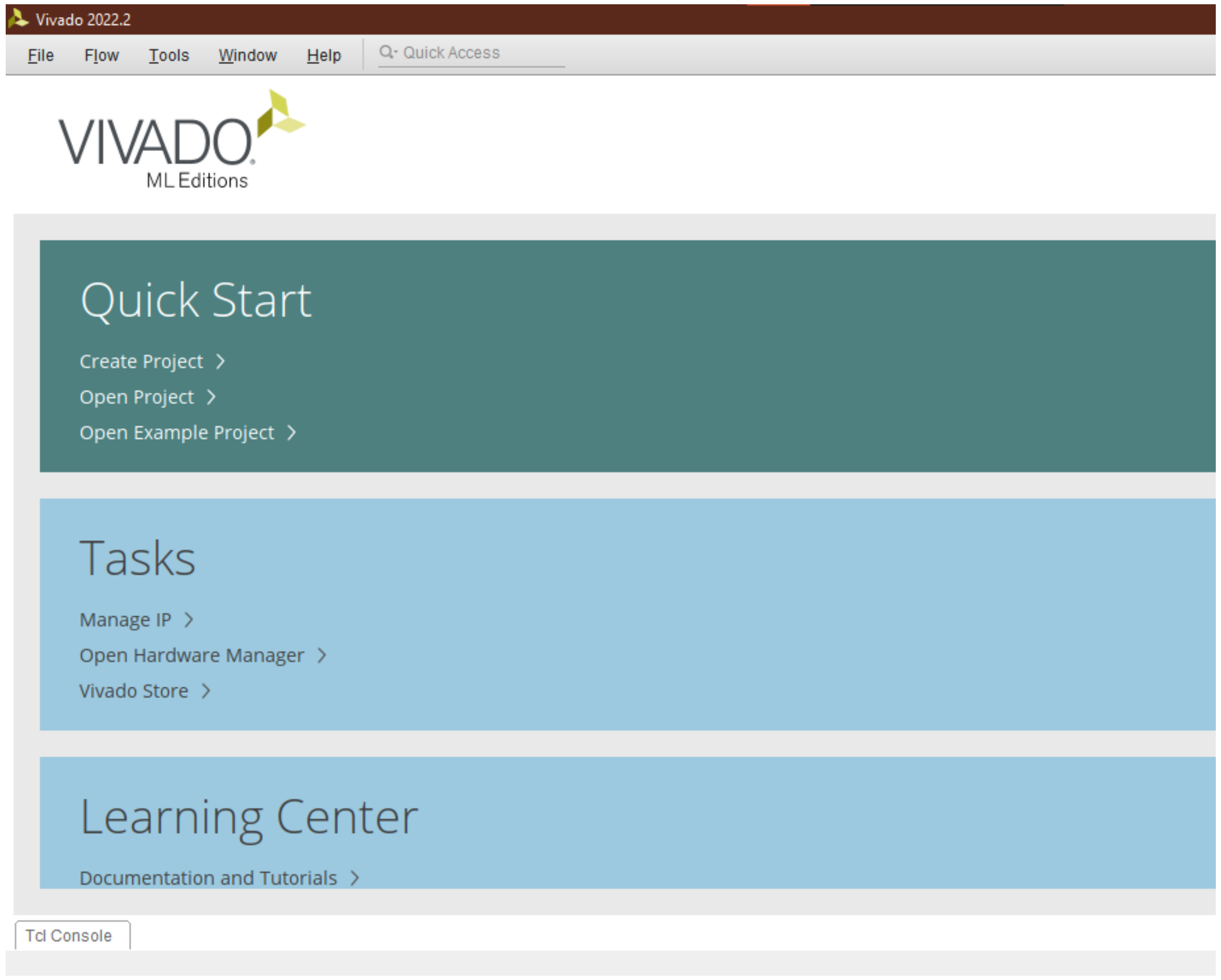
KRIA KV260
PYNQ DMA FIFO

Table of Contents

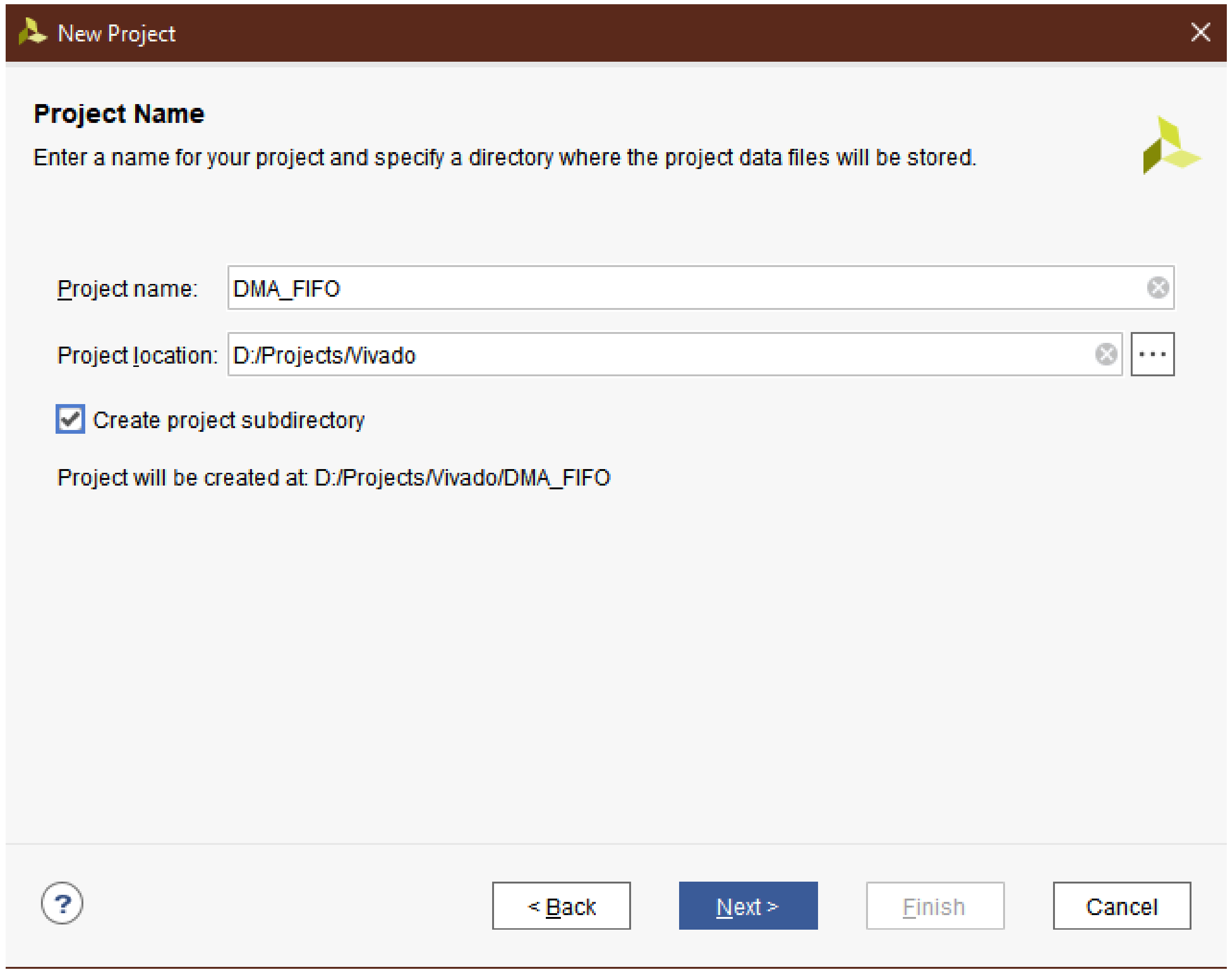
Hardware design
Project creation

- Open Vivado
- Create Project



Hardware design
Project creation

- Open Vivado
- Create Project
- Choose your project name and directory



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_FIFO

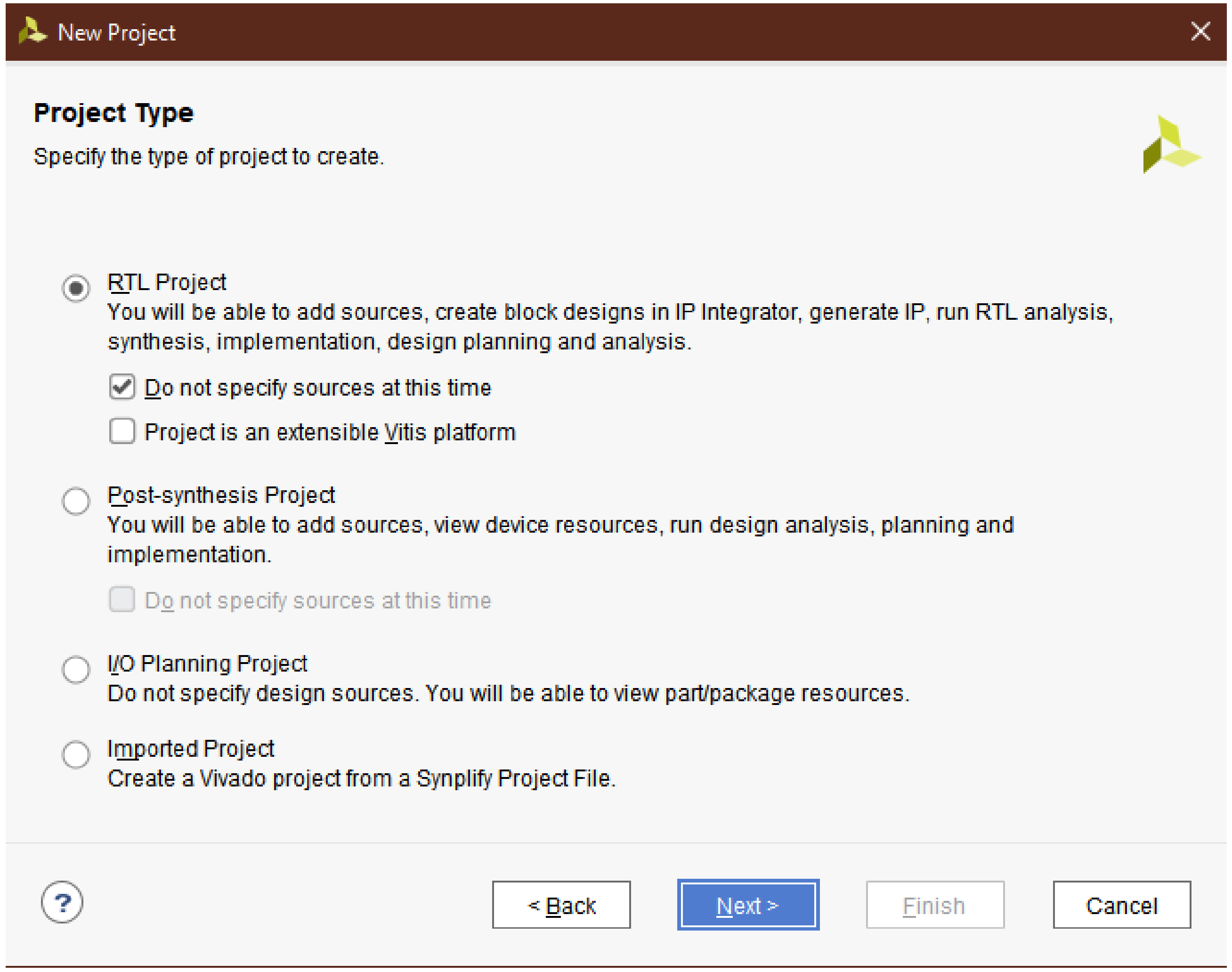
Project location: D:/Projects/Vivado

[✓] Create project subdirectory

Project will be created at: D:/Projects/Vivado/DMA_FIFO

Hardware design
Project creation

- Open Vivado
- Create Project
- Choose your project name and directory
- Choose RTL Project and not specify sources for now



Project Type

Specify the type of project to create.

[● RTL Project]
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

[✓ Do not specify sources at this time]
☐ Project is an extensible Vitis platform

○ Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.

