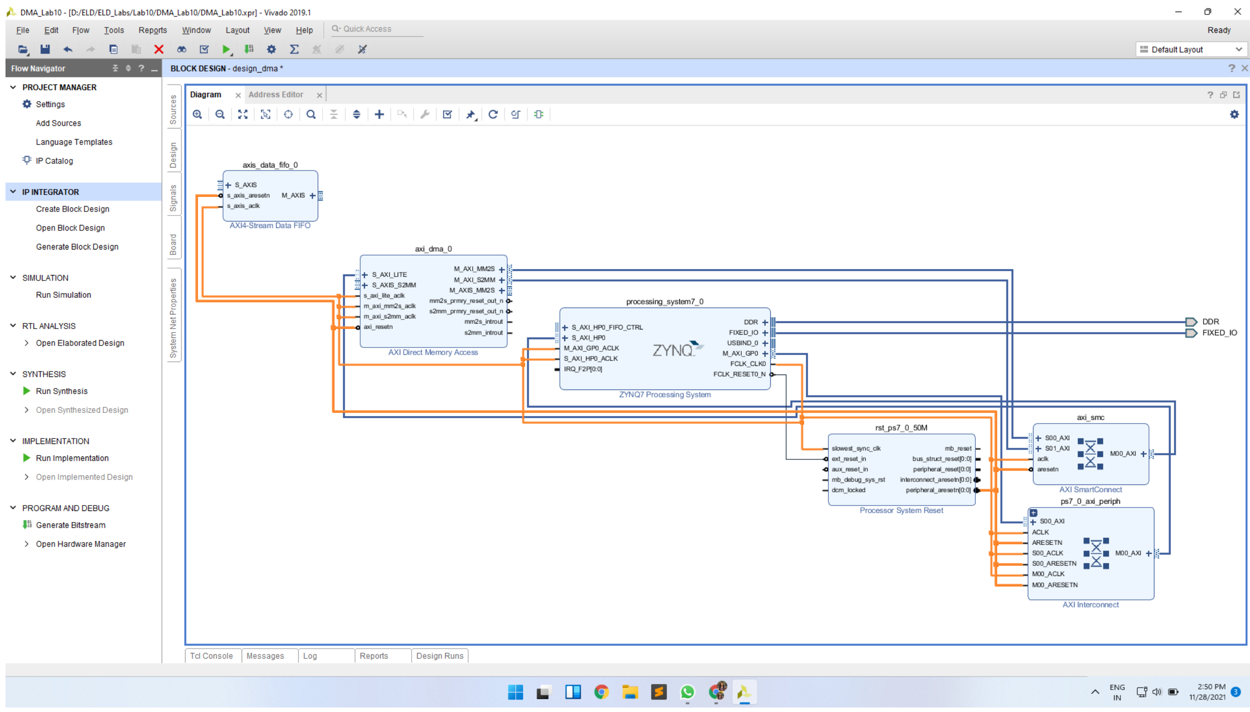
i. Since the configuration information is very small in size, the GP port of Zynq IP is used to transfer this information.

12. The block design should now look like in the figure below.



13. Now add the AXI Stream Data FIFO IP.

a. Connect the s_axis_clk and s_axis_aresetn ports of this IP to the clock and reset the network of the design respectively.



b. The S_AXIS port will be connected to the M_AXIS_MM2S port of the DMA.

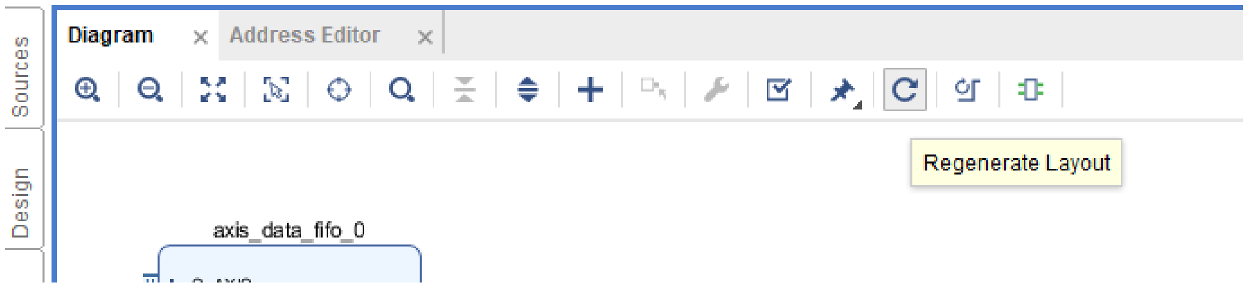
c. The M_AXIS port will be connected to the S_AXIS_S2MM port of the DMA.

