Praktikum/Forschungpraktikum Embedded System Design
Chair for Processor Design, cfaed, TUD
Winter Semester, 2019-2020

Lab 2 – Designing Hardware Accelerators with Streaming Interface for High-speed and Low-latency Continuous Data Transaction

In this lab, the students will be guided through two tutorials to know how to work with the AMBA AXI-Stream interface and how to use ChipScope to debug on the FPGA. There are two homework assignments after the lab which focus more on the performance analysis between using the AXI4-Lite and AXI-Stream interface. Besides, the students need to perform the analysis at the system level where multiple accelerators are trying to access the memory at the same time.

Introduction

AMBA AXI (Advanced eXtensible Interface) is the industry-standard communication interface proposed by ARM. It is used in almost every recent System-on-Chip design. The purpose of having a standard interface is that SoC design companies can easily integrate the IPs from the IP vendors into their system without knowing the internal implementations. IP vendors are the ones who are exclusively designing and selling IPs (such as video encoder, decoder, encryption, decryption, etc.). It is similar to plugging in the USB-compatible devices into the computers. There are other similar interfaces such as IBM PLB (Processor Local Bus, quite popular 15-20 years ago), Wishbone (an open-sourced interface used mainly by the designs published on OpenCores).

The latest AXI version is 5, however, only version 4 is supported by Xilinx Vivado. Official specifications from ARM can be found here. The ARM specification is very sophisticated with many features. It is up to the provider to support all of the features or just a subset of them. The document provided by Xilinx is found here. It discusses which features are supported by Xilinx IP and the corresponding standard IPs.

AXI-Stream Interface

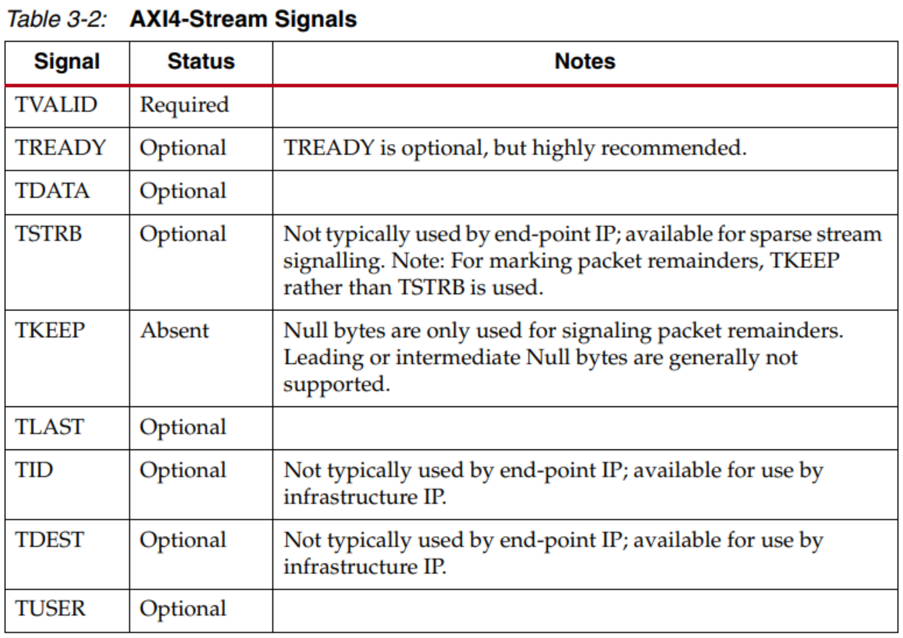


Table 3-2: **AXI4-Stream Signals**

Signal	Status	Notes
TVALID	Required	
TREADY	Optional	TREADY is optional, but highly recommended.
TDATA	Optional	
TSTRB	Optional	Not typically used by end-point IP; available for sparse stream signalling. Note: For marking packet remainders, TKEEP rather than TSTRB is used.
TKEEP	Absent	Null bytes are only used for signaling packet remainders. Leading or intermediate Null bytes are generally not supported.
TLAST	Optional	
TID	Optional	Not typically used by end-point IP; available for use by infrastructure IP.
TDEST	Optional	Not typically used by end-point IP; available for use by infrastructure IP.
TUSER	Optional	

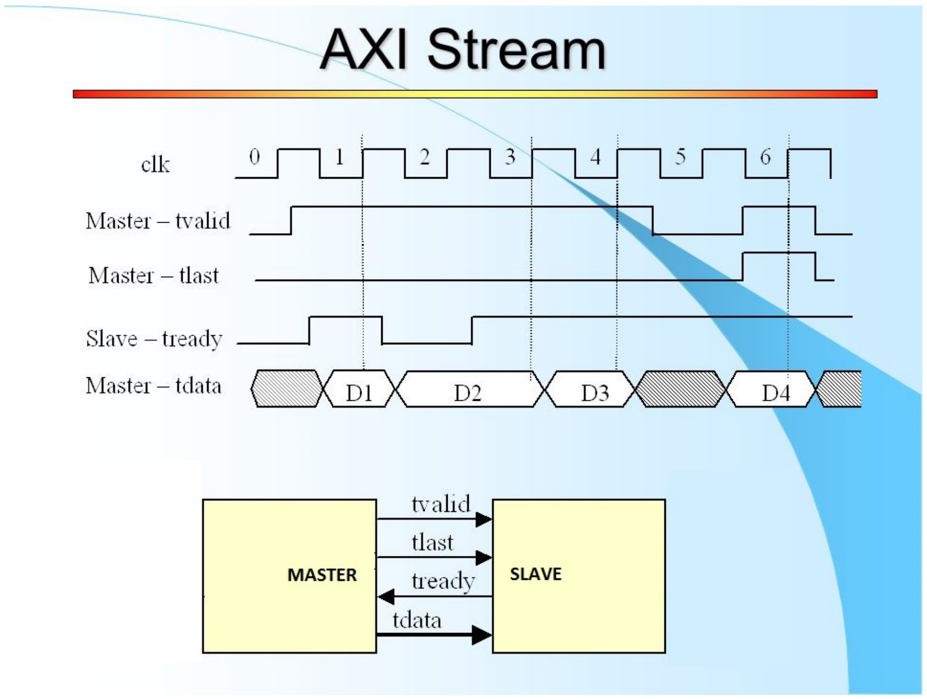
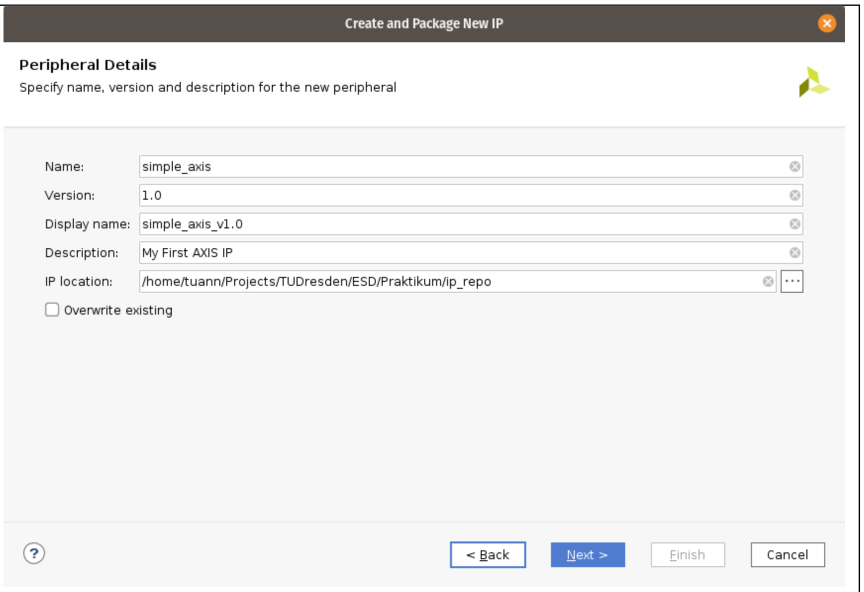


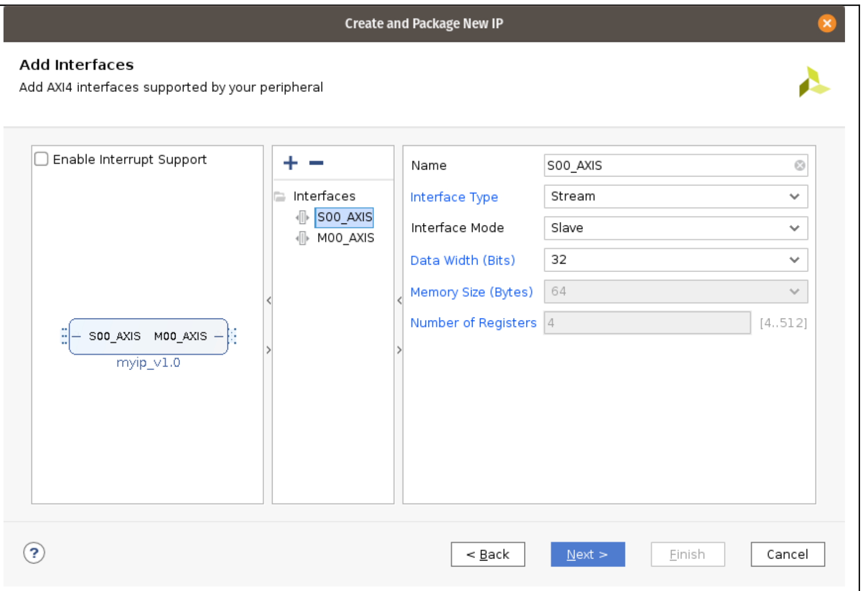
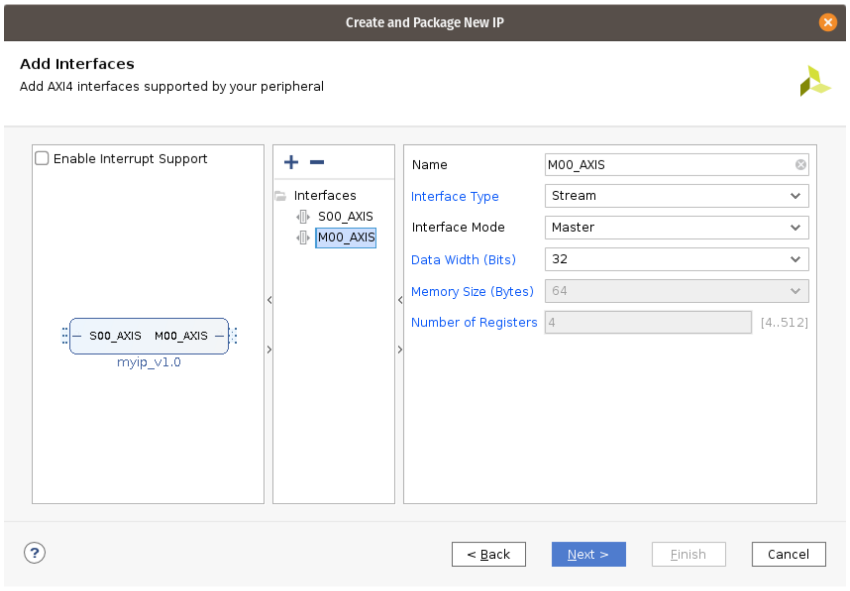
Figure 1: The AXIS Communication between Master and Slave (http://fpgasite.blogspot.com/2017/07/xilinx-axi-stream-tutorial-part-1.html)

Lab 2 – A

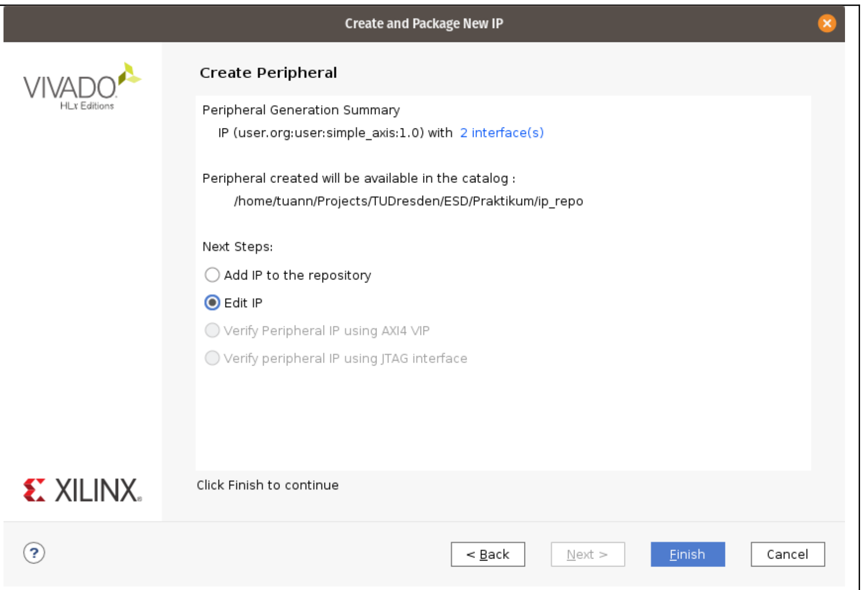
Create a new project. When a new project is opened, create a new IP and name it similar to the picture.	