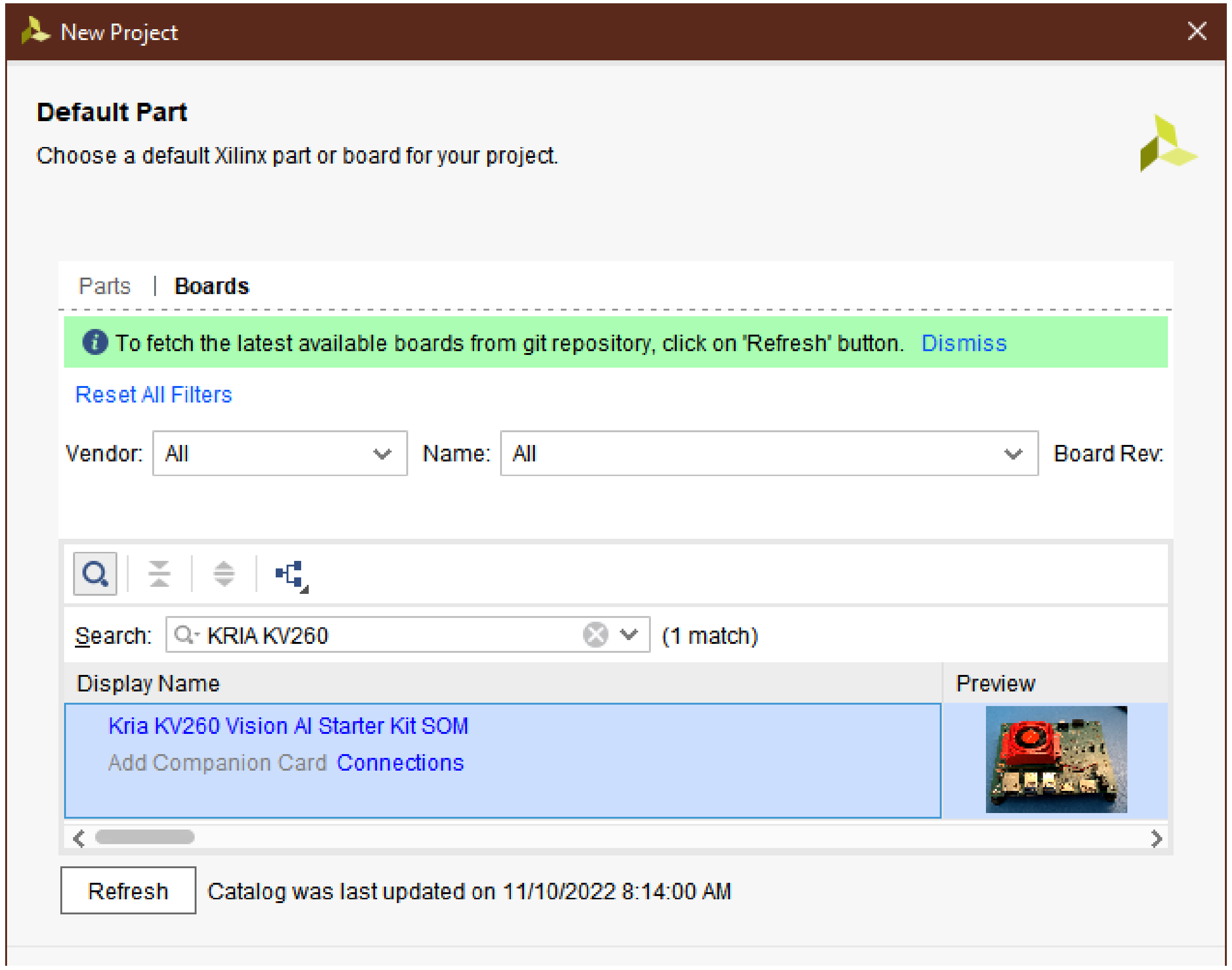
☐ Do not specify sources at this time

○ I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

○ Imported Project
Create a Vivado project from a Synplify Project File.

Hardware design
Project creation

- Open Vivado
- Create Project
- Choose your project name and directory
- Choose RTL Project and not specify sources for now
- Under the “Boards” tab, search and choose the board “KRIA KV260”


New Project

Default Part

Choose a default Xilinx part or board for your project.

Parts | **Boards**

To fetch the latest available boards from git repository, click on 'Refresh' button. Dismiss

Reset All Filters

Vendor: All Name: All Board Rev:

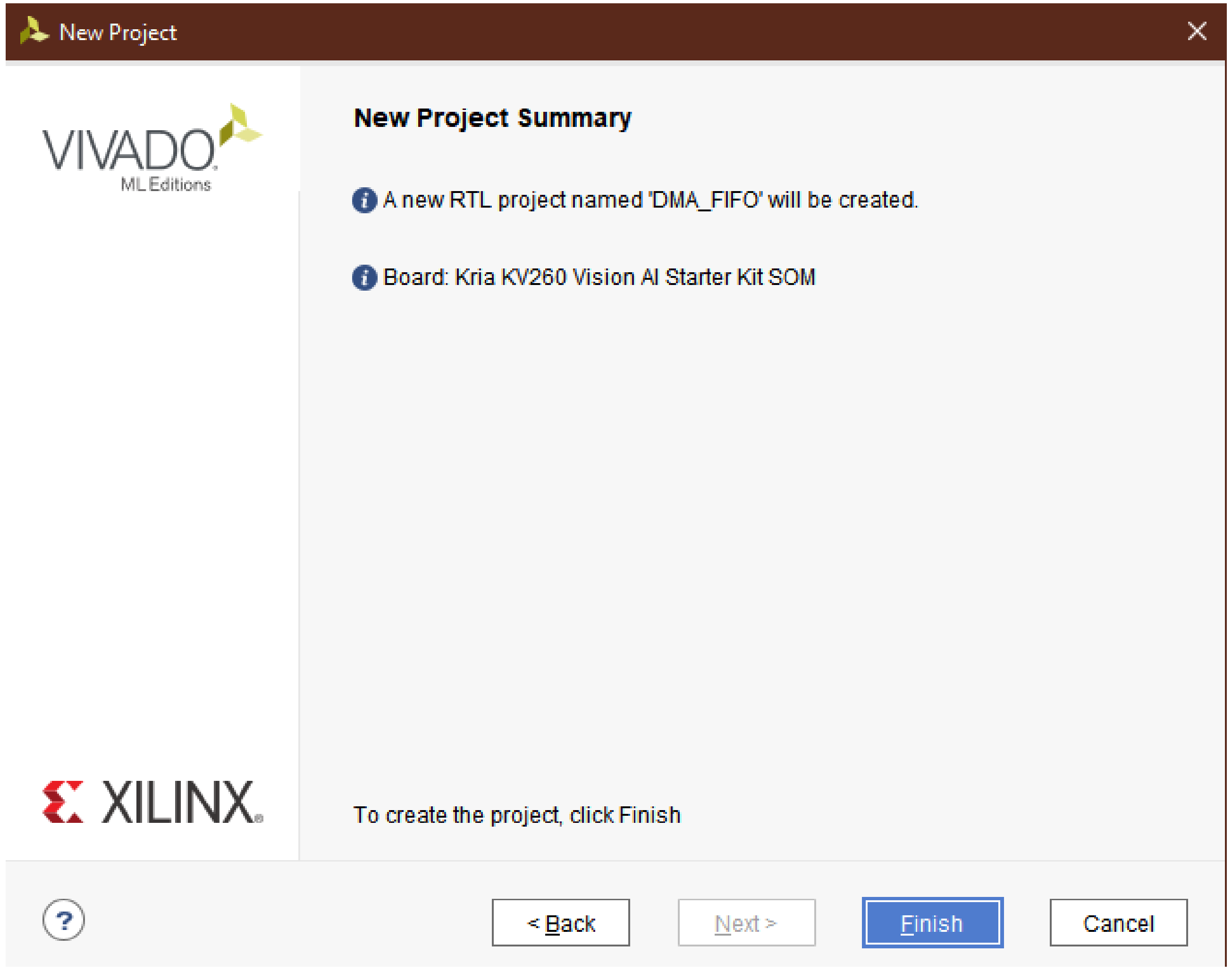
Search: KRIA KV260 (1 match)

Display Name	Preview
Kria KV260 Vision AI Starter Kit SOM Add Companion Card Connections	

Refresh Catalog was last updated on 11/10/2022 8:14:00 AM

Hardware design
Project creation

- Open Vivado
- Create Project
- Choose your project name and directory
- Choose RTL Project and not specify sources for now
- Under the "Boards" tab, search and choose the board "KRIA KV260"
- Click Next and Finish



New Project Summary

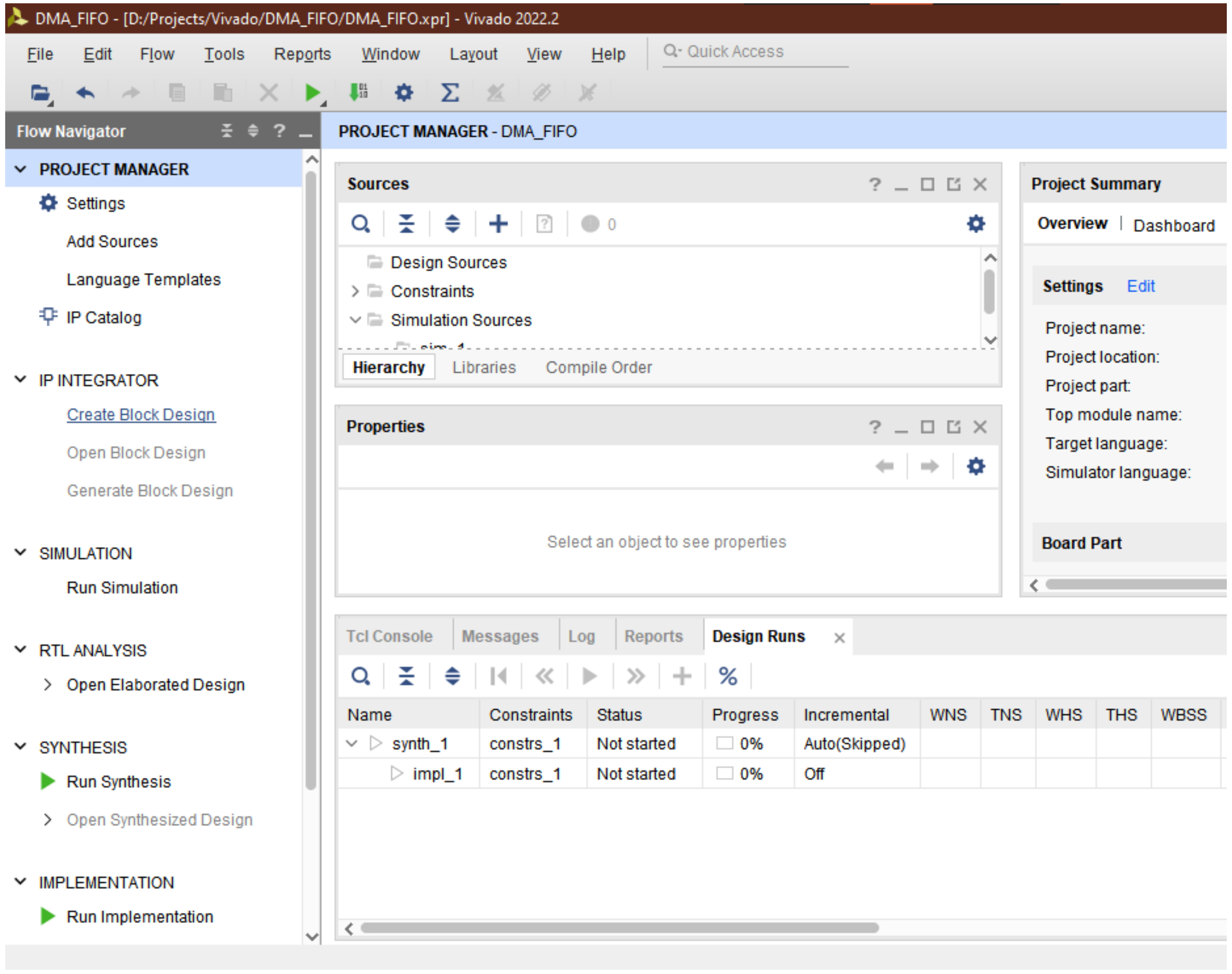
A new RTL project named 'DMA_FIFO' will be created.