14. (only for interrupt mode) Add Concat IP. This IP simply concatenates the input signals.

a. Connect its inputs in0[0:0] and in1[0:0] ports to the mm2s_introut and s2mm_introut ports of the DMA. These two are the interrupt signals generated by the DMA to signal the completion of MM2S and S2MM transactions respectively.

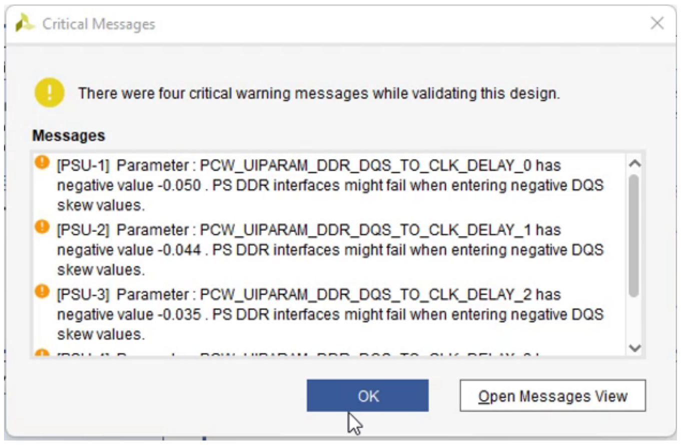
b. The output dout[1:0] will then be connected to the IRQ_F2P port of the Zynq IP.

15. The block design is now complete. Click on regenerate layout for a better view of the design.



16. Go to the Address Editor Tab. Check for any Unmapped Slaves or Excluded Segment. If any, right-click on that particular element and select Auto-Assign Address or Include Segment.

17. Then, validate your design to check for any missing or incorrect connection. If the following warnings appear, they can be ignored.


Critical Messages

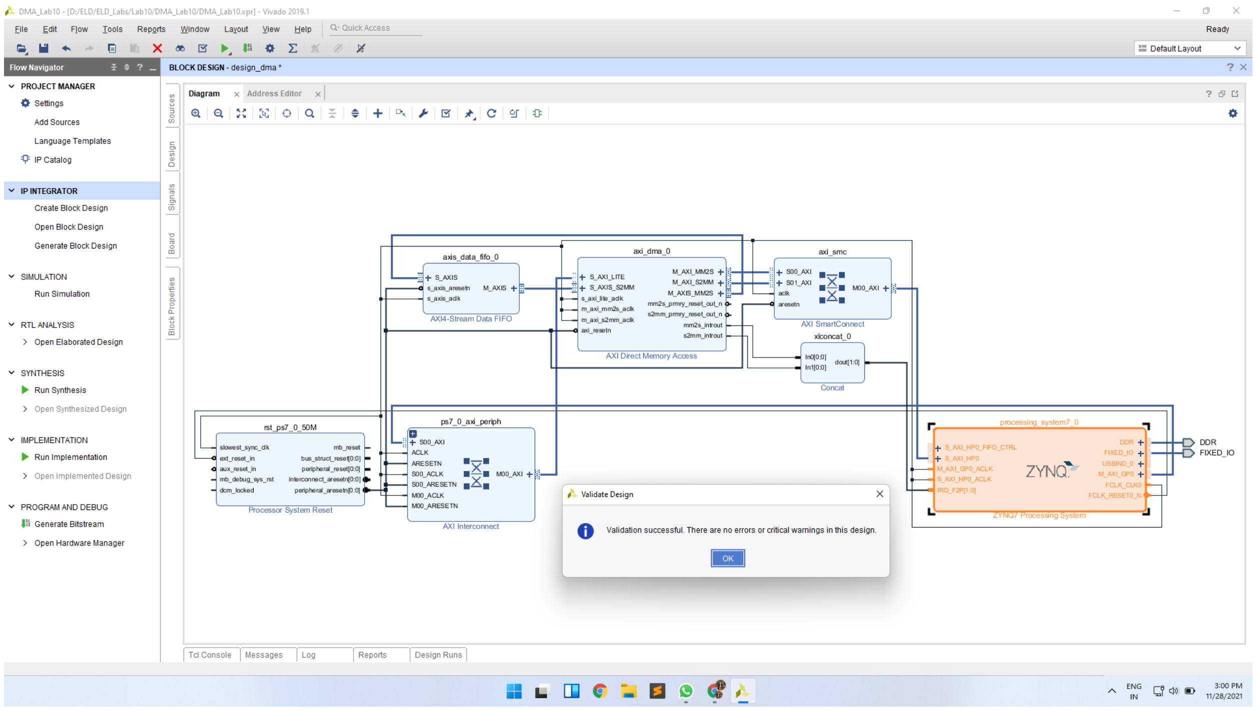
There were four critical warning messages while validating this design.