Create and Package New IP
Peripheral Details
Specify name, version and description for the new peripheral
< Back
Next >
Finish
Cancel

Name: simple_axis
Version: 1.0
Display name: simple_axis_v1.0
Description: My First AXIS IP
IP location: /home/tuann/Projects/TUDresden/ESD/Praktikum/ip_repo
☐ Overwrite existing

Create 2 AXIS interfaces as shown: S00_AXIS (slave) and M00_AXIS (master). Then click "Next"	 Create and Package New IP **Add Interfaces** Add AXI4 interfaces supported by your peripheral ☐ Enable Interrupt Support Interfaces: S00_AXIS, M00_AXIS Name: S00_AXIS Interface Type: Stream Interface Mode: Slave Data Width (Bits): 32 Memory Size (Bytes): 64 Number of Registers: 4 [4..512] S00_AXIS M00_AXIS myip_v1.0 < Back Next > Finish Cancel  Create and Package New IP **Add Interfaces** Add AXI4 interfaces supported by your peripheral ☐ Enable Interrupt Support Interfaces: S00_AXIS, M00_AXIS Name: M00_AXIS Interface Type: Stream Interface Mode: Master Data Width (Bits): 32 Memory Size (Bytes): 64 Number of Registers: 4 [4..512] S00_AXIS M00_AXIS myip_v1.0 < Back Next > Finish Cancel

Choose “Edit IP” then Finish to start editing the IP



Create and Package New IP

Create Peripheral

Peripheral Generation Summary
IP (user.org:user:simple_axis:1.0) with 2 interface(s)

Peripheral created will be available in the catalog :
/home/tuann/Projects/TUDresden/ESD/Praktikum/ip_repo

Next Steps:

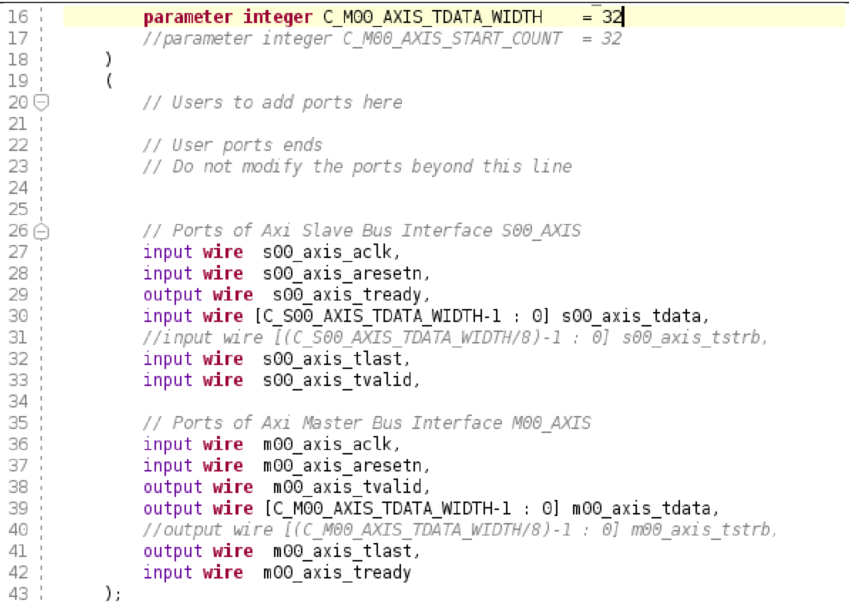
- Add IP to the repository
- [✓ Edit IP]
- Verify Peripheral IP using AXI4 VIP
- Verify peripheral IP using JTAG interface

Click Finish to continue

In the file simple_axis_v1_0.v, comment out the line 17, 31 and 40 as shown. We are not using the AXIS_TSTRB signals (what is it?).

In the default template, the AXIS Master port will wait for several clock cycles before sending some data (configured by the C_M00_AXIS_START_COUNT parameter).

Here, we are going to receive data from ARM (at the AXIS Slave interface) then send them back by using the AXIS Master interface. The data will be buffered in a memory block between these two interfaces (Slave and Master).

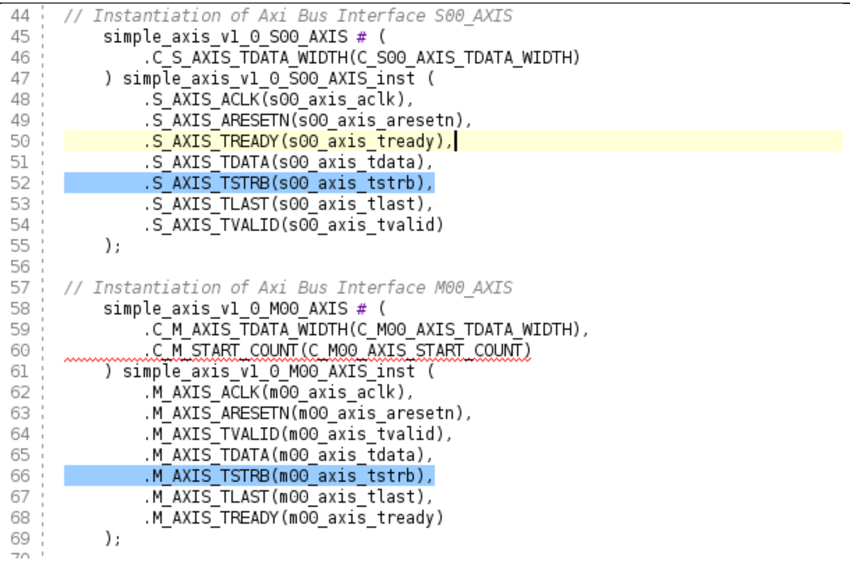


```
16          parameter integer C_M00_AXIS_TDATA_WIDTH    = 32
17          //parameter integer C_M00_AXIS_START_COUNT   = 32
18      )
19      (
20          // Users to add ports here
21
22          // User ports ends
23          // Do not modify the ports beyond this line
24
25
26          // Ports of Axi Slave Bus Interface S00_AXIS
27          input wire  s00_axis_aclk,
28          input wire  s00_axis_aresetn,
29          output wire  s00_axis_tready,
30          input wire [C_S00_AXIS_TDATA_WIDTH-1 : 0] s00_axis_tdata,
31          //input wire [(C_S00_AXIS_TDATA_WIDTH/8)-1 : 0] s00_axis_tstrb,
32          input wire  s00_axis_tlast,
33          input wire  s00_axis_tvalid,
34
35          // Ports of Axi Master Bus Interface M00_AXIS
36          input wire  m00_axis_aclk,
37          input wire  m00_axis_aresetn,
38          output wire  m00_axis_tvalid,
39          output wire [C_M00_AXIS_TDATA_WIDTH-1 : 0] m00_axis_tdata,
40          //output wire [(C_M00_AXIS_TDATA_WIDTH/8)-1 : 0] m00_axis_tstrb,
41          output wire  m00_axis_tlast,
42          input wire  m00_axis_tready
43      );
```

You need to update the module instantiations for AXIS Slave and AXIS Master accordingly to the changes in the previous step.

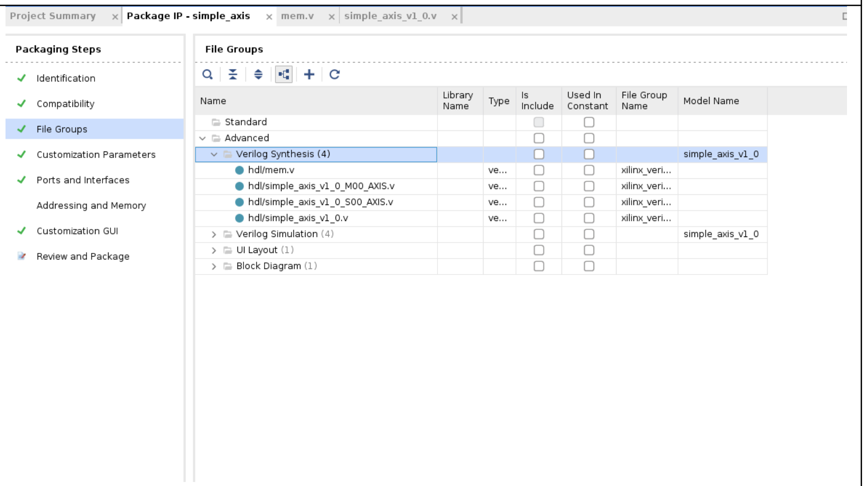
Similarly, you need to update the port descriptions and related signal assignments for those two modules as well.