Board: Kria KV260 Vision AI Starter Kit SOM

To create the project, click Finish

Hardware design
Block design

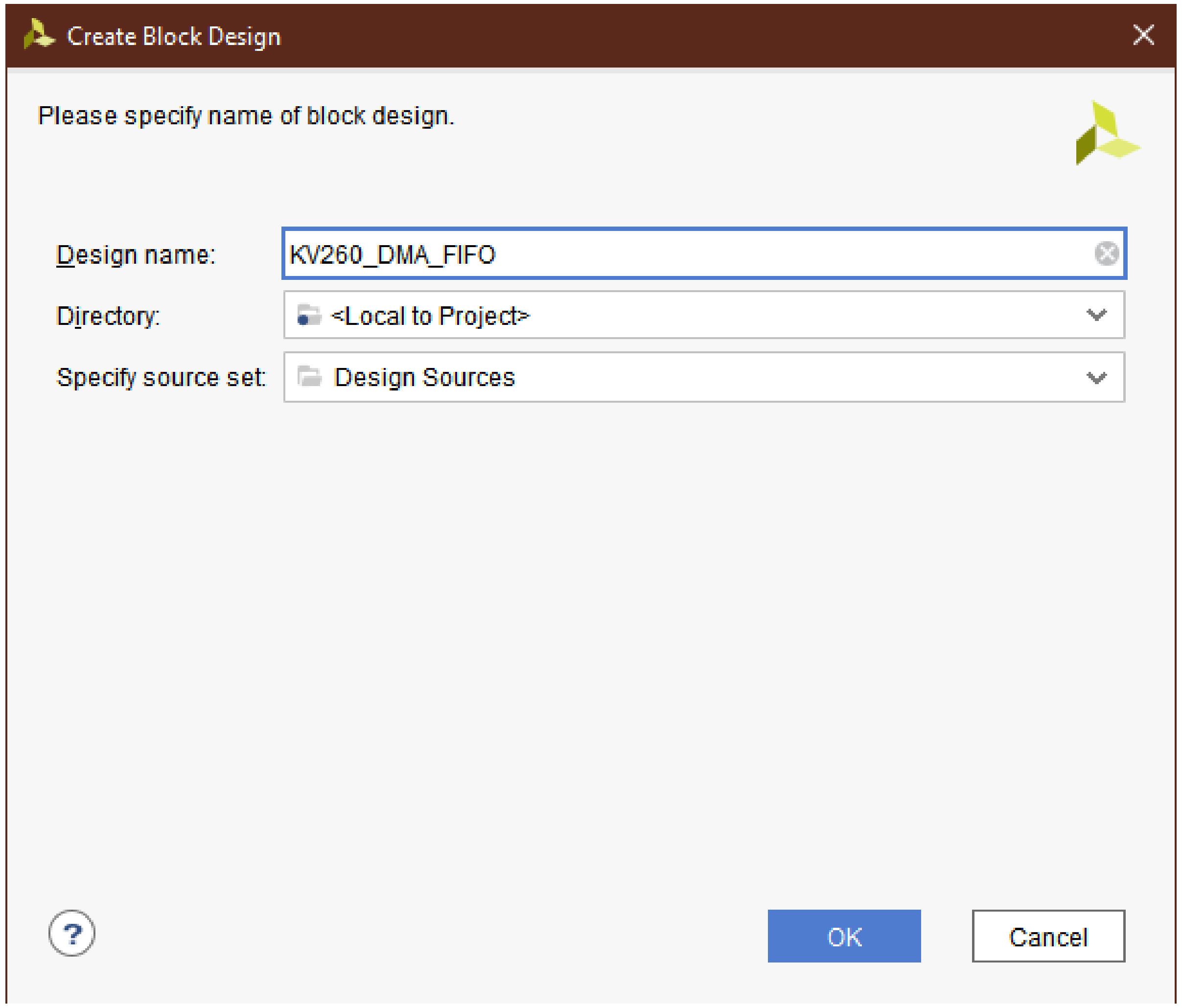
- Click “Create Block Design”


DMA_FIFO - [D:/Projects/Vivado/DMA_FIFO/DMA_FIFO.xpr] - Vivado 2022.2

Name	Constraints	Status	Progress	Incremental	WNS	TNS	WHS	THS	WBSS
synth_1	constrs_1	Not started	0%	Auto(Skipped)					
impl_1	constrs_1	Not started	0%	Off					

Hardware design
Block design

- Click "Create Block Design"
- Choose your name and click "OK"



Please specify name of block design.

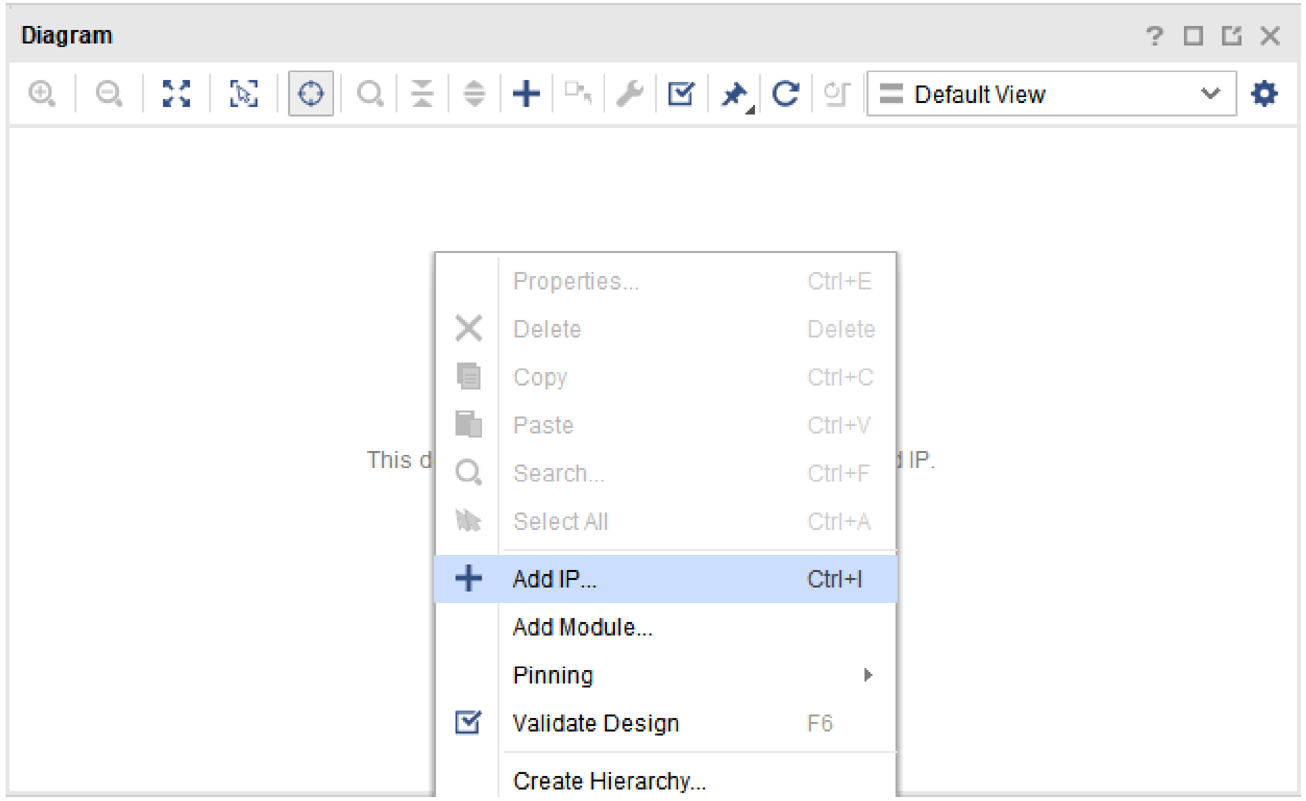
Design name: KV260_DMA_FIFO

Directory: <Local to Project>

Specify source set: Design Sources

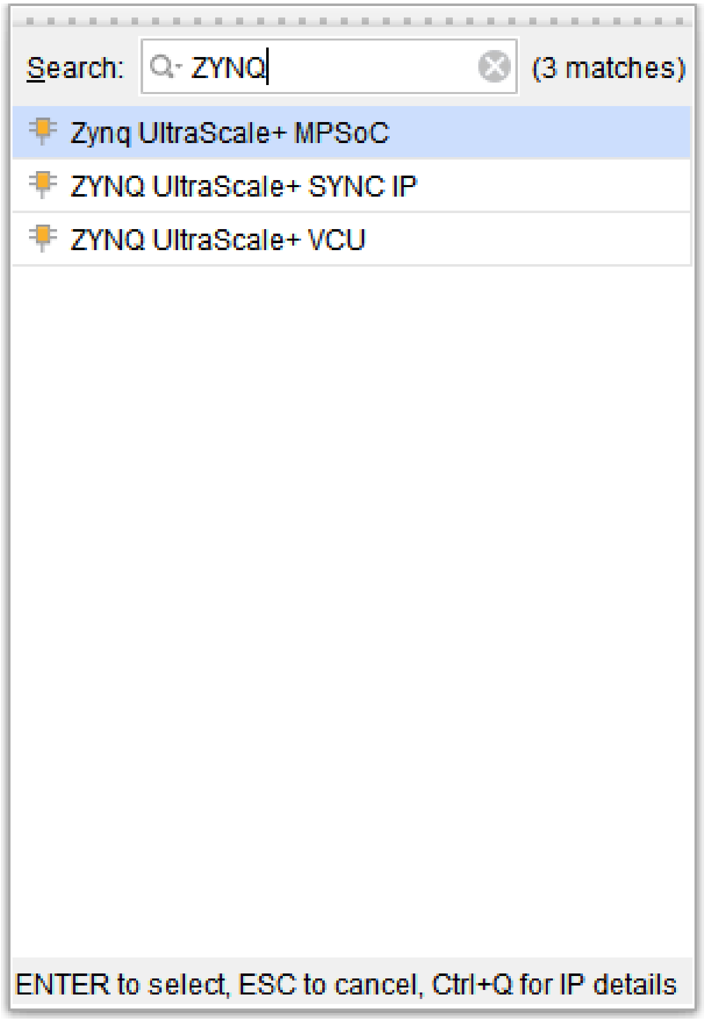
Hardware design
Block design

- Click “Create Block Design”
- Choose your name and click “OK”
- Right click on the diagram and click “Add IP”



Hardware design
Block design

- Click “Create Block Design”
- Choose your name and click “OK”
- Right click on the diagram and click “Add IP”
- Search and select ZYNQ MPSoC



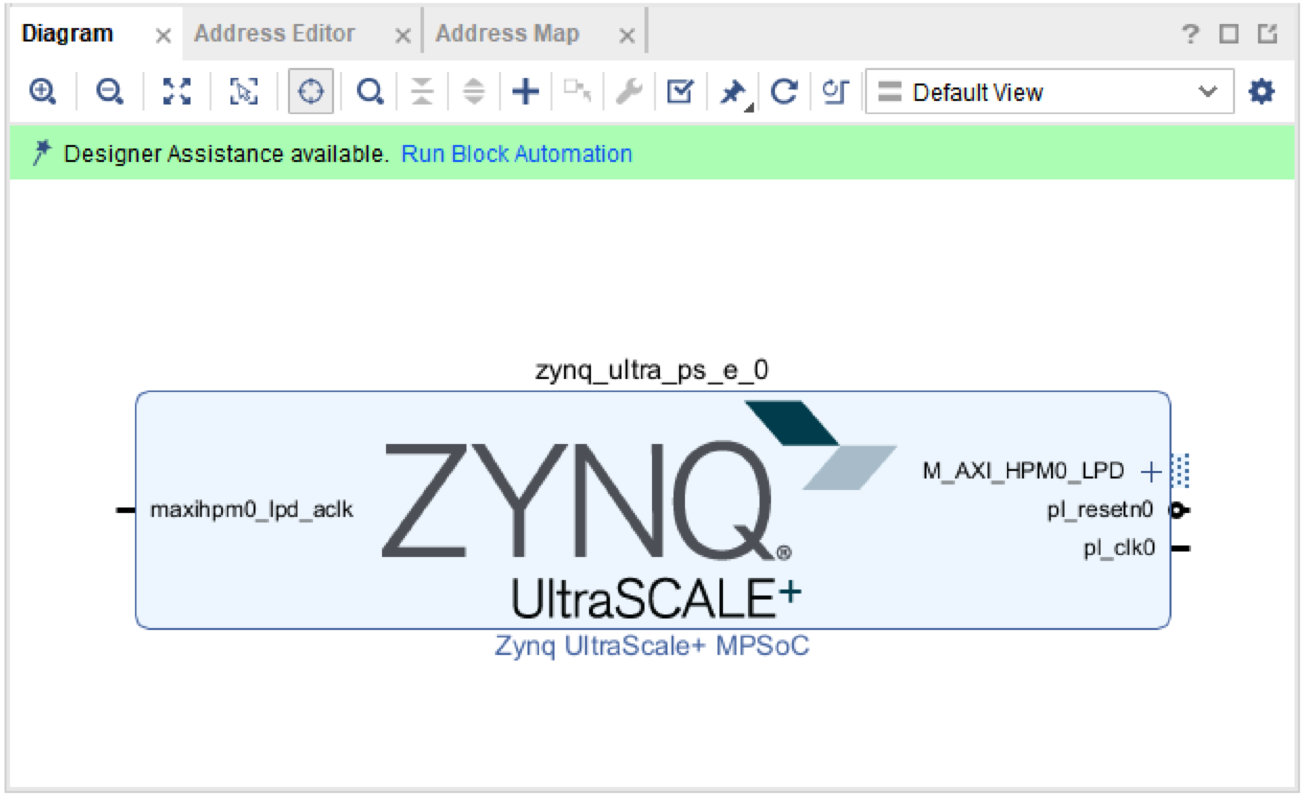
Search: ZYNQ (3 matches)