Messages

[PSU-1] Parameter : PCW_UIPARAM_DDR_DQS_TO_CLK_DELAY_0 has negative value -0.050 . PS DDR interfaces might fail when entering negative DQS skew values.

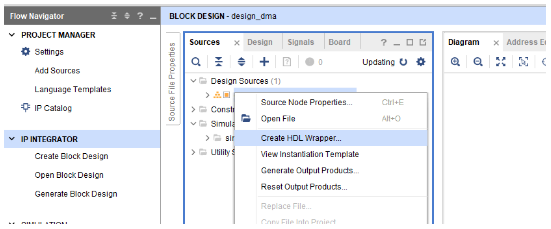
[PSU-2] Parameter : PCW_UIPARAM_DDR_DQS_TO_CLK_DELAY_1 has negative value -0.044 . PS DDR interfaces might fail when entering negative DQS skew values.

[PSU-3] Parameter : PCW_UIPARAM_DDR_DQS_TO_CLK_DELAY_2 has negative value -0.035 . PS DDR interfaces might fail when entering negative DQS skew values.

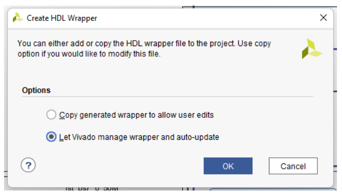

Validate Design

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

18. 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.



$\longrightarrow$


Create HDL Wrapper

You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.

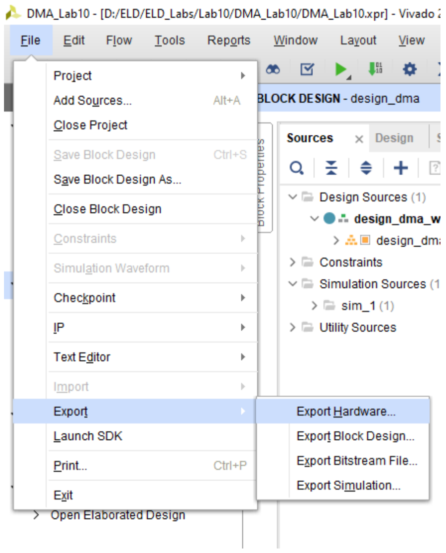
Options

( ) Copy generated wrapper to allow user edits

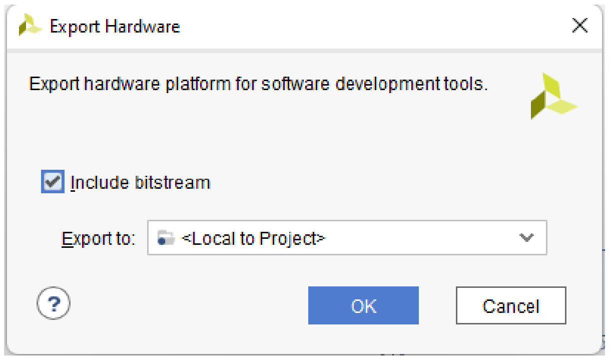
(●) Let Vivado manage wrapper and auto-update

19. Now, click on Generate Bitstream and then click on Ok.

20. Now, click on File → Export Hardware. Check the include bitstream box.



→

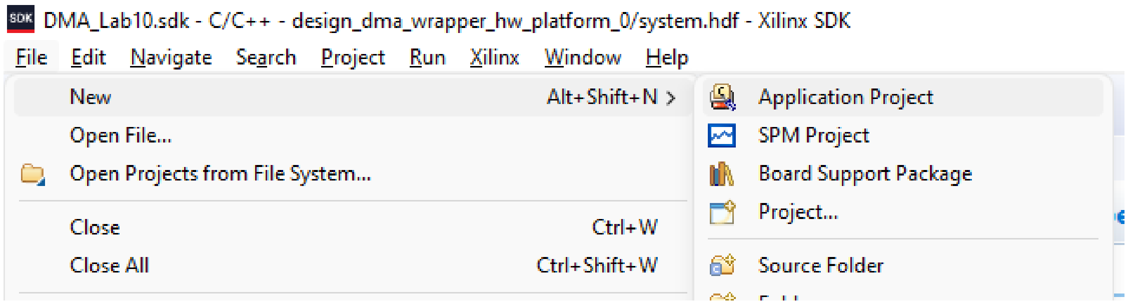

Export Hardware

Export hardware platform for software development tools.