You can use the code I prepared here to replace the Xilinx generated files: https://www.dropbox.com/sh/dzlpb1nw1ys8bng/AACiYEyg1iH-C2ddpV3cNJIPa?dl=0

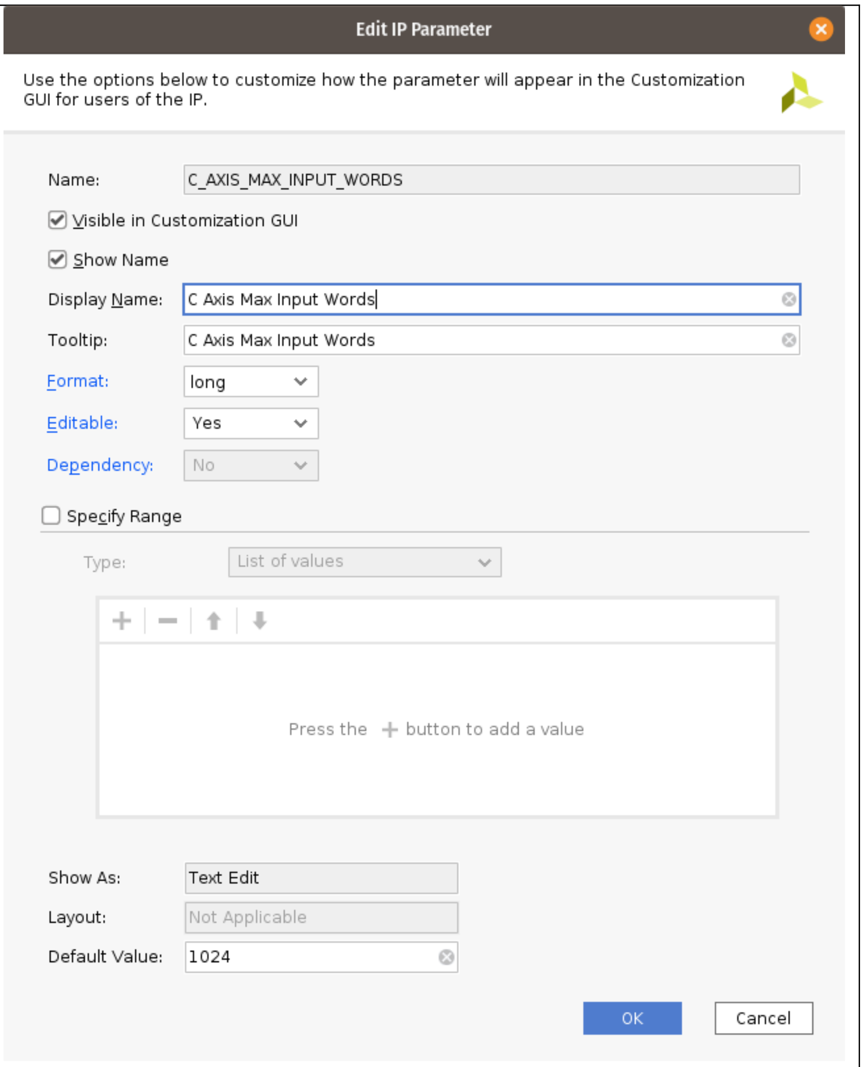
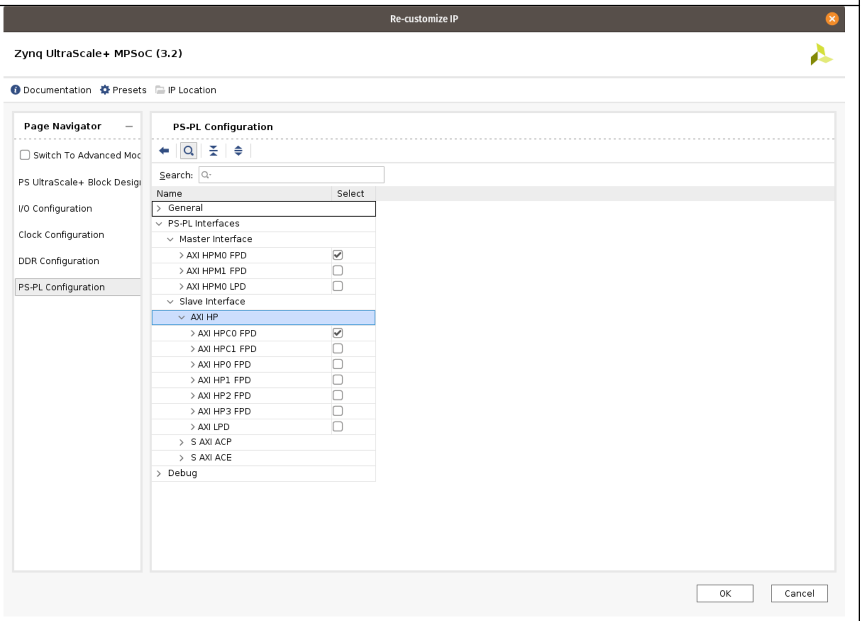


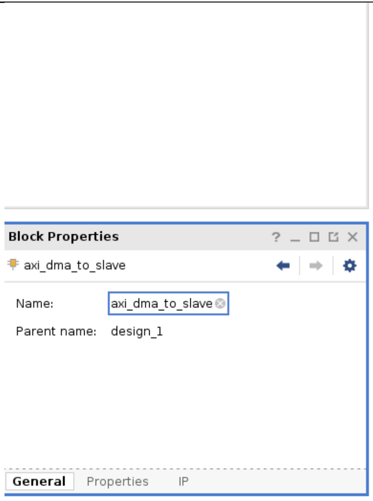
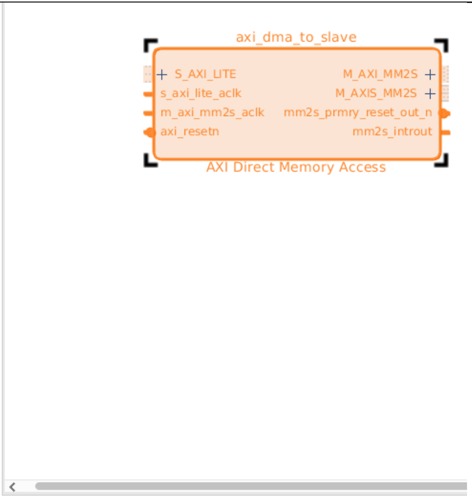
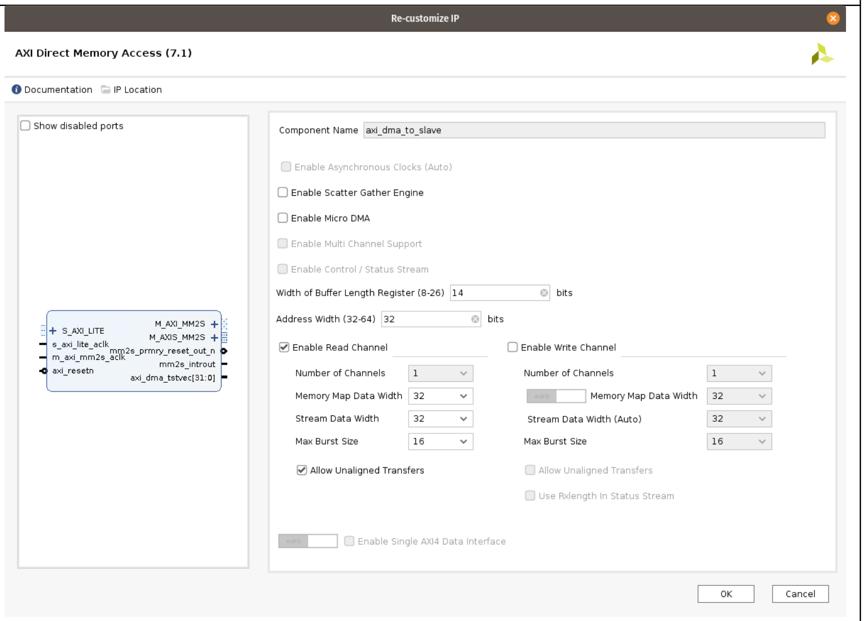
```verilog
// Instantiation of Axi Bus Interface S00_AXIS
    simple_axis_v1_0_S00_AXIS # (
        .C_S_AXIS_TDATA_WIDTH(C_S00_AXIS_TDATA_WIDTH)
    ) simple_axis_v1_0_S00_AXIS_inst (
        .S_AXIS_ACLK(s00_axis_aclk),
        .S_AXIS_ARESETN(s00_axis_aresetn),
        .S_AXIS_TREADY(s00_axis_tready),
        .S_AXIS_TDATA(s00_axis_tdata),
        .S_AXIS_TSTRB(s00_axis_tstrb),
        .S_AXIS_TLAST(s00_axis_tlast),
        .S_AXIS_TVALID(s00_axis_tvalid)
    );

// Instantiation of Axi Bus Interface M00_AXIS
    simple_axis_v1_0_M00_AXIS # (
        .C_M_AXIS_TDATA_WIDTH(C_M00_AXIS_TDATA_WIDTH),
        .C_M_START_COUNT(C_M00_AXIS_START_COUNT)
    ) simple_axis_v1_0_M00_AXIS_inst (
        .M_AXIS_ACLK(m00_axis_aclk),
        .M_AXIS_ARESETN(m00_axis_aresetn),
        .M_AXIS_TVALID(m00_axis_tvalid),
        .M_AXIS_TDATA(m00_axis_tdata),
        .M_AXIS_TSTRB(m00_axis_tstrb),
        .M_AXIS_TLAST(m00_axis_tlast),
        .M_AXIS_TREADY(m00_axis_tready)
    );
```

After that, merge all changes in the Package IP tab and make sure that you have 4 source files as shown.

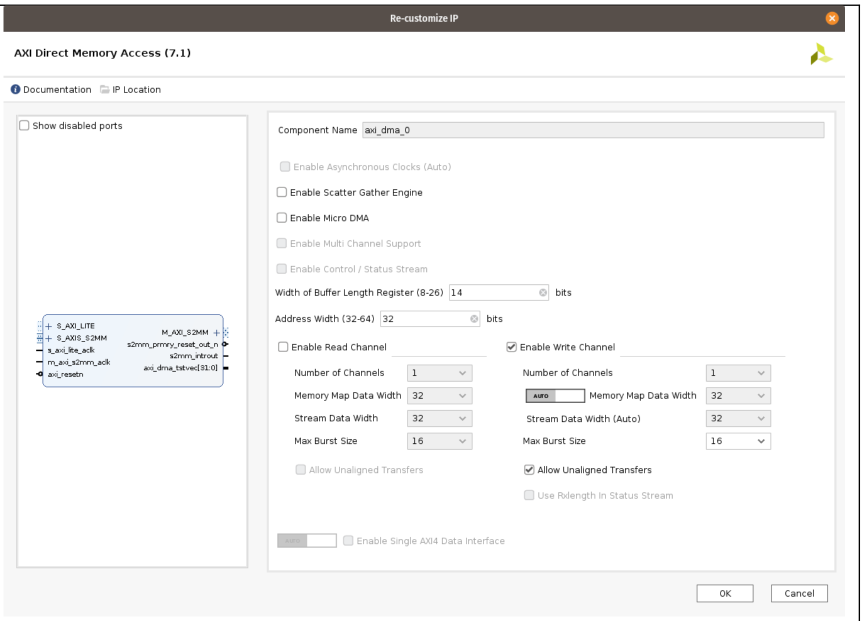


Name	Library Name	Type	Is Include	Used In Constant	File Group Name	Model Name
Standard			☐	☐		
Advanced			☐	☐		
Verilog Synthesis (4)			☐	☐		simple_axis_v1_0
hdl/mem.v		ve...	☐	☐	xilinx_veri...	
hdl/simple_axis_v1_0_M00_AXIS.v		ve...	☐	☐	xilinx_veri...	
hdl/simple_axis_v1_0_S00_AXIS.v		ve...	☐	☐	xilinx_veri...	
hdl/simple_axis_v1_0.v		ve...	☐	☐	xilinx_veri...	
Verilog Simulation (4)			☐	☐		simple_axis_v1_0
UI Layout (1)			☐	☐		
Block Diagram (1)			☐	☐		

In the Customization Parameters of the Package IP, right click on the available parameters, choose edit → check the Visible in Customization GUI to allow changes from the Block Design. Then click Re-Package IP to save the changes and close the Edit IP project.	 **Edit IP Parameter** Use the options below to customize how the parameter will appear in the Customization GUI for users of the IP. Name: C_AXIS_MAX_INPUT_WORDS ☑ Visible in Customization GUI ☑ Show Name Display Name: C Axis Max Input Words Tooltip: C Axis Max Input Words Format: long Editable: Yes Dependency: No ☐ Specify Range Type: List of values Press the + button to add a value Show As: Text Edit Layout: Not Applicable Default Value: 1024 OK Cancel
Go back to the original project, create a block design, add the Zynq then Block Automation. Then configure the Zynq block as shown.	 **Re-customize IP** Zynq UltraScale+ MPSoC (3.2) Documentation Presets IP Location Page Navigator: ☐ Switch To Advanced Mod; PS UltraScale+ Block Design; I/O Configuration; Clock Configuration; DDR Configuration; PS-PL Configuration PS-PL Configuration Search: Name / Select › General ⌄ PS-PL Interfaces ⌄ Master Interface › AXI HPM0 FPD ✓ › AXI HPM1 FPD ☐ › AXI HPM0 LPD ☐ ⌄ Slave Interface ⌄ AXI HP › AXI HPC0 FPD ✓ › AXI HPC1 FPD ☐ › AXI HP0 FPD ☐ › AXI HP1 FPD ☐ › AXI HP2 FPD ☐ › AXI HP3 FPD ☐ › AXI LPD ☐ › S AXI ACP › S AXI ACE › Debug OK Cancel