Zynq UltraScale+ MPSoC
ZYNQ UltraScale+ SYNC IP
ZYNQ UltraScale+ VCU

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

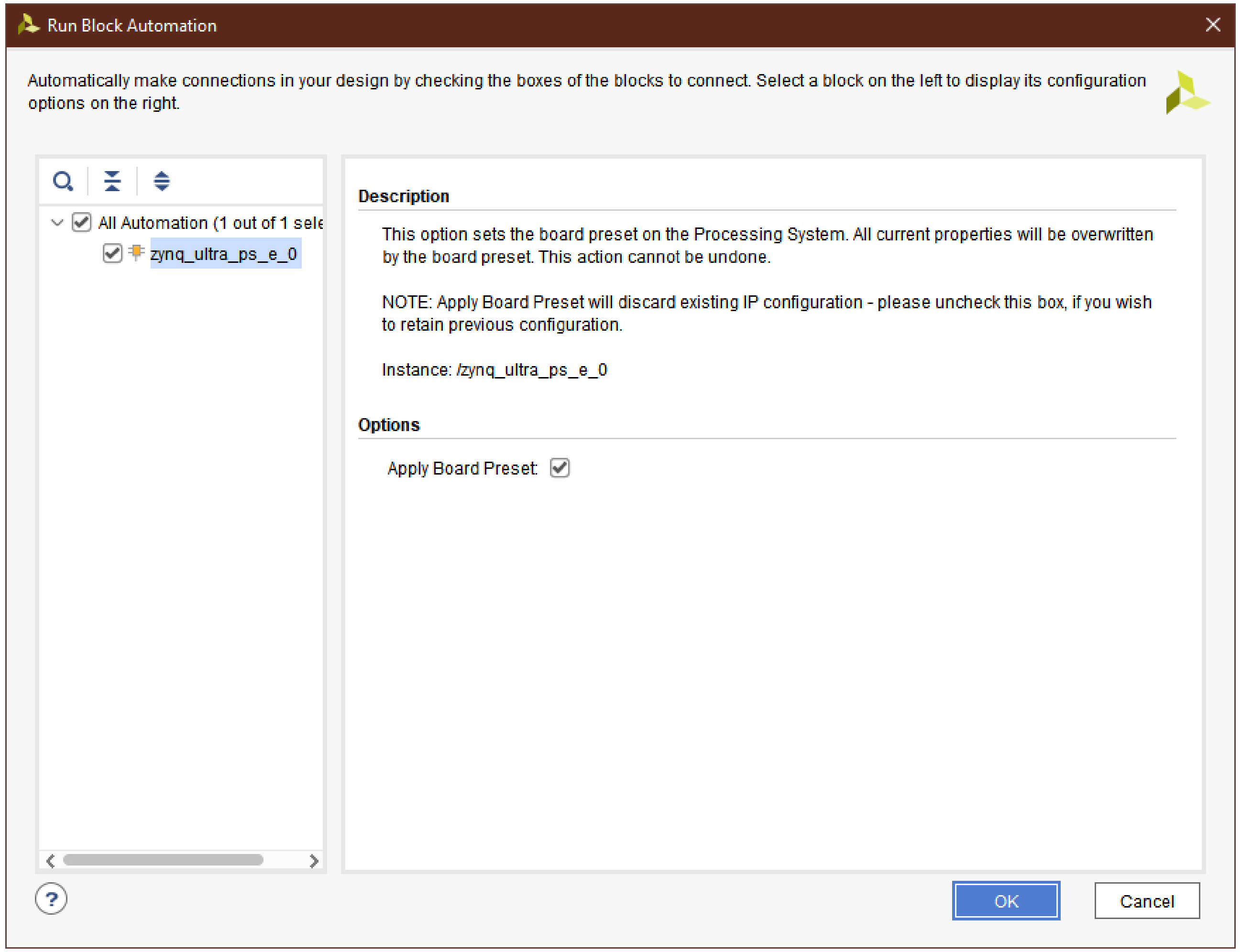
Hardware design
Block design

- Click “Create Block Design”
- Choose your name and click “OK”
- Right click on the diagram and click “Add IP”
- Search and select ZYNQ MPSoC
- Click “Run Block Automation” to apply the specifications and settings of the KV260 Board to the ZYNQ block



Hardware design
Block design

- Click “Create Block Design”
- Choose your name and click “OK”
- Right click on the diagram and click “Add IP”
- Search and select ZYNQ MPSoC
- Click “Run Block Automation” to apply the specifications and settings of the KV260 Board to the ZYNQ block



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 (1 out of 1 sele
☑ zynq_ultra_ps_e_0

Description

This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone.

NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

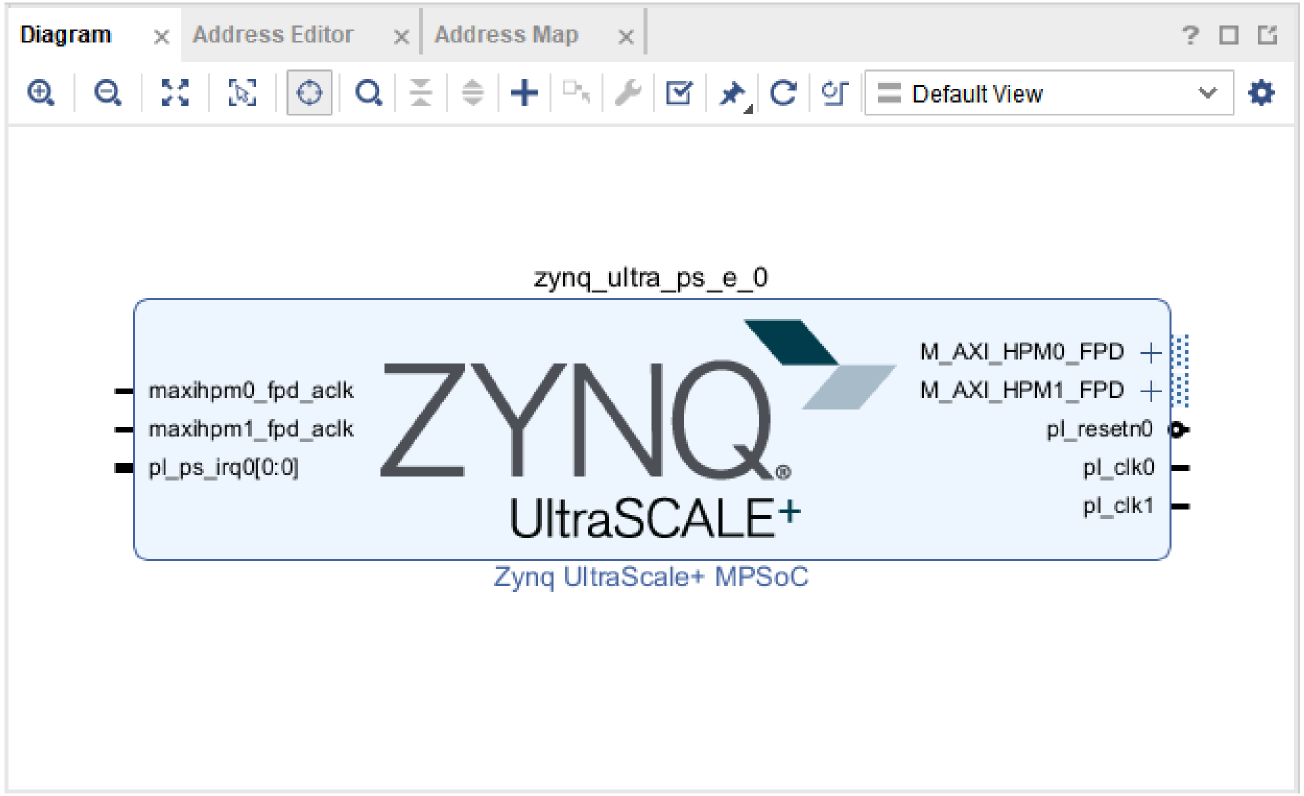
Instance: /zynq_ultra_ps_e_0

Options

Apply Board Preset ☑

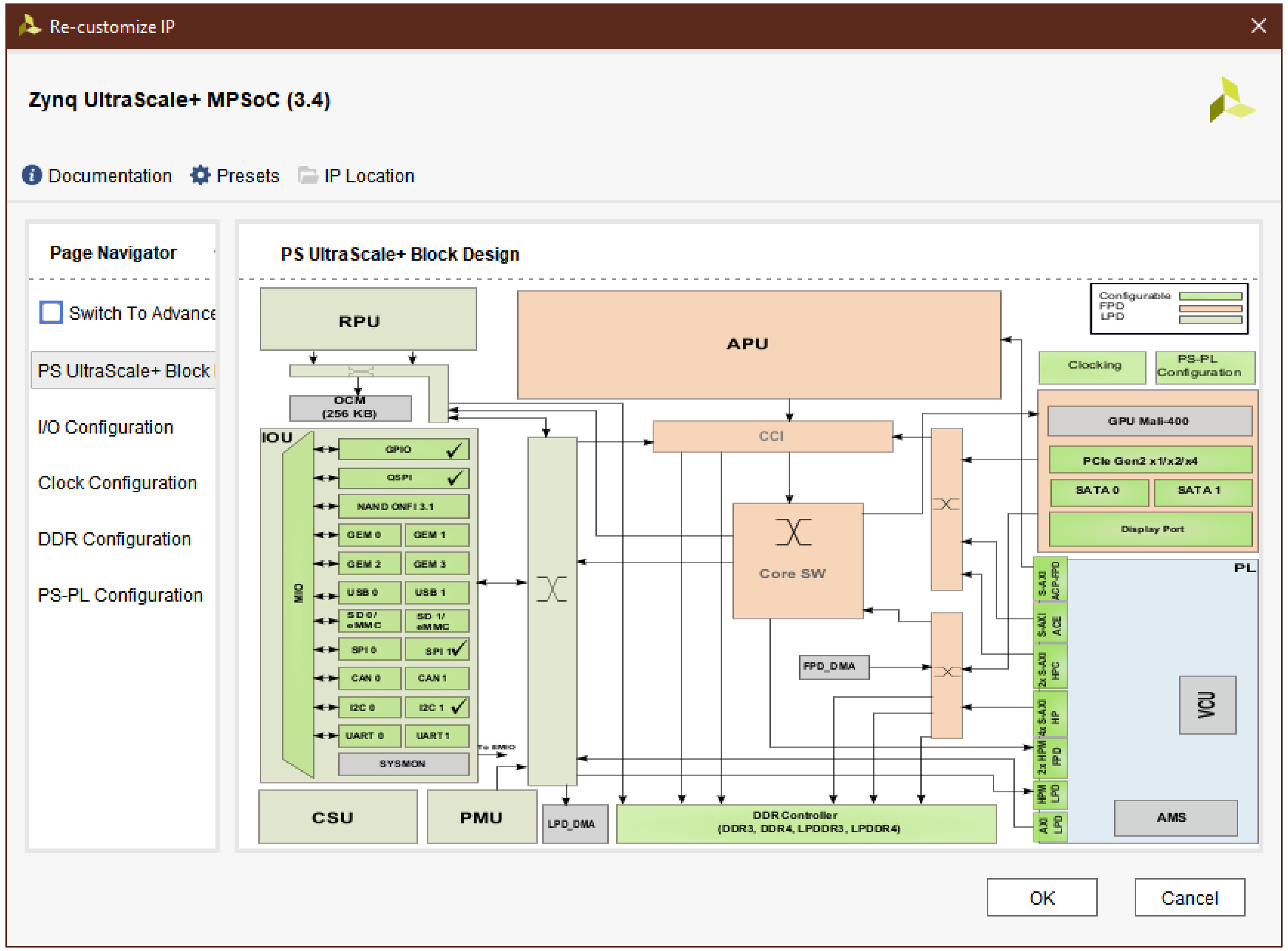
Hardware design
Block design

- Click “Create Block Design”
- Choose your name and click “OK”
- Right click on the diagram and click “Add IP”
- Search and select ZYNQ MPSoC
- Click “Run Block Automation” to apply the specifications and settings of the KV260 Board to the ZYNQ block