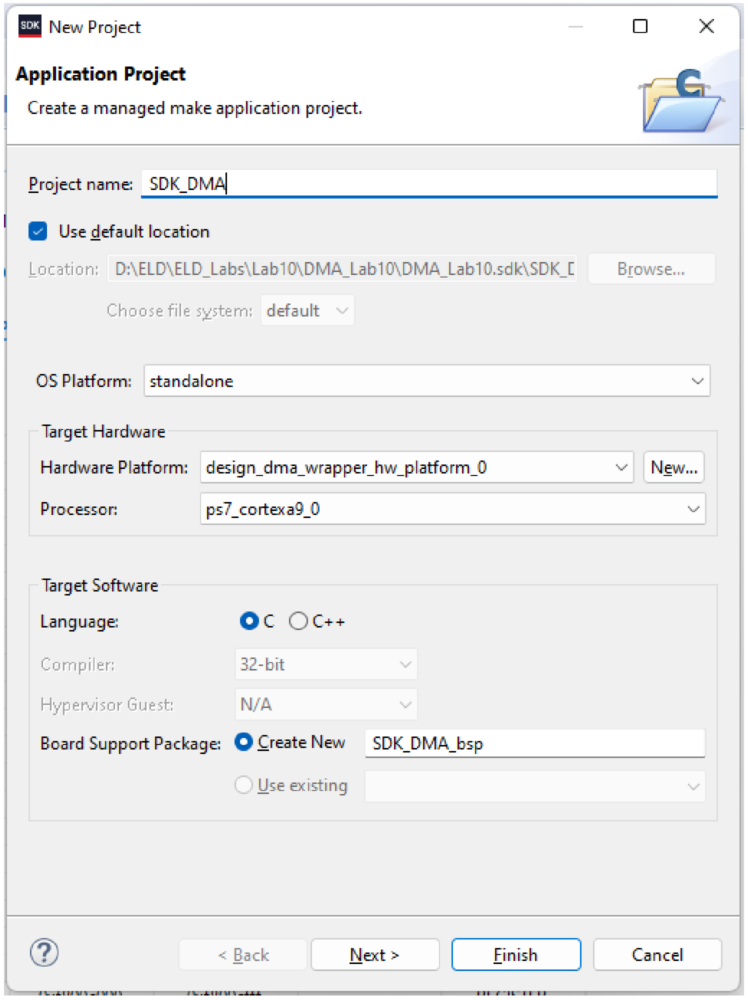
☑ Include bitstream

Export to: <Local to Project>

21. Click on File → Launch SDK. Click on Ok.
22. In the SDK window, select File → New → New Application Project.


DMA_Lab10.sdk - C/C++ - design_dma_wrapper_hw_platform_0/system.hdf - Xilinx SDK

23. Create a new project with the following settings.



Application Project

Create a managed make application project.

Project name: SDK_DMA

[✓] Use default location

Location: D:\ELD\ELD_Labs\Lab10\DMA_Lab10\DMA_Lab10.sdk\SDK_D

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: design_dma_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] SDK_DMA_bsp

Use existing

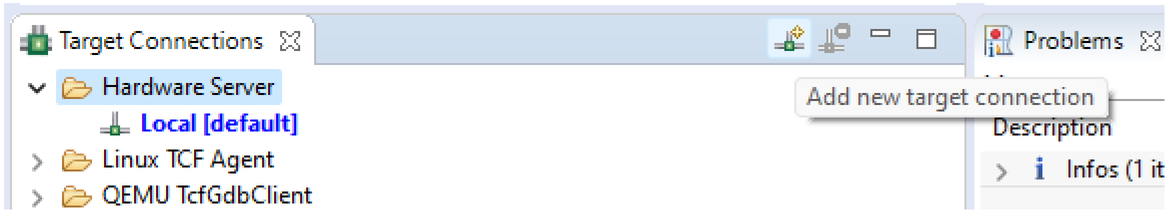
24. Click on Next. Select Hello World Application. Click on finish.
25. Open the helloworld.c file from SDK_DMA/src folder.
26. Add the following code to this file.