Add the AXI Direct Memory Access IP, rename it inside the Block Properties.	 Block Properties axi_dma_to_slave Name: axi_dma_to_slave Parent name: design_1  axi_dma_to_slave S_AXI_LITE s_axi_lite_aclk m_axi_mm2s_aclk axi_resetn M_AXI_MM2S M_AXIS_MM2S mm2s_prmry_reset_out_n mm2s_introut AXI Direct Memory Access
Double click the DMA IP to open the Re-customize IP, configure it as shown. We use this DMA to data to the IP by reading from the DDR. Therefore, we enable only the Read Channel.	 Re-customize IP AXI Direct Memory Access (7.1) Documentation IP Location Show disabled ports S_AXI_LITE s_axi_lite_aclk m_axi_mm2s_aclk axi_resetn M_AXI_MM2S M_AXIS_MM2S mm2s_prmry_reset_out_n mm2s_introut axi_dma_tstvec[31:0] Component Name axi_dma_to_slave 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 32 Stream Data Width 32 Max Burst Size 16 ✓ Allow Unaligned Transfers Enable Write Channel Number of Channels 1 AUTO Memory Map Data Width 32 Stream Data Width (Auto) 32 Max Burst Size 16 Allow Unaligned Transfers Use Rxlength In Status Stream AUTO Enable Single AXI4 Data Interface

Similarly for the DMA_FROM_MASTER, we enable the write channel to write the output from the IP to the DDR.



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

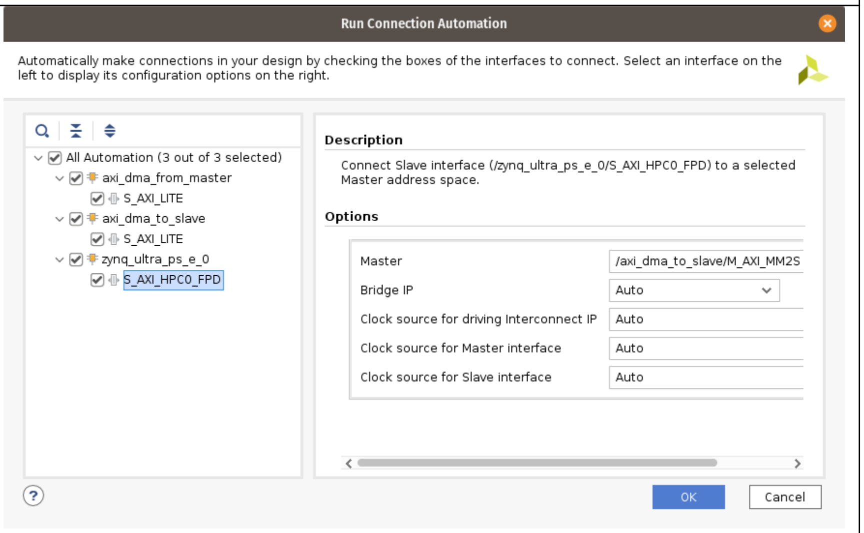
Address Width (32-64): 32 bits

☐ Enable Read Channel		✓ Enable Write Channel	
Number of Channels	1	Number of Channels	1
Memory Map Data Width	32	AUTO Memory Map Data Width	32
Stream Data Width	32	Stream Data Width (Auto)	32
Max Burst Size	16	Max Burst Size	16
☐ Allow Unaligned Transfers		✓ Allow Unaligned Transfers	
		☐ Use Rxlength In Status Stream	

AUTO ☐ Enable Single AXI4 Data Interface

OK   Cancel

Run connection automation to connect the ports.



Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

- ✓ All Automation (3 out of 3 selected)
  - ✓ axi_dma_from_master
    - ✓ S_AXI_LITE
  - ✓ axi_dma_to_slave
    - ✓ S_AXI_LITE
  - ✓ zynq_ultra_ps_e_0
    - ✓ S_AXI_HPC0_FPD

Description

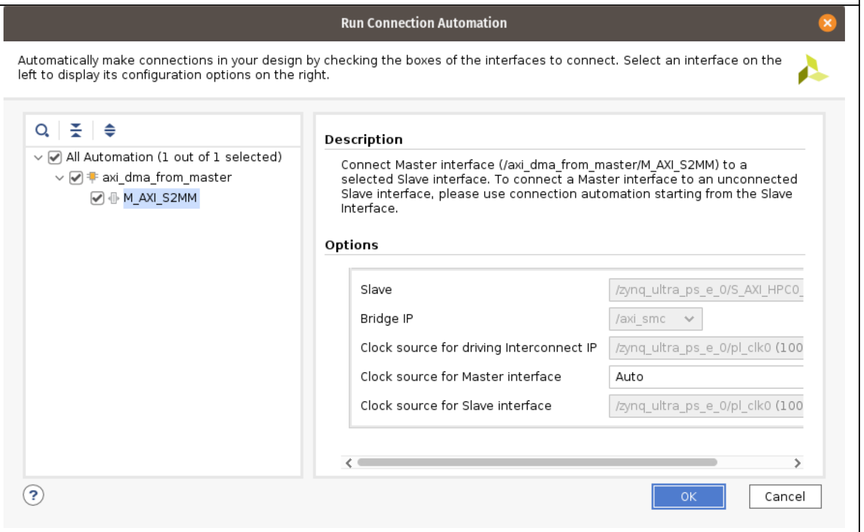
Connect Slave interface (/zynq_ultra_ps_e_0/S_AXI_HPC0_FPD) to a selected Master address space.

Options

Master	/axi_dma_to_slave/M_AXI_MM2S
Bridge IP	Auto
Clock source for driving Interconnect IP	Auto
Clock source for Master interface	Auto
Clock source for Slave interface	Auto

OK   Cancel

Again, run connection automation to connect the ports.



Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

- ✓ All Automation (1 out of 1 selected)
  - ✓ axi_dma_from_master
    - ✓ M_AXI_S2MM

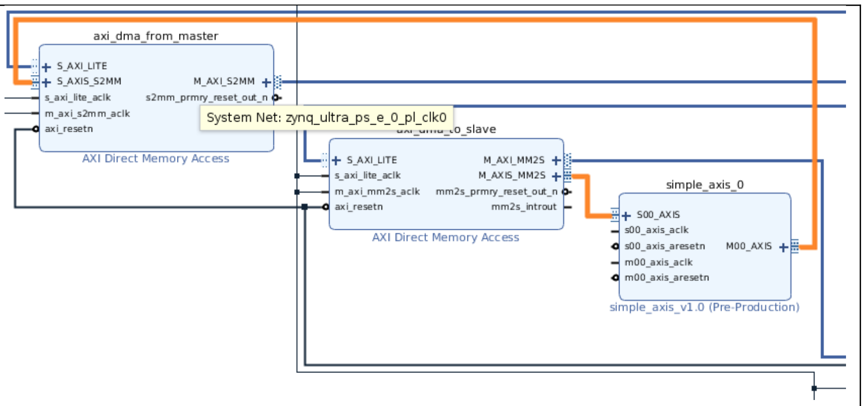
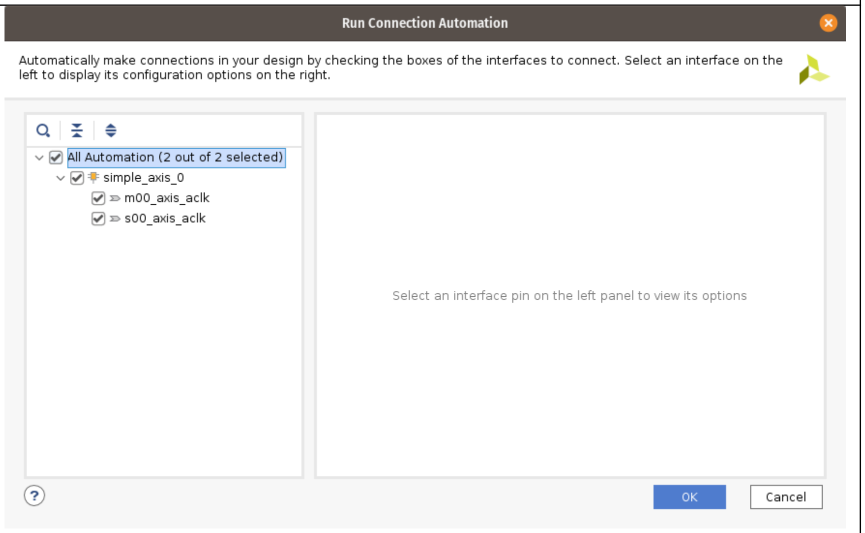
Description

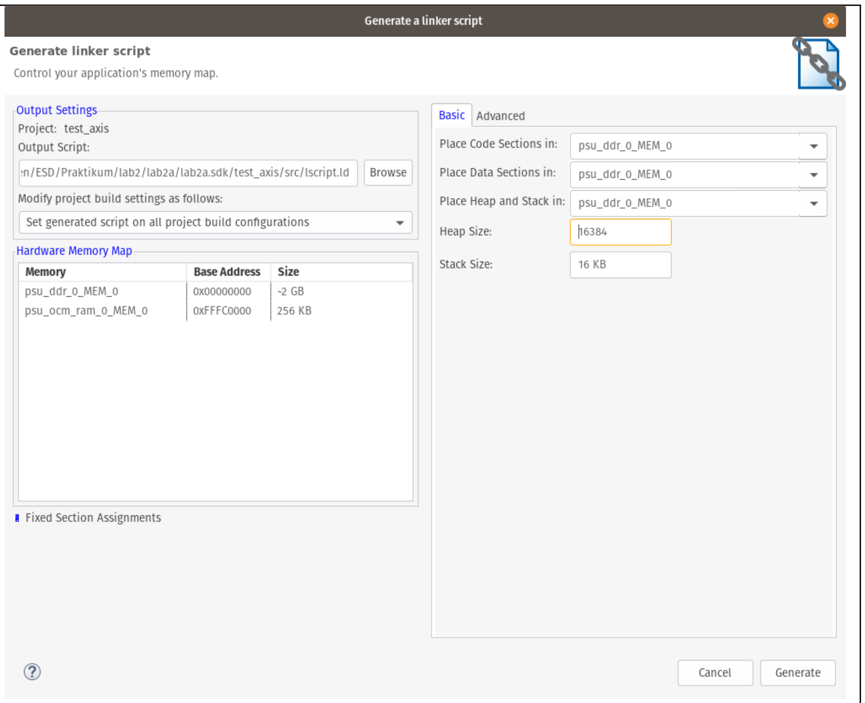
Connect Master interface (/axi_dma_from_master/M_AXI_S2MM) to a selected Slave interface. To connect a Master interface to an unconnected Slave interface, please use connection automation starting from the Slave Interface.