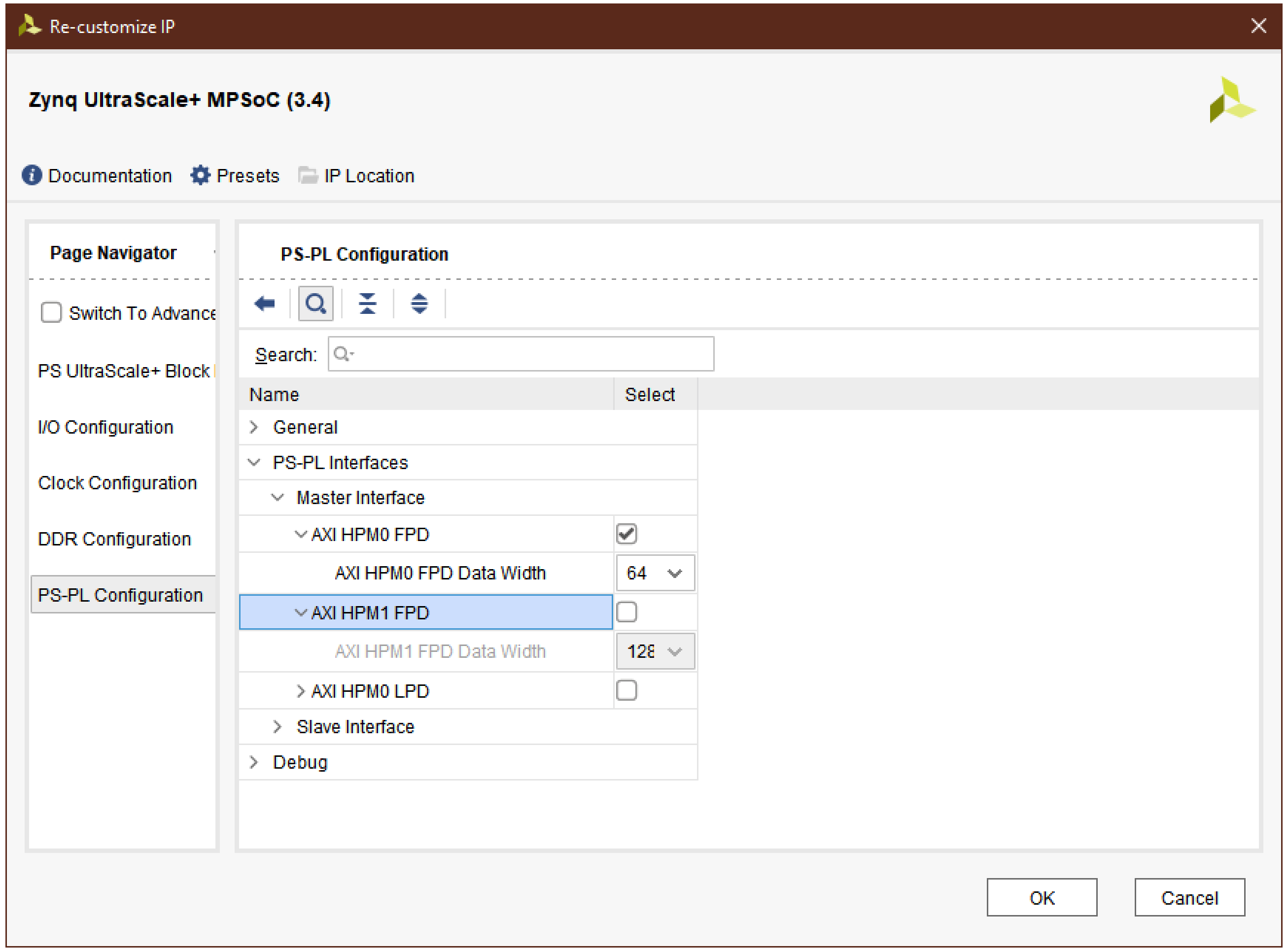
Hardware design
Block design

- Click “Create Block Design”
- Choose your name and click “OK”
- Right click on the diagram and click “Add IP”
- Search and select ZYNQ MPSoC
- Click “Run Block Automation” to apply the specifications and settings of the KV260 Board to the ZYNQ block
- Double click on the block to customize your IP



Hardware design
Block design

- Click “Create Block Design”
- Choose your name and click “OK”
- Right click on the diagram and click “Add IP”
- Search and select ZYNQ MPSoC
- Click “Run Block Automation” to apply the specifications and settings of the KV260 Board to the ZYNQ block
- Double click on the block to customize your IP
- Under PS-PL configuration → PS-PL Interfaces → Master Interface, deselect AXI HPM1 FPD and set the AXI HPM0 FPD Data width to 64



Re-customize IP

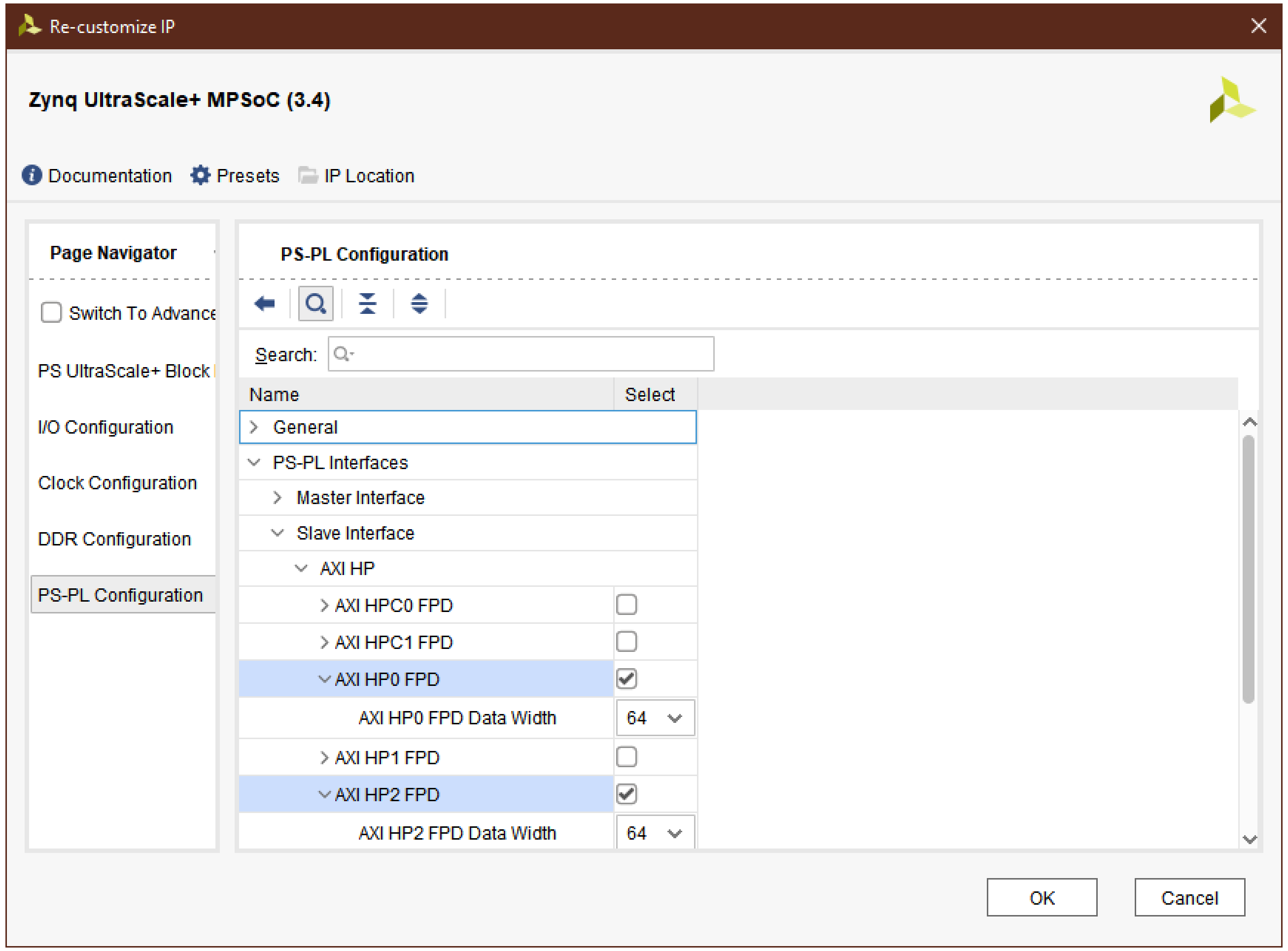
Zynq UltraScale+ MPSoC (3.4)

PS-PL Configuration

Name	Select
General	
PS-PL Interfaces	
Master Interface	
AXI HPM0 FPD	✓
AXI HPM0 FPD Data Width	64
AXI HPM1 FPD	☐
AXI HPM1 FPD Data Width	128
AXI HPM0 LPD	☐
Slave Interface	
Debug	

Hardware design Block design (Cont.)

- Under PS-PL configuration → PS-PL Interfaces → Slave Interface, enable AXI HP0 HPD and AXI HP2 HPD. Set their data width to 64



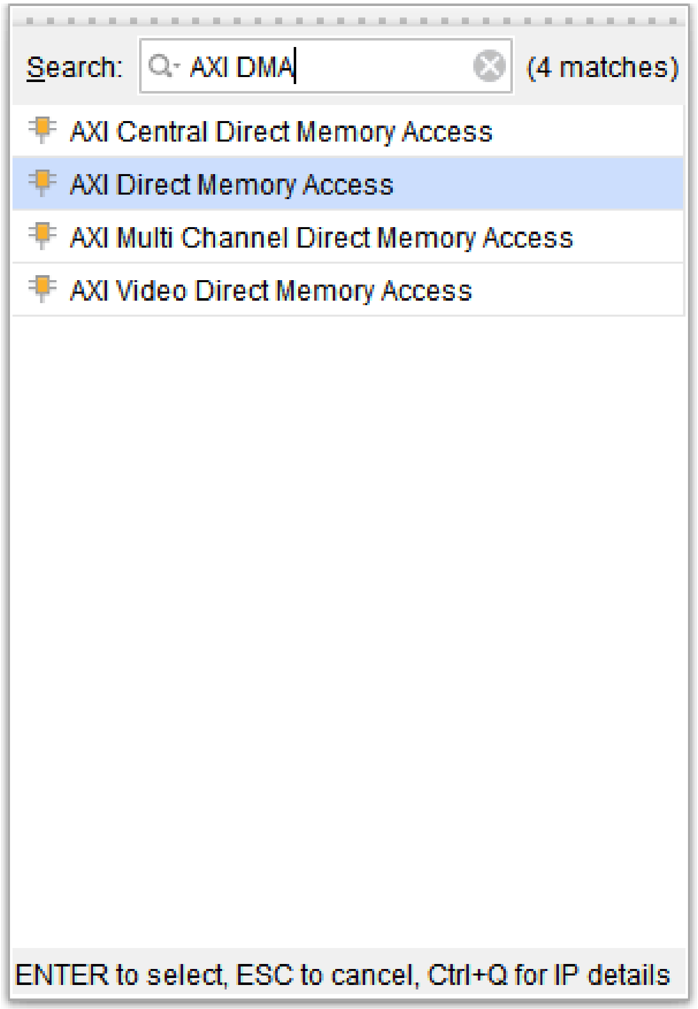
Zynq UltraScale+ MPSoC (3.4)

PS-PL Configuration

Name	Select
General	
PS-PL Interfaces	
Master Interface	
Slave Interface	
AXI HP	
AXI HPC0 FPD	☐
AXI HPC1 FPD	☐
AXI HP0 FPD	✓
AXI HP0 FPD Data Width	64
AXI HP1 FPD	☐
AXI HP2 FPD	✓
AXI HP2 FPD Data Width	64

Hardware design
Block design (Cont.)