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

#define FIFO_DEPTH                128             //number of data transfers

XAxiDma AxiDma;      //creating an instance of DMA block
```

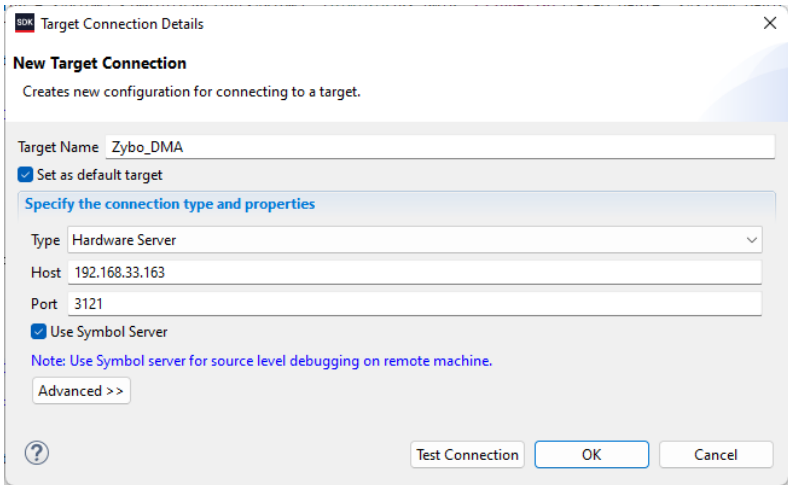
```
int init_DMA()        //this function initializes the DMA instance with required
configurations
{
      XAxiDma_Config *CfgPtr;
      int status;

      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()
{
	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;
	}

	int TX_PNTR[FIFO_DEPTH];
	int RX_PNTR[FIFO_DEPTH];

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

	for (int i = 0 ; i < FIFO_DEPTH ; i++)	//loop to initialize the data to be
transmitted
		TX_PNTR[i] = 2*i;

	Xil_DCacheFlushRange((UINTPTR)TX_PNTR, sizeof(int)*FIFO_DEPTH);		//cache
flush to write back the data in the DDR memory
	Xil_DCacheFlushRange((UINTPTR)RX_PNTR, sizeof(int)*FIFO_DEPTH);

	//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;
	status_transfer = XAxiDma_SimpleTransfer(&AxiDma, (UINTPTR)TX_PNTR,
sizeof(int)*FIFO_DEPTH, XAXIDMA_DMA_TO_DEVICE);
		//transferring data from DDR to FIFO.
	if (status_transfer != XST_SUCCESS)
	{
		xil_printf("Writing data to FIFO 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)RX_PNTR,
sizeof(int)*FIFO_DEPTH, XAXIDMA_DEVICE_TO_DMA);
	//receiving data from FIFO in DDR.
	if (status_transfer != XST_SUCCESS)
	{
		xil_printf("Reading data from FIFO 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));


	int status = checkIdle(XPAR_AXI_DMA_0_BASEADDR, 0x4);		//DMA to device
	while(status != 2)
	{
		status = checkIdle(XPAR_AXI_DMA_0_BASEADDR, 0x4);
	}

	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 waiting\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 data transmitted to FIFO and read from FIFO via DMA
\r\n");
     int j = 0;
     for (j = 0 ; j < FIFO_DEPTH ; j++)
     {
          if (TX_PNTR[j] != RX_PNTR[j])
          {
               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 , TX_PNTR[j], RX_PNTR[j]);
     else
          xil_printf("\nDMA ran successfully!! :)");