Options

Slave	/zynq_ultra_ps_e_0/S_AXI_HPC0_
Bridge IP	/axi_smc
Clock source for driving Interconnect IP	/zynq_ultra_ps_e_0/pl_clk0 (100
Clock source for Master interface	Auto
Clock source for Slave interface	/zynq_ultra_ps_e_0/pl_clk0 (100

OK   Cancel

Now, you have to connect the ports from the simple_axis_0 block to the corresponding DMAs.	 axi_dma_from_master S_AXI_LITE S_AXIS_S2MM M_AXI_S2MM s_axi_lite_aclk s2mm_prmry_reset_out_n m_axi_s2mm_aclk axi_resetn AXI Direct Memory Access System Net: zynq_ultra_ps_e_0_pl_clk0 axi_dma_to_slave S_AXI_LITE M_AXI_MM2S s_axi_lite_aclk M_AXIS_MM2S m_axi_mm2s_aclk mm2s_prmry_reset_out_n axi_resetn mm2s_introut AXI Direct Memory Access simple_axis_0 S00_AXIS s00_axis_aclk s00_axis_aresetn M00_AXIS m00_axis_aclk m00_axis_aresetn simple_axis_v1.0 (Pre-Production)
run connection automation to connect the ports.	 **Run Connection Automation** Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right. ☑ All Automation (2 out of 2 selected) ☑ simple_axis_0 ☑ m00_axis_aclk ☑ s00_axis_aclk Select an interface pin on the left panel to view its options
Generate bitstream, export the hardware + bitstream to SDK. Create a new BSP and application from the Hello World template. Use the code I prepared here to replace the one generated by Xilinx: https://www.dropbox.com/sh/aylrjv0xlpz83nb/AAC7idfuSg5agvazPkr2Uwuva?dl=0	

In the application, we use two large arrays of 4KB each, we need to adjust the linker script to properly allocation the application memory to DDR. Right click on the project, choose Generate Linker Script, change the Heap Size and Stack Size accordingly (What's the difference between these two?)	
Connect the board then run to verify the results.	

Generate a linker script

Generate linker script

Control your application's memory map.

Output Settings

Project: test_axis

Output Script:

:n/ESD/Praktikum/lab2/lab2a/lab2a.sdk/test_axis/src/lscript.ld

Modify project build settings as follows:

Set generated script on all project build configurations

Hardware Memory Map

Memory	Base Address	Size
psu_ddr_0_MEM_0	0x00000000	~2 GB
psu_ocm_ram_0_MEM_0	0xFFFC0000	256 KB

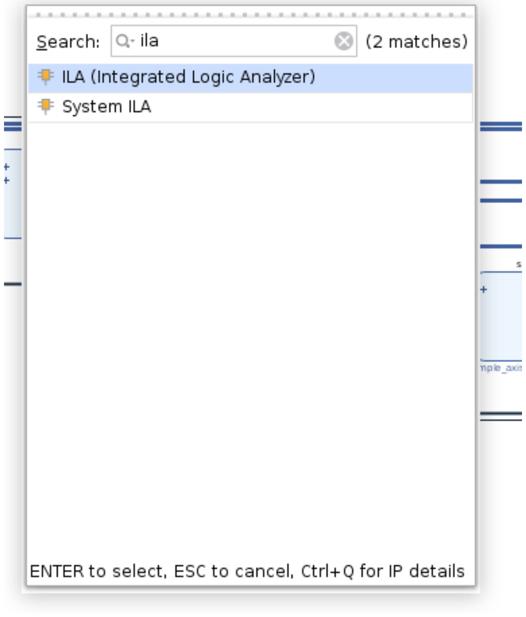
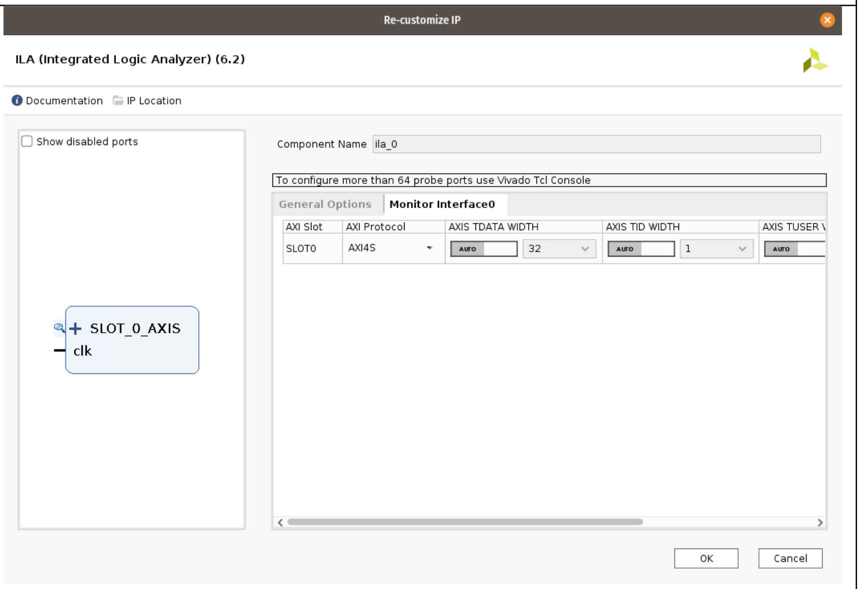
Fixed Section Assignments

Basic | Advanced

Place Code Sections in:	psu_ddr_0_MEM_0
Place Data Sections in:	psu_ddr_0_MEM_0
Place Heap and Stack in:	psu_ddr_0_MEM_0
Heap Size:	16384
Stack Size:	16 KB

Lab 2 – B

The results are not correct. However, the state machines of the accelerator seems to work correctly because the numbers of data that is received and sent back are consistent. We need to debug the design to see what is happening on the FPGA to locate the issue. First we need to verify that the data sent by the DMA_TO_SLAVE to the IP are correct and we only have the issue with the results.	

Open Block Design, add the ILA IP.	
Configure it to use the AXIS interface.	

Search: ila (2 matches)

ILA (Integrated Logic Analyzer)

System ILA

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

Re-customize IP

ILA (Integrated Logic Analyzer) (6.2)

Documentation IP Location

Show disabled ports

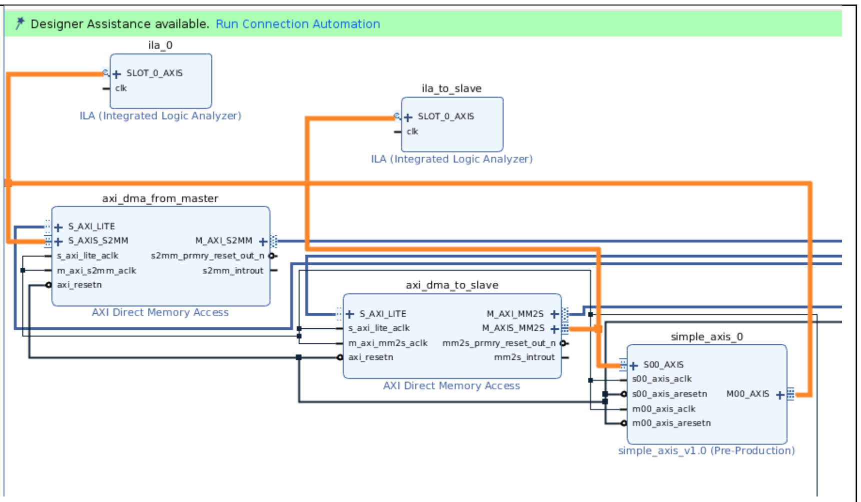
Component Name ila_0

To configure more than 64 probe ports use Vivado Tcl Console

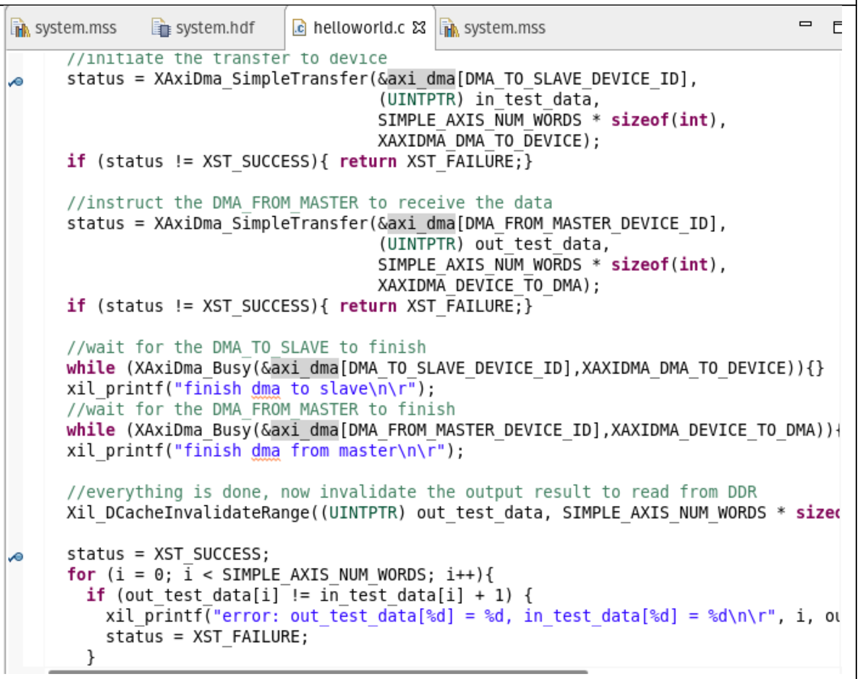
General Options | Monitor Interface0

AXI Slot	AXI Protocol	AXIS TDATA WIDTH	AXIS TID WIDTH	AXIS TUSER
SLOT0	AXI4S	AUTO 32	AUTO 1	AUTO

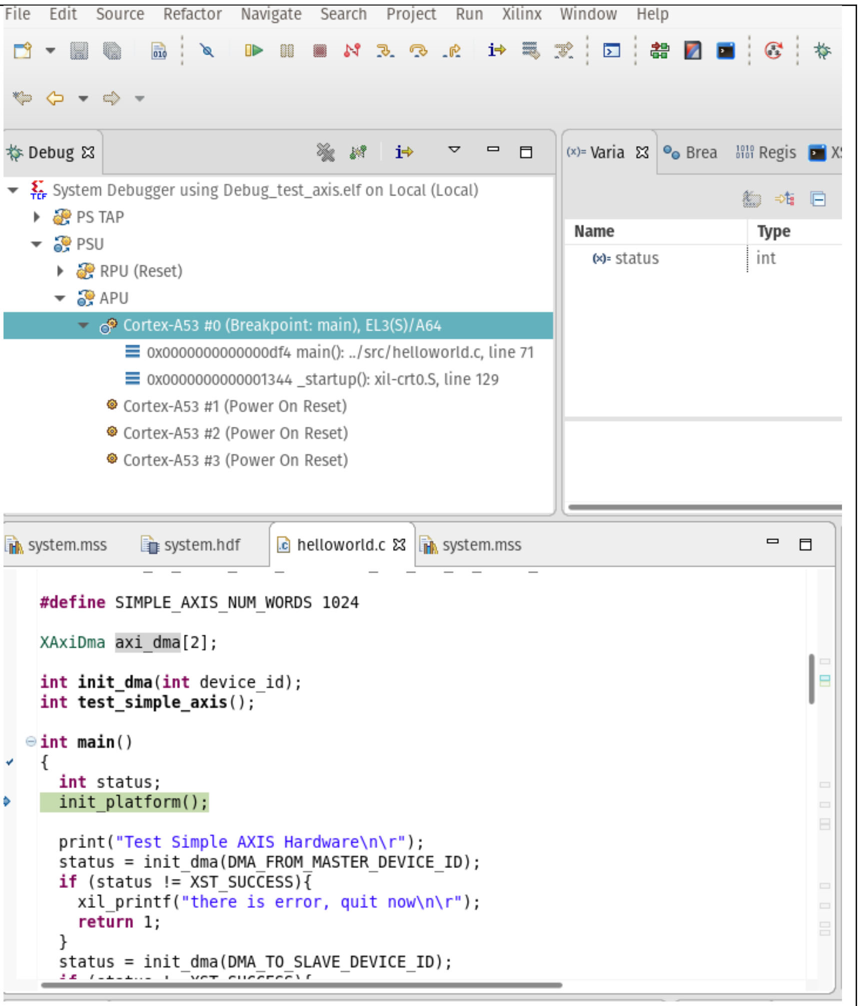
Similarly, add another ILA and connect two of them as shown. One ILA is for the Slave port, the another one is for the Master port.