- Under PS-PL configuration → PS-PL Interfaces → Slave Interface, enable AXI HP0 HPD and AXI HP2 HPD. Set their data width to 64
- Add an IP called AXI Direct Memory Access. Double click to customize it

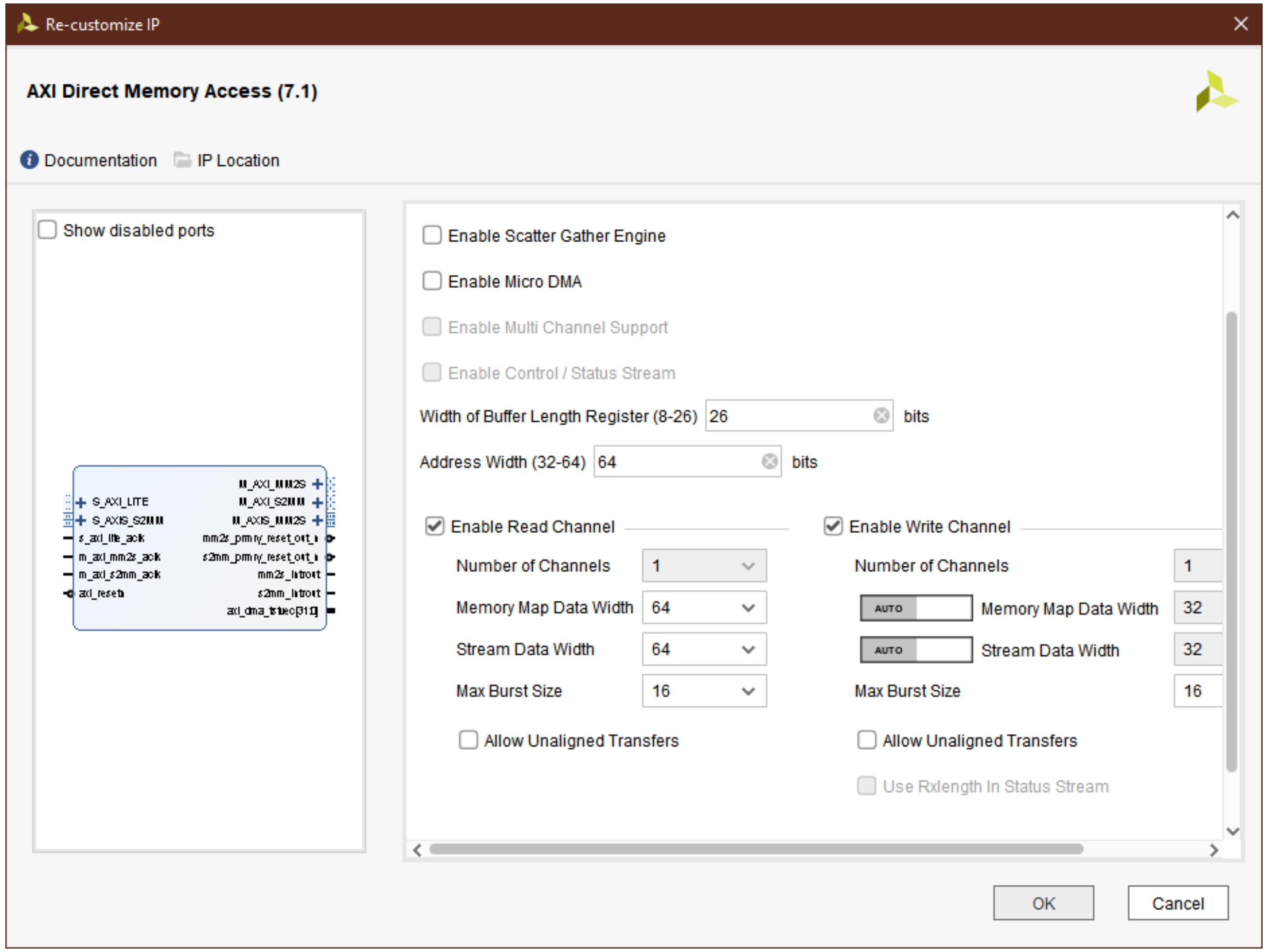

Search: AXI DMA (4 matches)

AXI Central Direct Memory Access
AXI Direct Memory Access
AXI Multi Channel Direct Memory Access
AXI Video Direct Memory Access

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

Hardware design
Block design (Cont.)

- Under PS-PL configuration → PS-PL Interfaces → Slave Interface, enable AXI HP0 HPD and AXI HP2 HPD. Set their data width to 64
- Add an IP called AXI Direct Memory Access. Double click to customize it
- Disable Scatter Gather Engine, set the width of buffer length register to 26, set the address width to 64. In the read channel panel, set the memory map data width and the stream data width to 64. Set the max burst size to 16. The write channel can be left as AUTO. Make sure no Allow Unaligned Transfers boxes are enabled



AXI Direct Memory Access (7.1)

- [ ] Enable Scatter Gather Engine
- [ ] Enable Micro DMA
- [ ] Enable Multi Channel Support
- [ ] Enable Control / Status Stream

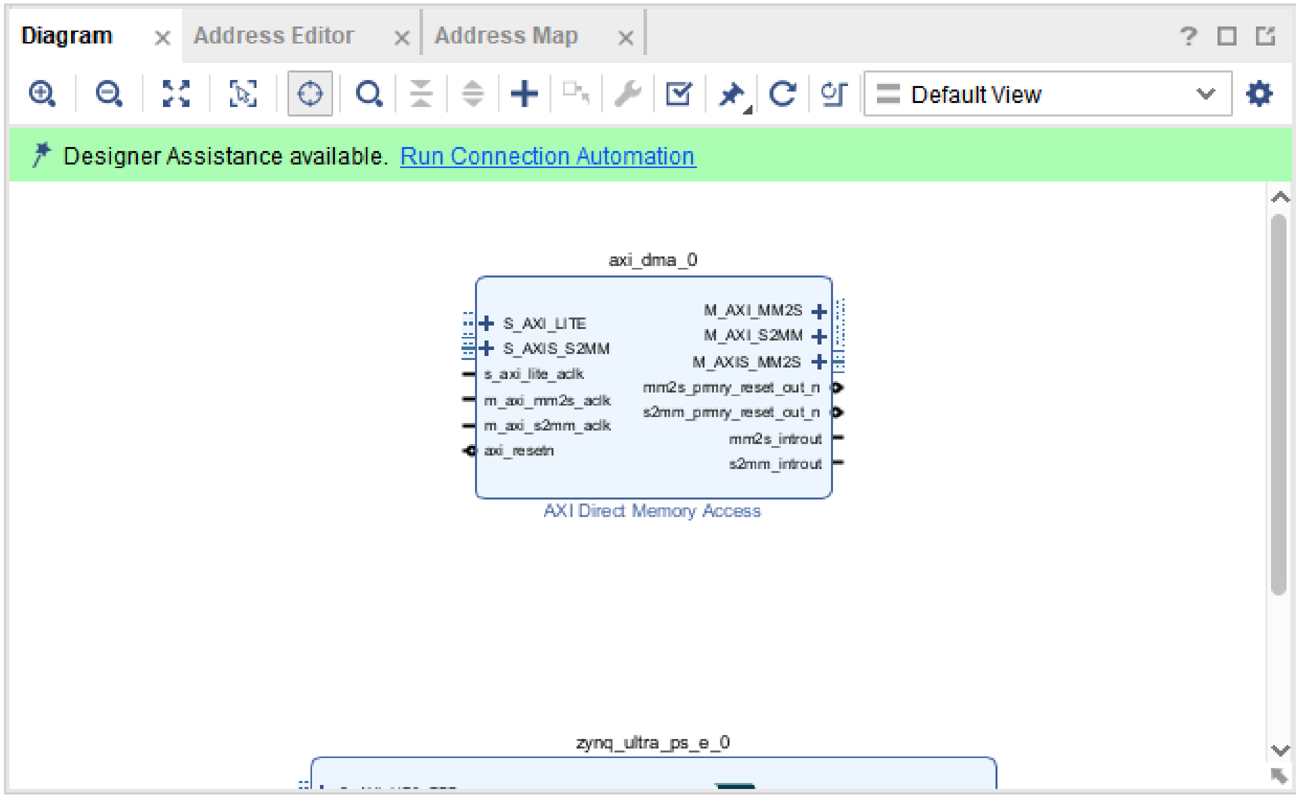
Width of Buffer Length Register (8-26): 26 bits

Address Width (32-64): 64 bits

✓ Enable Read Channel	✓ Enable Write Channel
Number of Channels: 1	Number of Channels: 1
Memory Map Data Width: 64	AUTO Memory Map Data Width: 32
Stream Data Width: 64	AUTO Stream Data Width: 32
Max Burst Size: 16	Max Burst Size: 16
☐ Allow Unaligned Transfers	☐ Allow Unaligned Transfers
	☐ Use Rxlength In Status Stream

Hardware design
Block design (Cont.)

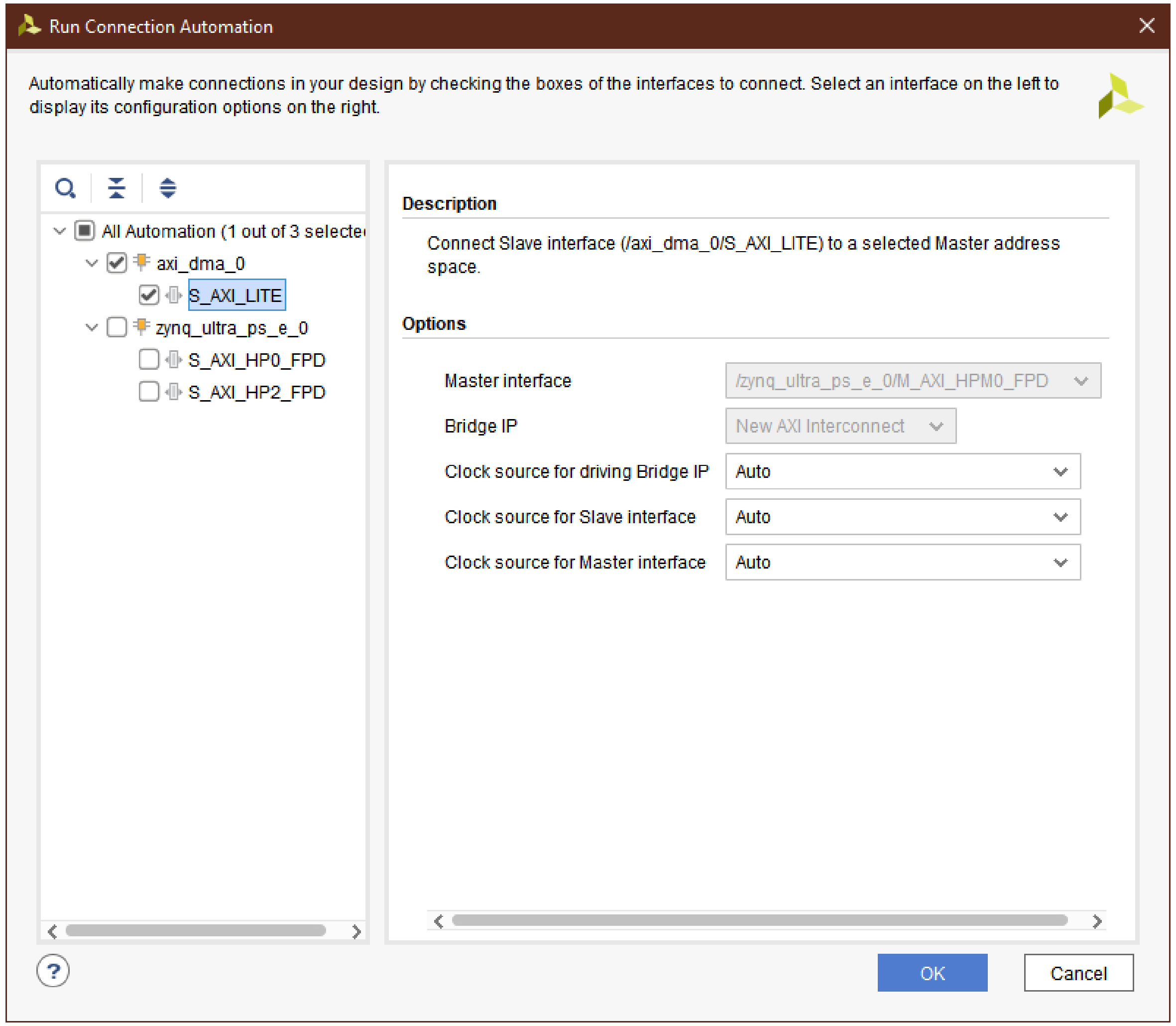
- Under PS-PL configuration → PS-PL Interfaces → Slave Interface, enable AXI HP0 HPD and AXI HP2 HPD. Set their data width to 64
- Add an IP called AXI Direct Memory Access. Double click to customize it
- Disable Scatter Gather Engine, set the width of buffer length register to 26, set the address width to 64. In the read channel panel, set the memory map data width and the stream data width to 64. Set the max burst size to 16. The write channel can be left as AUTO. Make sure no Allow Unaligned Transfers boxes are enabled
- Click Run Connection Automation



Designer Assistance available. Run Connection Automation

Hardware design
Block design (Cont.)