     return XST_SUCCESS;
}
```

27. Add the hardware server.
    a. Add a new target connection.



b. Add the connection details as follows. (Use the Host IP provided for your group).



Target Connection Details

New Target Connection

Creates new configuration for connecting to a target.

Target Name: Zybo_DMA

☑ Set as default target

Specify the connection type and properties

Type: Hardware Server

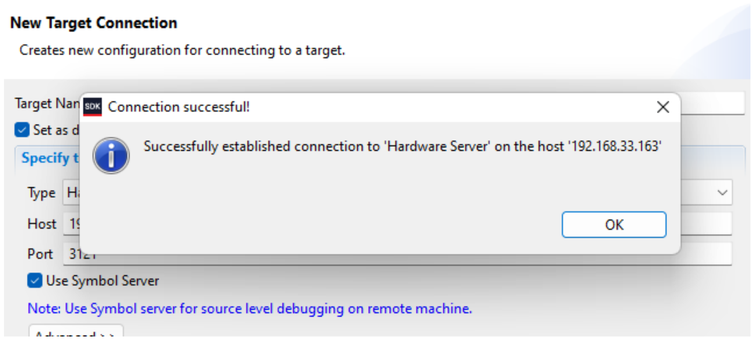
Host: 192.168.33.163

Port: 3121

☑ Use Symbol Server

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

c. Test the connection for a successful connection.



New Target Connection

Creates new configuration for connecting to a target.

Connection successful!

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

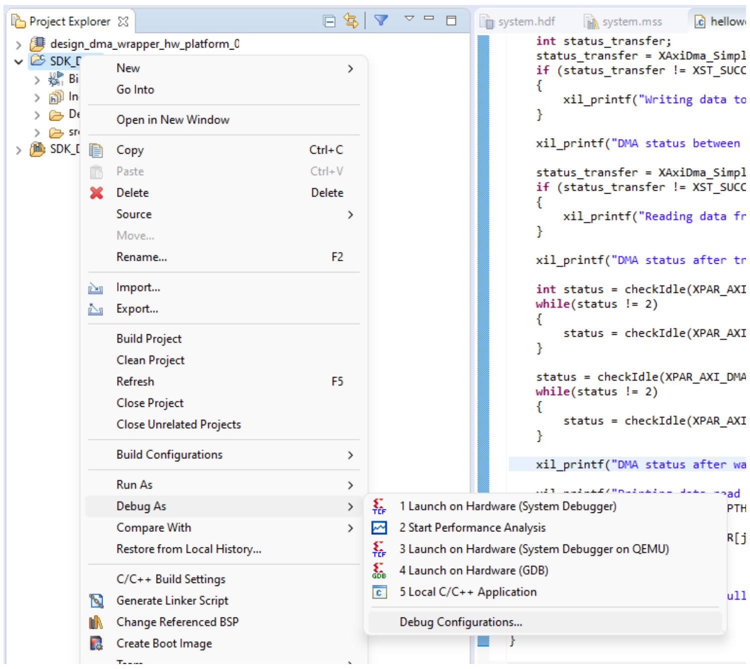
Note: Use Symbol server for source level debugging on remote machine.

d. Click on Ok.

28. Go to system.mss file (in the SDK_DMA_bsp folder). Select Modify BSP's settings.

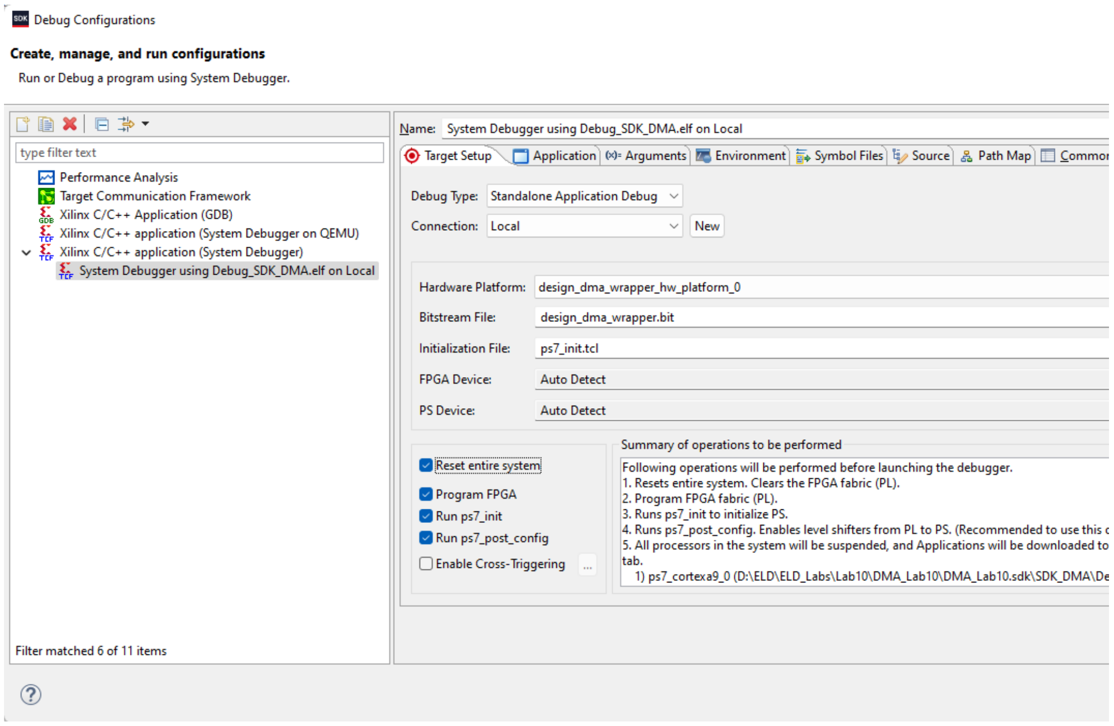
a. Change the standard input/output to ps7_coresight_comp_0 (to enable JTAG interface instead to default UART). Click on Ok.

29. Right-click the project SDK_DMA → Debug as → Debug Configurations.