Click Generate Bitstream. In the meantime, go to SDK, set two breakpoints as shown. One is just before sending the data to the IP, and the other is just after receiving the results from the IP.



```
//initiate the transfer to device
status = XAxiDma_SimpleTransfer(&axi_dma[DMA_TO_SLAVE_DEVICE_ID],
                                (UINTPTR) in_test_data,
                                SIMPLE_AXIS_NUM_WORDS * sizeof(int),
                                XAXIDMA_DMA_TO_DEVICE);
if (status != XST_SUCCESS){ return XST_FAILURE;}

//instruct the DMA_FROM_MASTER to receive the data
status = XAxiDma_SimpleTransfer(&axi_dma[DMA_FROM_MASTER_DEVICE_ID],
                                (UINTPTR) out_test_data,
                                SIMPLE_AXIS_NUM_WORDS * sizeof(int),
                                XAXIDMA_DEVICE_TO_DMA);
if (status != XST_SUCCESS){ return XST_FAILURE;}

//wait for the DMA_TO_SLAVE to finish
while (XAxiDma_Busy(&axi_dma[DMA_TO_SLAVE_DEVICE_ID],XAXIDMA_DMA_TO_DEVICE)){}
xil_printf("finish dma to slave\n\r");
//wait for the DMA_FROM_MASTER to finish
while (XAxiDma_Busy(&axi_dma[DMA_FROM_MASTER_DEVICE_ID],XAXIDMA_DEVICE_TO_DMA)){
xil_printf("finish dma from master\n\r");

//everything is done, now invalidate the output result to read from DDR
Xil_DCacheInvalidateRange((UINTPTR) out_test_data, SIMPLE_AXIS_NUM_WORDS * sizeo

status = XST_SUCCESS;
for (i = 0; i < SIMPLE_AXIS_NUM_WORDS; i++){
  if (out_test_data[i] != in_test_data[i] + 1) {
    xil_printf("error: out_test_data[%d] = %d, in_test_data[%d] = %d\n\r", i, ou
    status = XST_FAILURE;
  }
```

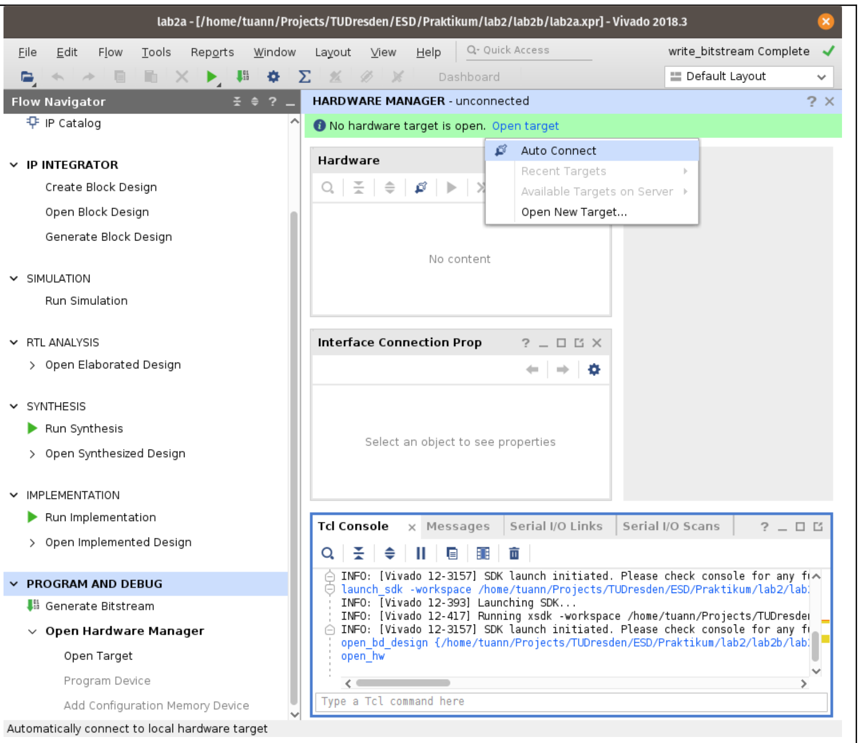
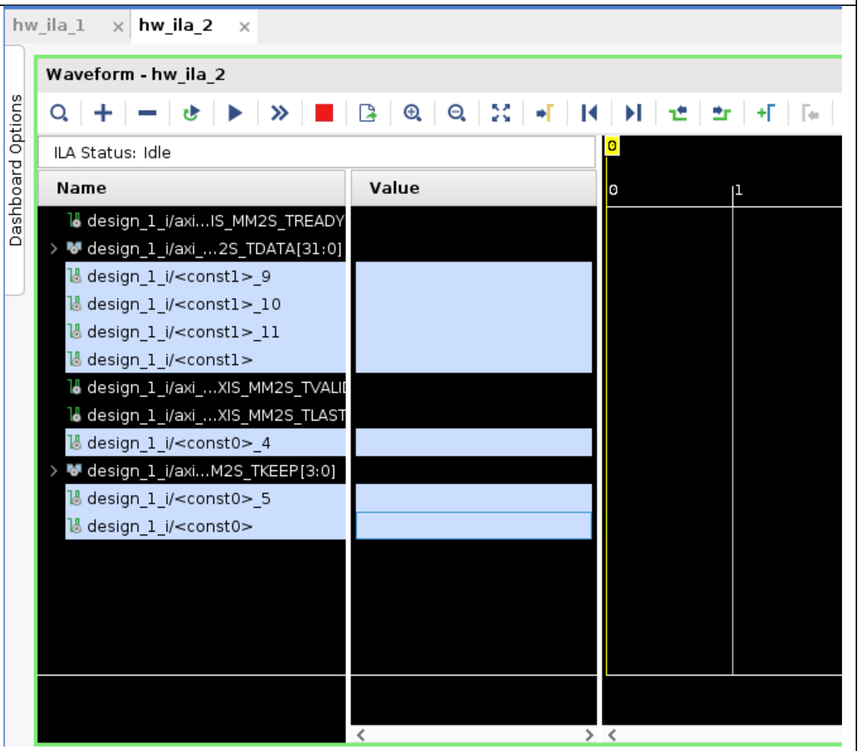
Once the bitstream generation from Vivado is complete, export the new bitstream and start debugging. The Debug perspective will be opened, once the bitstream and the application are loaded, the ARM will be stopped at the main() function.	

File Edit Source Refactor Navigate Search Project Run Xilinx Window Help
Debug
Varia
Brea
Regis
Name
Type
system.mss
system.hdf
helloworld.c
system.mss

```
System Debugger using Debug_test_axis.elf on Local (Local)
  PS TAP
  PSU
    RPU (Reset)
    APU
      Cortex-A53 #0 (Breakpoint: main), EL3(S)/A64
        0x0000000000000df4 main(): ../src/helloworld.c, line 71
        0x0000000000001344 _startup(): xil-crt0.S, line 129
      Cortex-A53 #1 (Power On Reset)
      Cortex-A53 #2 (Power On Reset)
      Cortex-A53 #3 (Power On Reset)
```

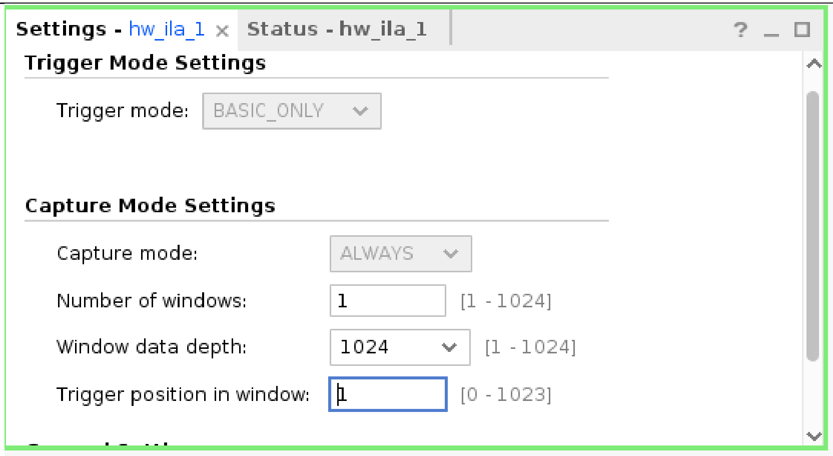
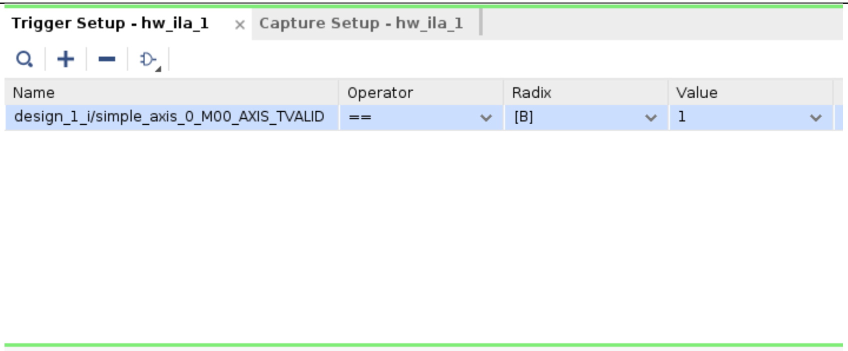
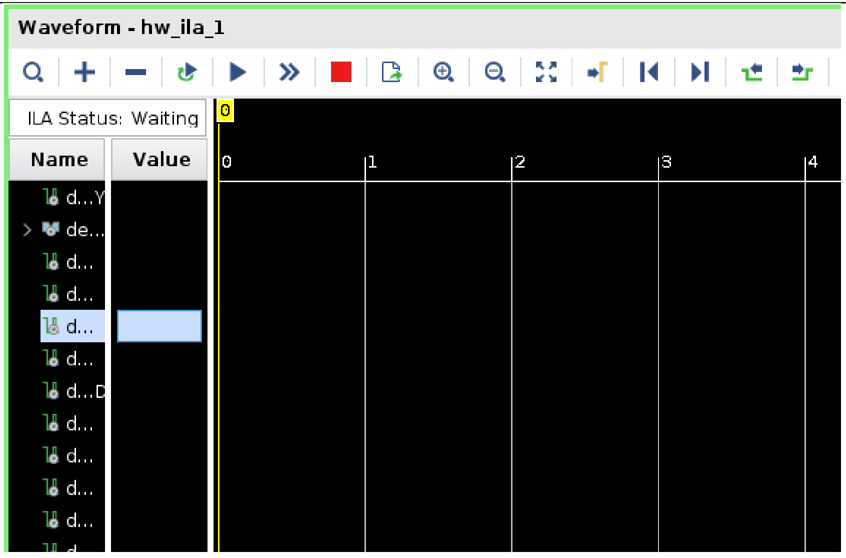
Name	Type
status	int

```
#define SIMPLE_AXIS_NUM_WORDS 1024

XAxiDma axi_dma[2];

int init_dma(int device_id);
int test_simple_axis();

int main()
{
  int status;
  init_platform();

  print("Test Simple AXIS Hardware\n\r");
  status = init_dma(DMA_FROM_MASTER_DEVICE_ID);
  if (status != XST_SUCCESS){
    xil_printf("there is error, quit now\n\r");
    return 1;
  }
  status = init_dma(DMA_TO_SLAVE_DEVICE_ID);
```

Go back to Vivado. Click on Open Hardware Manager → Open Target → Auto Connect. The purpose is to connect to the JTAG server on the board to access the ILAs that we instantiated before.	
Vivado recognizes 2 ILAs and the corresponding signals that they are connected to. Delete non-relevant signals.	