- Select S_AXI_LITE


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 3 selecte
  - [✓] axi_dma_0
    - [✓] S_AXI_LITE
  - [ ] zynq_ultra_ps_e_0
    - [ ] S_AXI_HP0_FPD
    - [ ] S_AXI_HP2_FPD

Description

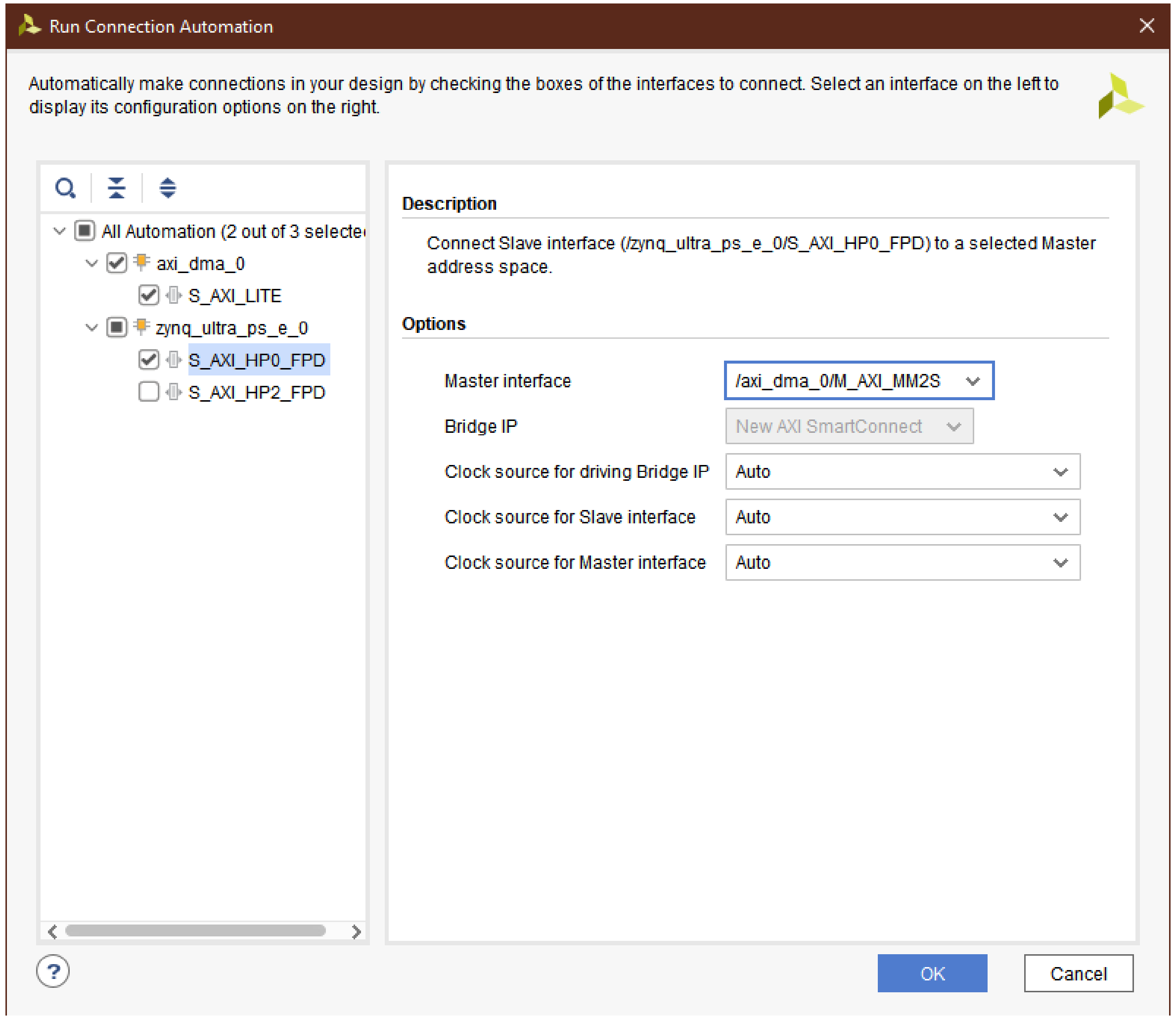
Connect Slave interface (/axi_dma_0/S_AXI_LITE) to a selected Master address space.

Options

Master interface	/zynq_ultra_ps_e_0/M_AXI_HPM0_FPD
Bridge IP	New AXI Interconnect
Clock source for driving Bridge IP	Auto
Clock source for Slave interface	Auto
Clock source for Master interface	Auto

Hardware design Block design (Cont.)

- Select S_AXI_LITE
- Select S_AXI_HP0_FPD and set the master interface to M_AXI_MM2S



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 3 selecte
  - [✓] axi_dma_0
    - [✓] S_AXI_LITE
  - zynq_ultra_ps_e_0
    - [✓] S_AXI_HP0_FPD
    - [ ] S_AXI_HP2_FPD

Description

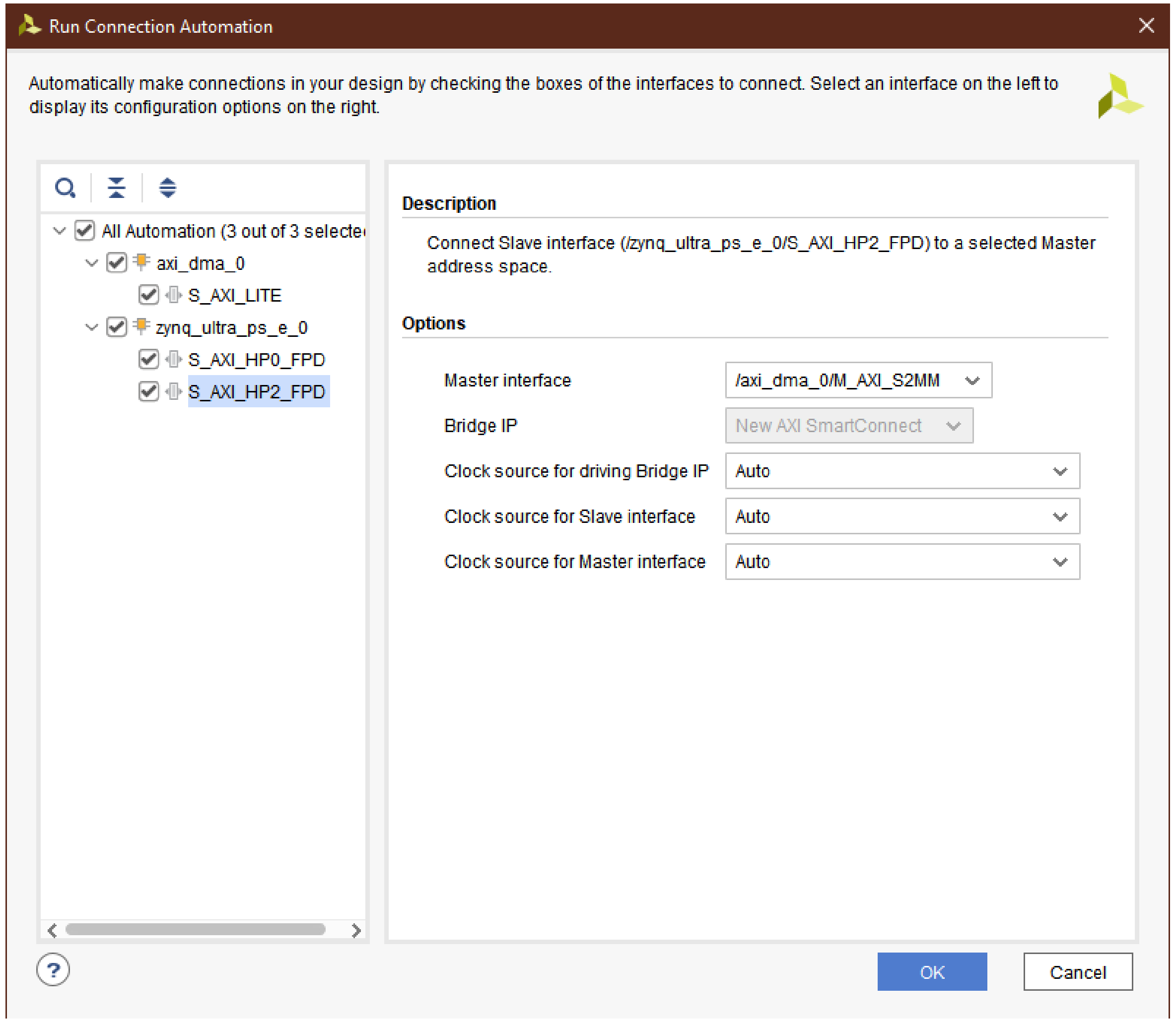
Connect Slave interface (/zynq_ultra_ps_e_0/S_AXI_HP0_FPD) to a selected Master address space.

Options

Master interface	/axi_dma_0/M_AXI_MM2S
Bridge IP	New AXI SmartConnect
Clock source for driving Bridge IP	Auto
Clock source for Slave interface	Auto
Clock source for Master interface	Auto

Hardware design Block design (Cont.)

- Select S_AXI_LITE
- Select S_AXI_HP0_FPD and set the master interface to M_AXI_MM2S
- Select S_AXI_HP2_FPD and set the master interface to M_AXI_S2MM


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 selecte
  - [✓] axi_dma_0
    - [✓] S_AXI_LITE
  - [✓] zynq_ultra_ps_e_0
    - [✓] S_AXI_HP0_FPD
    - [✓] S_AXI_HP2_FPD

Description

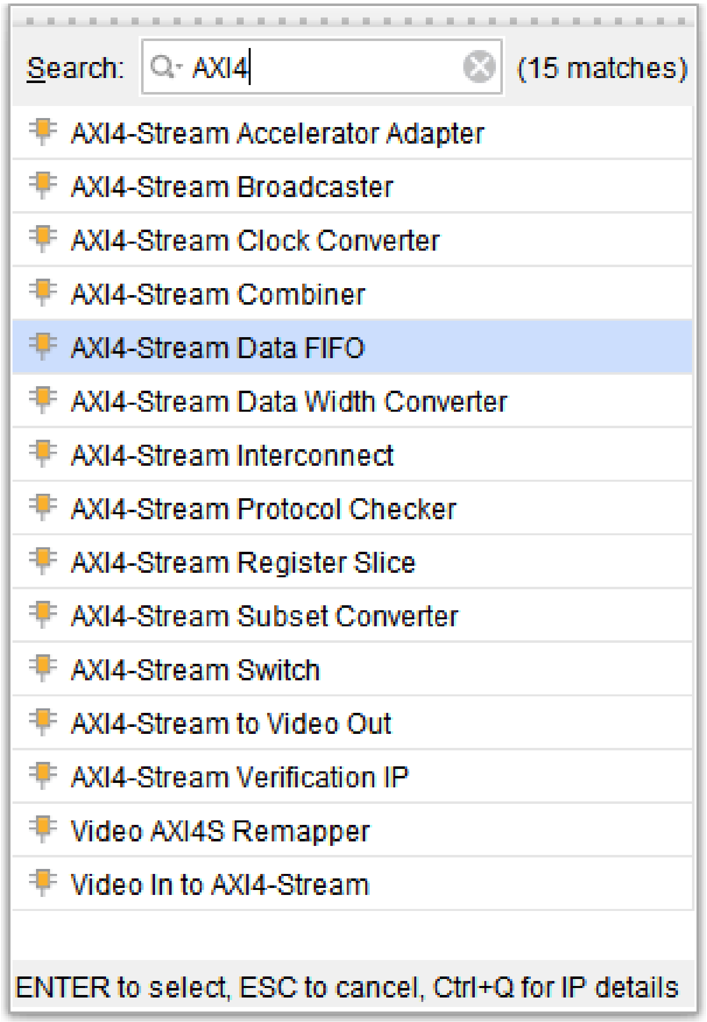
Connect Slave interface (/zynq_ultra_ps_e_0/S_AXI_HP2_FPD) to a selected Master address space.

Options

Master interface	/axi_dma_0/M_AXI_S2MM
Bridge IP	New AXI SmartConnect
Clock source for driving Bridge IP	Auto
Clock source for Slave interface	Auto
Clock source for Master interface	Auto

Hardware design Block design (Cont.)