```
int status_transfer;
status_transfer = XAxiDma_Simpl
if (status_transfer != XST_SUCC
{
    xil_printf("Writing data to
}

xil_printf("DMA status between

status_transfer = XAxiDma_Simpl
if (status_transfer != XST_SUCC
{
    xil_printf("Reading data fr
}

xil_printf("DMA status after tr

int status = checkIdle(XPAR_AXI
while(status != 2)
{
    status = checkIdle(XPAR_AXI
}

status = checkIdle(XPAR_AXI_DMA
while(status != 2)
{
    status = checkIdle(XPAR_AXI
}

xil_printf("DMA status after wa
```

30. Double click the Xilinx C/C++ Application (System Debugger) to define the debug configuration for the current run. Check reset entire system and program FPGA. Leave other settings unchanged.


Debug Configurations
Create, manage, and run configurations
Run or Debug a program using System Debugger.
type filter text
Performance Analysis
Target Communication Framework
Xilinx C/C++ Application (GDB)
Xilinx C/C++ application (System Debugger on QEMU)
Xilinx C/C++ application (System Debugger)
System Debugger using Debug_SDK_DMA.elf on Local
Filter matched 6 of 11 items
Target Setup
Application
Arguments
Environment
Symbol Files
Source
Path Map
Common
New

Name: System Debugger using Debug_SDK_DMA.elf on Local

Debug Type: Standalone Application Debug
Connection: Local

Hardware Platform: design_dma_wrapper_hw_platform_0
Bitstream File: design_dma_wrapper.bit
Initialization File: ps7_init.tcl
FPGA Device: Auto Detect
PS Device: Auto Detect

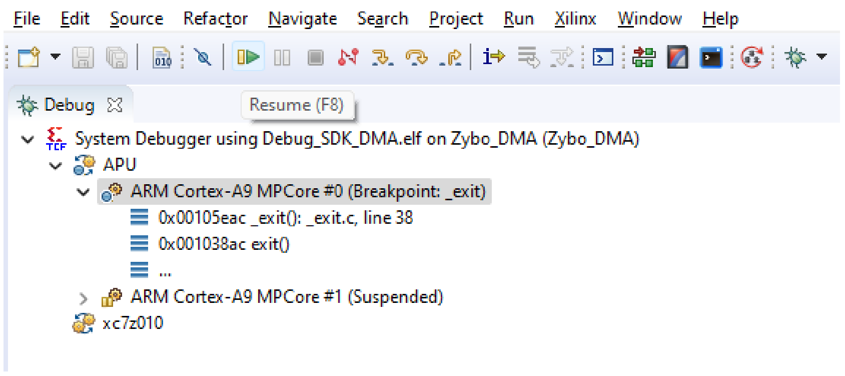
- [x] Reset entire system
- [x] Program FPGA
- [x] Run ps7_init
- [x] Run ps7_post_config
- [ ] Enable Cross-Triggering

Summary of operations to be performed

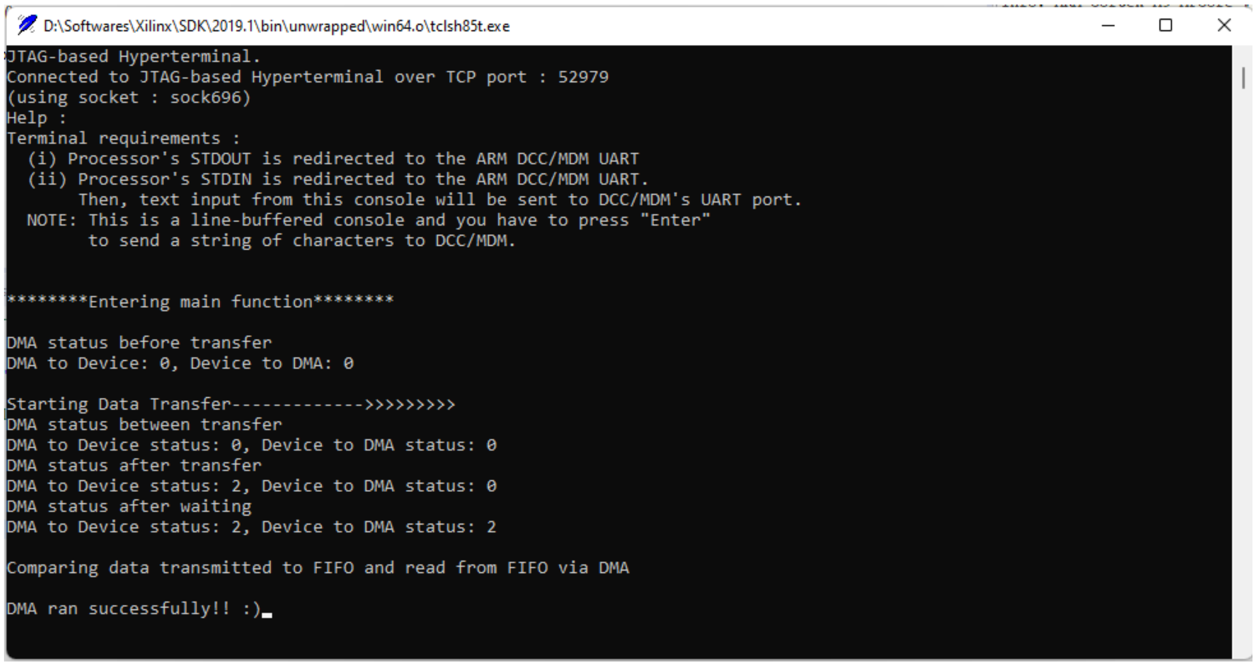
Following operations will be performed before launching the debugger.
1. Resets entire system. Clears the FPGA fabric (PL).
2. Program FPGA fabric (PL).