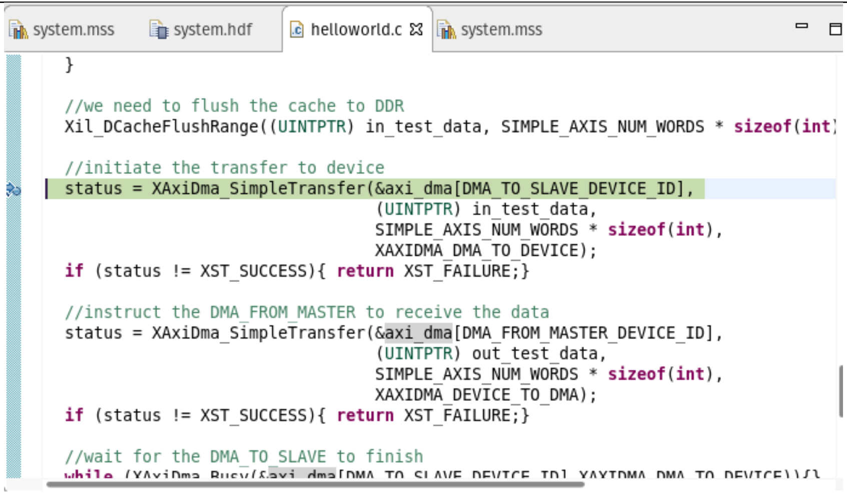
```
INFO: [Vivado 12-3157] SDK launch initiated. Please check console for any fu
launch_sdk -workspace /home/tuann/Projects/TUDresden/ESD/Praktikum/lab2/lab
INFO: [Vivado 12-393] Launching SDK...
INFO: [Vivado 12-417] Running xsdk -workspace /home/tuann/Projects/TUDresden
INFO: [Vivado 12-3157] SDK launch initiated. Please check console for any fu
open_bd_design {/home/tuann/Projects/TUDresden/ESD/Praktikum/lab2/lab2b/lab
open_hw
```

Name	Value
design_1_i/axi...IS_MM2S_TREADY	
design_1_i/axi_...2S_TDATA[31:0]	
design_1_i/<const1>_9	
design_1_i/<const1>_10	
design_1_i/<const1>_11	
design_1_i/<const1>	
design_1_i/axi_...XIS_MM2S_TVALID	
design_1_i/axi_...XIS_MM2S_TLAST	
design_1_i/<const0>_4	
design_1_i/axi...M2S_TKEEP[3:0]	
design_1_i/<const0>_5	
design_1_i/<const0>	

Under the Settings tab of the ILA that you are current opening, set the trigger position in window to 1. We can capture at most 1024 data, and we are sending 1024 data to the IP. We need to see the whole signals from the beginning.	 Settings - hw_ila_1 × Status - hw_ila_1 **Trigger Mode Settings** Trigger mode: BASIC_ONLY **Capture Mode Settings** Capture mode: ALWAYS Number of windows: 1 [1 - 1024] Window data depth: 1024 [1 - 1024] Trigger position in window: 1 [0 - 1023]
Set the trigger to sense the AXIS_TVALID signal when it gets value 1. When ever the signal is 1, all signals connected to the ILA will be captured from this point onwards until the buffer is full. The values will be shown as a waveform.	 Trigger Setup - hw_ila_1 × Capture Setup - hw_ila_1 Name: design_1_i/simple_axis_0_M00_AXIS_TVALID; Operator: ==; Radix: [B]; Value: 1
Press the "rectangle" button to start the trigger. It will wait for the signal TVALID. Similarly, setup the another ILA and start the trigger.	 Waveform - hw_ila_1 ILA Status: Waiting Name Value

Go back to SDK, click "resume" to resume the execution of the application. It will stop at the breakpoint that we set earlier.

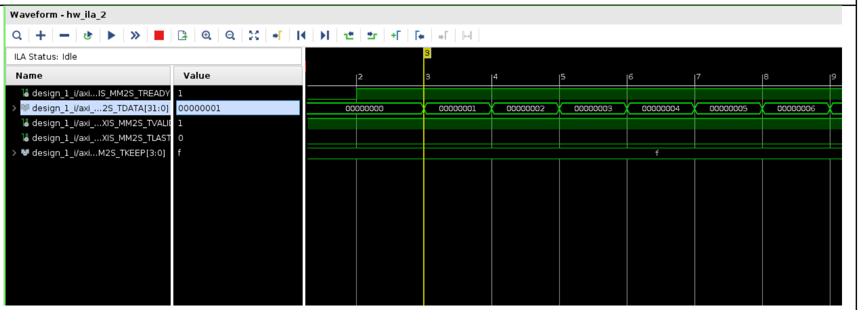
Click resume again to run to the next breakpoint.



```c
    }

    //we need to flush the cache to DDR
    Xil_DCacheFlushRange((UINTPTR) in_test_data, SIMPLE_AXIS_NUM_WORDS * sizeof(int)

    //initiate the transfer to device
    status = XAxiDma_SimpleTransfer(&axi_dma[DMA_TO_SLAVE_DEVICE_ID],
                                    (UINTPTR) in_test_data,
                                    SIMPLE_AXIS_NUM_WORDS * sizeof(int),
                                    XAXIDMA_DMA_TO_DEVICE);
    if (status != XST_SUCCESS){ return XST_FAILURE;}

    //instruct the DMA_FROM_MASTER to receive the data
    status = XAxiDma_SimpleTransfer(&axi_dma[DMA_FROM_MASTER_DEVICE_ID],
                                    (UINTPTR) out_test_data,
                                    SIMPLE_AXIS_NUM_WORDS * sizeof(int),
                                    XAXIDMA_DEVICE_TO_DMA);
    if (status != XST_SUCCESS){ return XST_FAILURE;}

    //wait for the DMA_TO_SLAVE to finish
    while (XAxiDma_Busy(&axi_dma[DMA_TO_SLAVE_DEVICE_ID],XAXIDMA_DMA_TO_DEVICE)){}
```

At this time, the data is transferred and received. You can see the waveform in Vivado now.

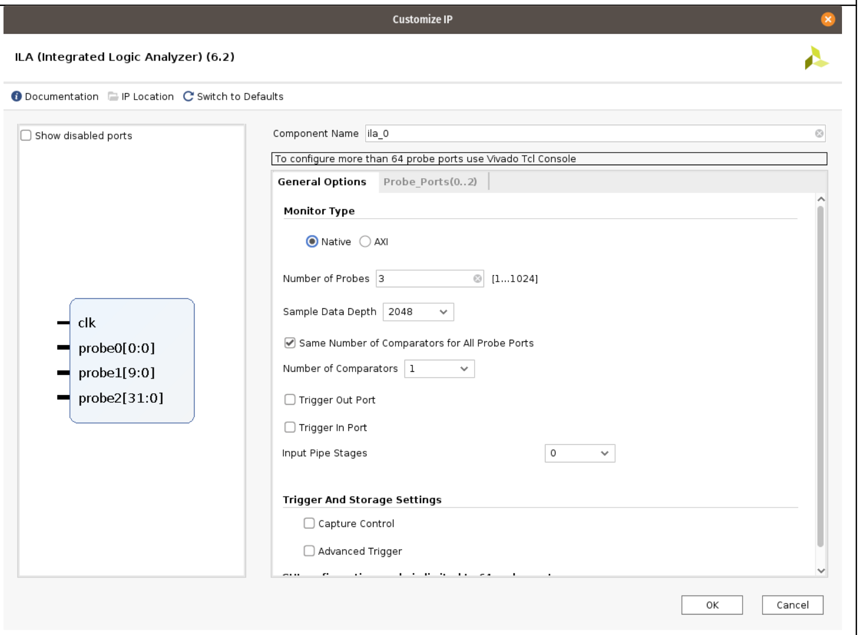
The DMA_TO_SLAVE seems to be correct. The values from DMA_FROM_MASTER are wrong. Perhaps it's the problem with the memory module.



Name	Value
design_1_i/axi...IS_MM2S_TREADY	1
design_1_i/axi_...2S_TDATA[31:0]	00000001
design_1_i/axi_...XIS_MM2S_TVALID	1
design_1_i/axi_...XIS_MM2S_TLAST	0
design_1_i/axi...M2S_TKEEP[3:0]	f

Go back to Block Design in Vivado, edit our IP.

In the IP Catalog, search for ILA and configure it to support Native signals with 3 probes (because we don't use standard interface).



ILA (Integrated Logic Analyzer) (6.2)

Show disabled ports

Component Name: ila_0

To configure more than 64 probe ports use Vivado Tcl Console

Monitor Type: [✓ Native] AXI

Number of Probes: 3 [1...1024]

Sample Data Depth: 2048

[✓] Same Number of Comparators for All Probe Ports

Number of Comparators: 1

[ ] Trigger Out Port

[ ] Trigger In Port

Input Pipe Stages: 0

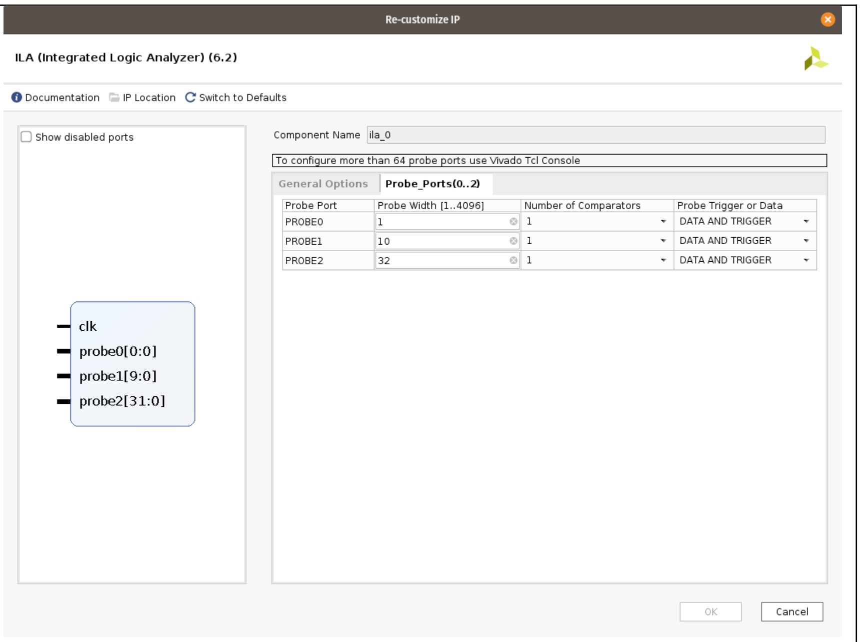
Trigger And Storage Settings

[ ] Capture Control

[ ] Advanced Trigger

Configure the width of the ports. Probe0, 1 and 2 will be connected to the En, Addr and Data signals correspondingly.

Click OK. When the Generate Output Products dialog appears, do not generate, click Cancel.


Re-customize IP

ILA (Integrated Logic Analyzer) (6.2)

Documentation IP Location Switch to Defaults

Show disabled ports

Component Name: ila_0

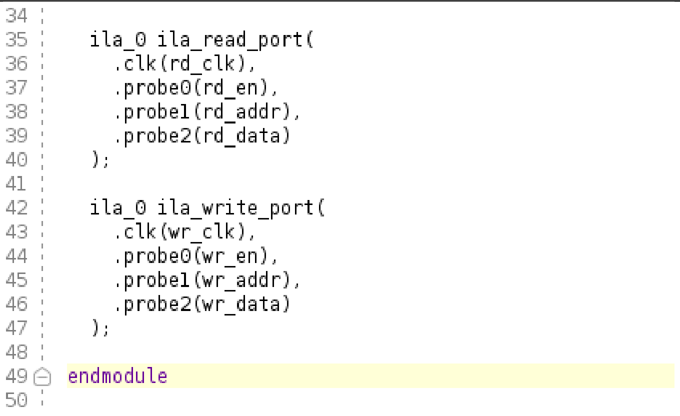
To configure more than 64 probe ports use Vivado Tcl Console

General Options | Probe_Ports(0..2)

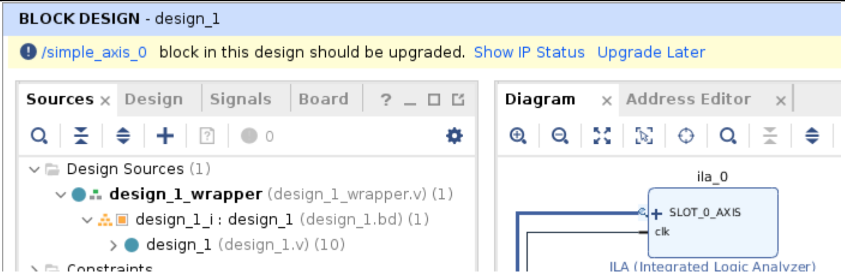
Probe Port	Probe Width [1..4096]	Number of Comparators	Probe Trigger or Data
PROBE0	1	1	DATA AND TRIGGER
PROBE1	10	1	DATA AND TRIGGER
PROBE2	32	1	DATA AND TRIGGER

Modify the memory.v file to instantiate the ILAs as shown.

Merge any changes in the IP Packager and Re-package the IPs.


```
    ila_0 ila_read_port(
      .clk(rd_clk),
      .probe0(rd_en),
      .probe1(rd_addr),
      .probe2(rd_data)
    );

    ila_0 ila_write_port(
      .clk(wr_clk),
      .probe0(wr_en),
      .probe1(wr_addr),
      .probe2(wr_data)
    );

  endmodule

```

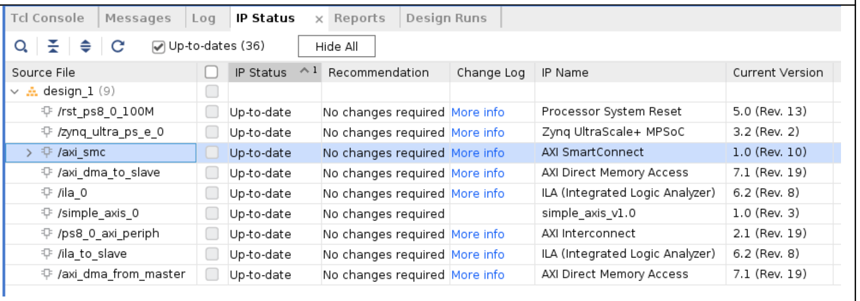
Go back to Vivado, it asks you to upgrade the IP to merge the changes. Click Show IP Status.


BLOCK DESIGN - design_1

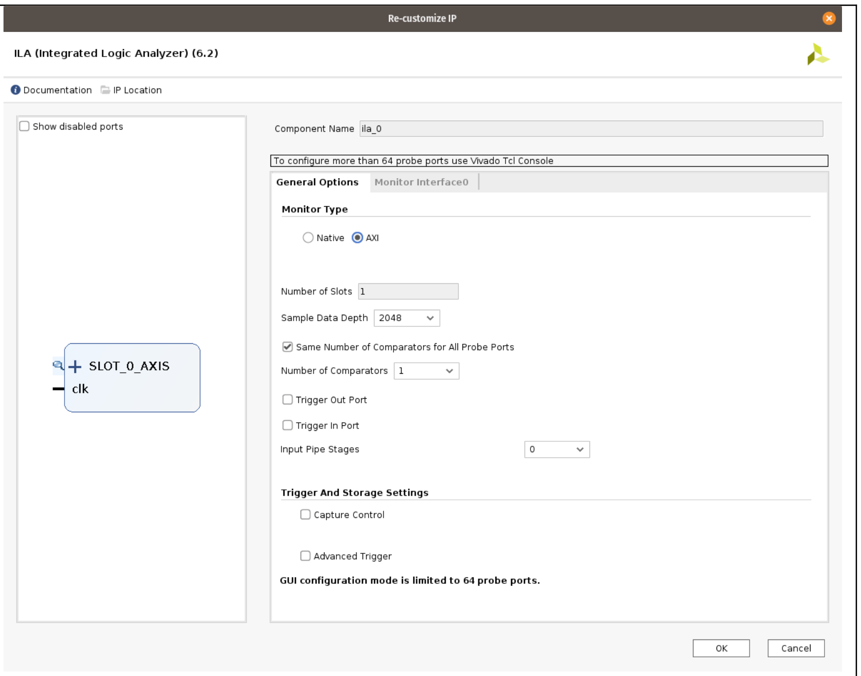
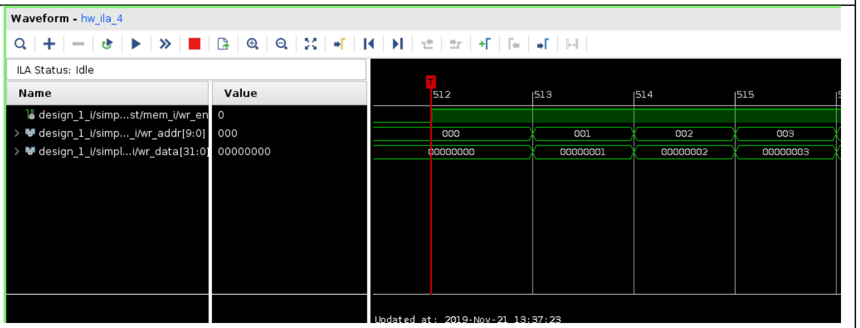
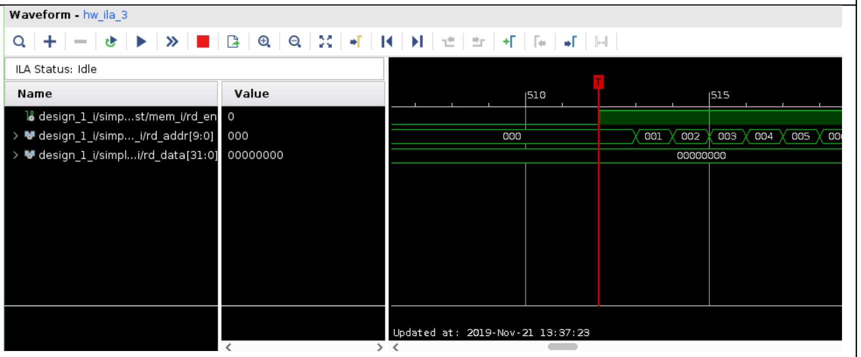
/simple_axis_0 block in this design should be upgraded. Show IP Status Upgrade Later

Sources
Design
Signals
Board
Diagram
Address Editor
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) (10)
Constraints
ila_0
SLOT_0_AXIS
clk
ILA (Integrated Logic Analyzer)

Then Upgrade Selected



Source File		IP Status	Recommendation	Change Log	IP Name	Current Version
design_1 (9)						
/rst_ps8_0_100M		Up-to-date	No changes required	More info	Processor System Reset	5.0 (Rev. 13)
/zynq_ultra_ps_e_0		Up-to-date	No changes required	More info	Zynq UltraScale+ MPSoC	3.2 (Rev. 2)
/axi_smc		Up-to-date	No changes required	More info	AXI SmartConnect	1.0 (Rev. 10)
/axi_dma_to_slave		Up-to-date	No changes required	More info	AXI Direct Memory Access	7.1 (Rev. 19)
/ila_0		Up-to-date	No changes required	More info	ILA (Integrated Logic Analyzer)	6.2 (Rev. 8)
/simple_axis_0		Up-to-date	No changes required		simple_axis_v1.0	1.0 (Rev. 3)
/ps8_0_axi_periph		Up-to-date	No changes required	More info	AXI Interconnect	2.1 (Rev. 19)
/ila_to_slave		Up-to-date	No changes required	More info	ILA (Integrated Logic Analyzer)	6.2 (Rev. 8)
/axi_dma_from_master		Up-to-date	No changes required	More info	AXI Direct Memory Access	7.1 (Rev. 19)

In the first attempt of debugging, we couldn't see the entire transaction, increase the Sample Data Depth. For two ILAs. Then generate the bitstream.	 Re-customize IP ILA (Integrated Logic Analyzer) (6.2) Documentation IP Location Show disabled ports SLOT_0_AXIS clk Component Name ila_0 To configure more than 64 probe ports use Vivado Tcl Console General Options Monitor Interface0 Monitor Type Native [✓ AXI] Number of Slots 1 Sample Data Depth 2048 ✓ Same Number of Comparators for All Probe Ports Number of Comparators 1 Trigger Out Port Trigger In Port Input Pipe Stages 0 Trigger And Storage Settings Capture Control Advanced Trigger GUI configuration mode is limited to 64 probe ports. OK Cancel
Rerun the debugging process with SDK and Vivado, setup the trigger for the two new ILAs for the Mem module. This is for the write port, it's correct.	 Waveform - hw_ila_4 ILA Status: Idle Name / Value design_1_i/simp...st/mem_i/wr_en 0 design_1_i/simp..._i/wr_addr[9:0] 000 design_1_i/simpl...i/wr_data[31:0] 00000000 512 513 514 515 000 001 002 003 00000000 00000001 00000002 00000003 Updated at: 2019-Nov-21 13:37:23
However, for the read port, the returned values are always 0. Try to look at the memory.v file and spot the issue. After fixing it, the design should work.	 Waveform - hw_ila_3 ILA Status: Idle Name / Value design_1_i/simp...st/mem_i/rd_en 0 design_1_i/simp..._i/rd_addr[9:0] 000 design_1_i/simpl...i/rd_data[31:0] 00000000 510 515 000 001 002 003 004 005 00000000 Updated at: 2019-Nov-21 13:37:23

Lab 2 – Homework 1

Modify the Vector processing accelerator in Lab1.Homework2 to receive 2 vectors from two AXIS Slave ports. The results are written to the AXIS Master port. The Size of vector and the operation are configured through the AXI4-Lite interface. The configurations received from the AXI4-Lite interface should be passed to the AXIS Slave and Master interface to know the expected number of data and the arithmetic operations.

You don't have to implement the Power operation here. If you can, it's great and can be considered as a bonus point for the final exam.

Lab 2 – Homework 2

Add timer in your C code to measure the performance between two implementations.