3. Runs ps7_init to initialize PS.
4. Runs ps7_post_config. Enables level shifters from PL to PS. (Recommended to use this c
5. All processors in the system will be suspended, and Applications will be downloaded to
tab.
1) ps7_cortexa9_0 (D:\ELD\ELD_Labs\Lab10\DMA_Lab10\DMA_Lab10.sdk\SDK_DMA\De

31. Click on Apply → Debug.
32. Click Ok on the dialogue box that opens.

33. Once the system debugger is launched, type **jtagterminal** in XSCT Console. This will open the terminal for displaying output messages.
34. Click on resume button.


File Edit Source Refactor Navigate Search Project Run Xilinx Window Help
Debug
Resume (F8)
System Debugger using Debug_SDK_DMA.elf on Zybo_DMA (Zybo_DMA)
APU
ARM Cortex-A9 MPCore #0 (Breakpoint: _exit)
0x00105eac _exit(): _exit.c, line 38
0x001038ac exit()
...
ARM Cortex-A9 MPCore #1 (Suspended)
xc7z010

35. Switch to the terminal window. The output should be as follows.


D:\Softwares\Xilinx\SDK\2019.1\bin\unwrapped\win64.o\tclsh85t.exe

```
JTAG-based Hyperterminal.
Connected to JTAG-based Hyperterminal over TCP port : 52979
(using socket : sock696)
Help :
Terminal requirements :
  (i) Processor's STDOUT is redirected to the ARM DCC/MDM UART
  (ii) Processor's STDIN is redirected to the ARM DCC/MDM UART.
       Then, text input from this console will be sent to DCC/MDM's UART port.
  NOTE: This is a line-buffered console and you have to press "Enter"
        to send a string of characters to DCC/MDM.


********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: 2, Device to DMA status: 0
DMA status after waiting
DMA to Device status: 2, Device to DMA status: 2

Comparing data transmitted to FIFO and read from FIFO via DMA

DMA ran successfully!! :)_
```

Lab Homework: There are 2 parts of lab homework this time. Each carries 1.5 marks

1. Complete the lab
2. Add ILA IP to show AXI transactions