- Select S_AXI_LITE
- Select S_AXI_HP0_FPD and set the master interface to M_AXI_MM2S
- Select S_AXI_HP2_FPD and set the master interface to M_AXI_S2MM
- Add AXI4-Stream Data FIFO to the design

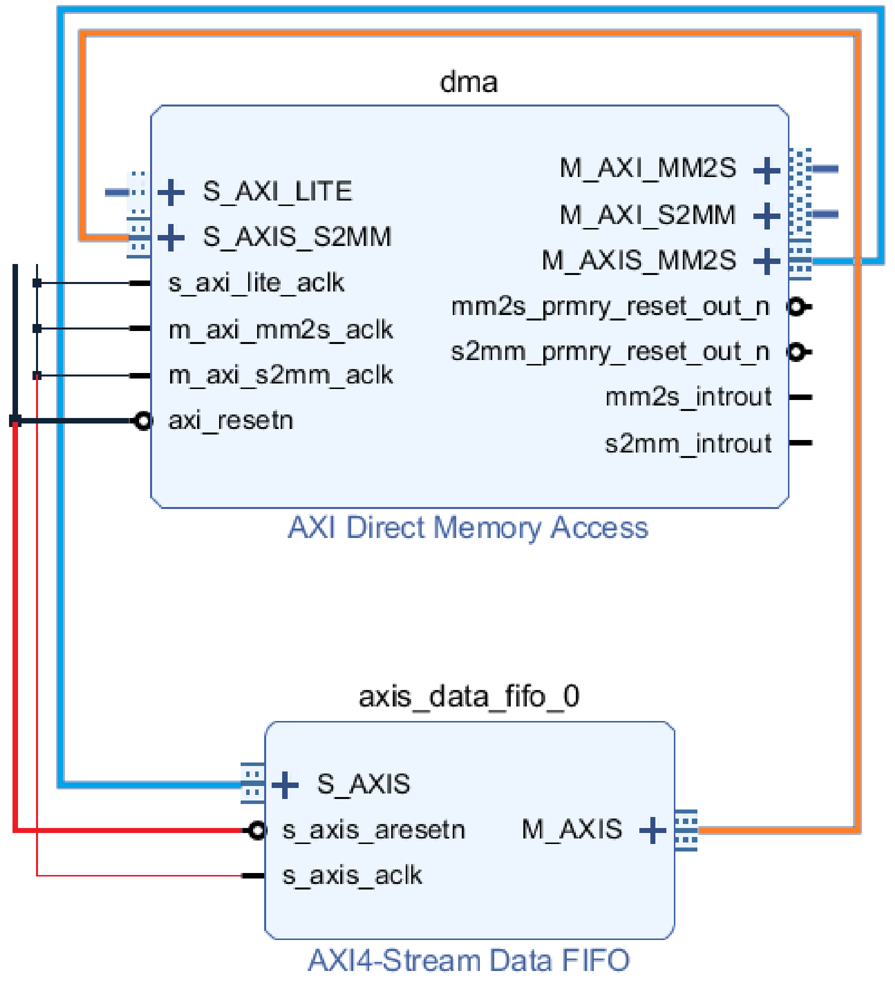

Search: AXI4 (15 matches)

AXI4-Stream Accelerator Adapter
AXI4-Stream Broadcaster
AXI4-Stream Clock Converter
AXI4-Stream Combiner
AXI4-Stream Data FIFO
AXI4-Stream Data Width Converter
AXI4-Stream Interconnect
AXI4-Stream Protocol Checker
AXI4-Stream Register Slice
AXI4-Stream Subset Converter
AXI4-Stream Switch
AXI4-Stream to Video Out
AXI4-Stream Verification IP
Video AXI4S Remapper
Video In to AXI4-Stream

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

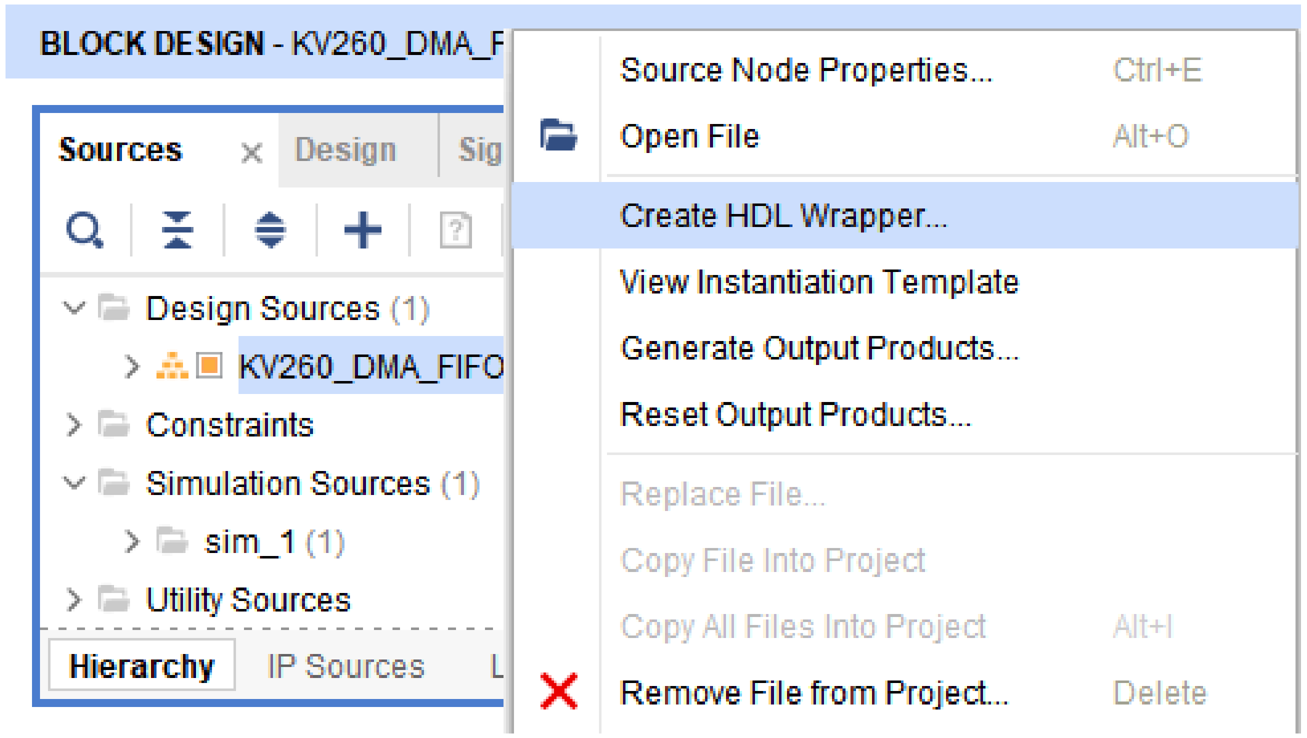
Hardware design
Block design (Cont.)

- Select S_AXI_LITE
- Select S_AXI_HP0_FPD and set the master interface to M_AXI_MM2S
- Select S_AXI_HP2_FPD and set the master interface to M_AXI_S2MM
- Add AXI4-Stream Data FIFO to the design
- Make the following connection:
  - S_AXIS → M_AXIS_MM2S
  - M_AXIS → S_AXIS_S2MM
  - s_axis_aclk → pl_clk0
  - s_axis_aresetn → axi_resetn



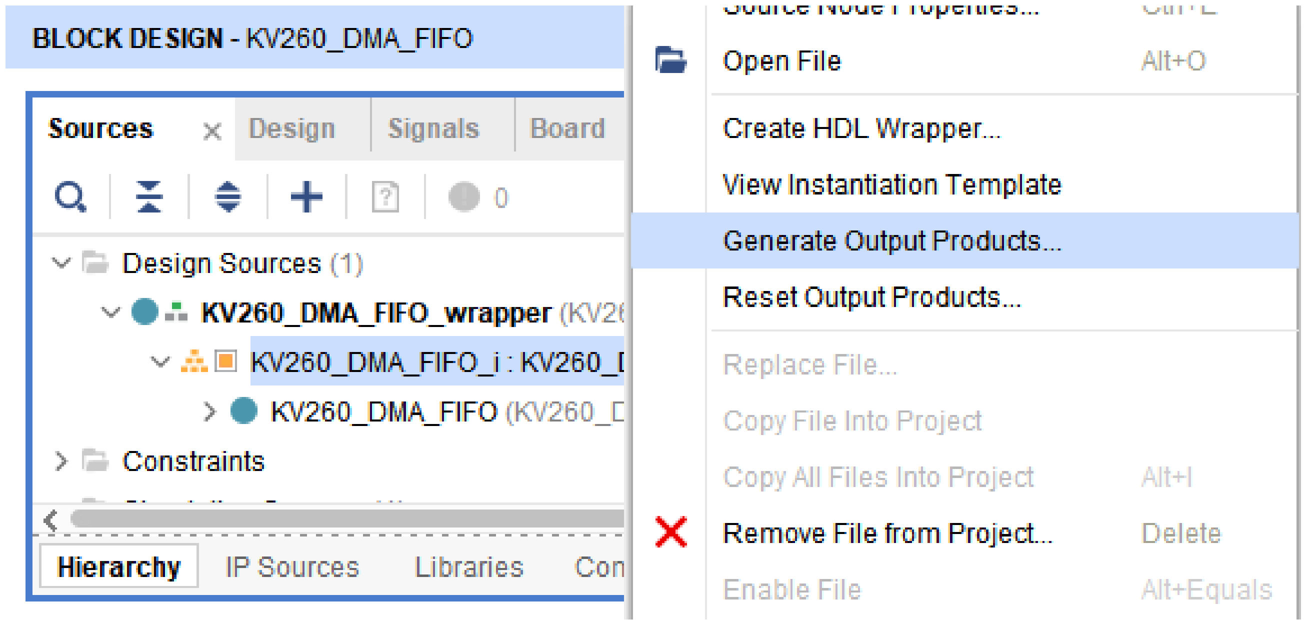
Hardware design Block design (Cont.)

- Select S_AXI_LITE
- Select S_AXI_HP0_FPD and set the master interface to M_AXI_MM2S
- Select S_AXI_HP2_FPD and set the master interface to M_AXI_S2MM
- Add AXI4-Stream Data FIFO to the design
- Make the following connection:
    - S_AXIS → M_AXIS_MM2S
    - M_AXIS → S_AXIS_S2MM
    - s_axis_aclk → pl_clk0
    - s_axis_aresetn → axi_resetn
- Finally, create HDL wrapper



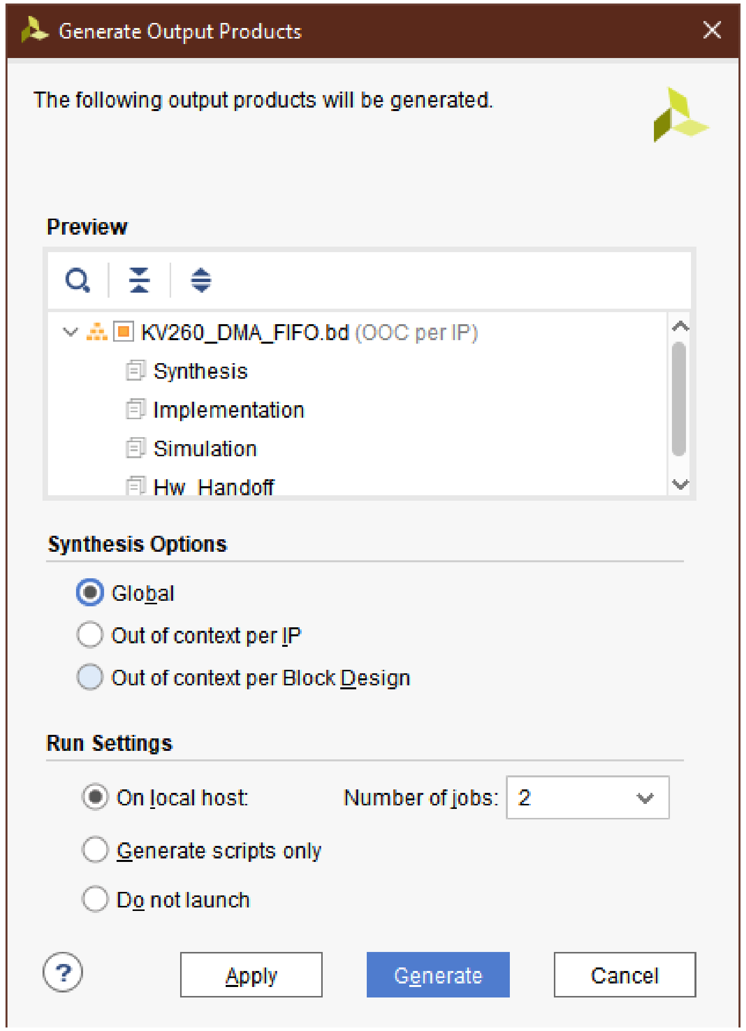
Hardware design
Block design (Cont.)

- Select S_AXI_LITE
- Select S_AXI_HP0_FPD and set the master interface to M_AXI_MM2S
- Select S_AXI_HP2_FPD and set the master interface to M_AXI_S2MM
- Add AXI4-Stream Data FIFO to the design
- Make the following connection:
  - S_AXIS → M_AXIS_MM2S
  - M_AXIS → S_AXIS_S2MM
  - s_axis_aclk → pl_clk0
  - s_axis_aresetn → axi_resetn
- Finally, create HDL wrapper
- Generate output product, set the synthesis option to global



Hardware design Block design (Cont.)

- Select S_AXI_LITE
- Select S_AXI_HP0_FPD and set the master interface to M_AXI_MM2S
- Select S_AXI_HP2_FPD and set the master interface to M_AXI_S2MM
- Add AXI4-Stream Data FIFO to the design
- Make the following connection:
    - S_AXIS → M_AXIS_MM2S
    - M_AXIS → S_AXIS_S2MM
    - s_axis_aclk → pl_clk0
    - s_axis_aresetn → axi_resetn
- Finally, create HDL wrapper
- Generate output product, set the synthesis option to global


Generate Output Products

The following output products will be generated.

Preview

KV260_DMA_FIFO.bd (OOC per IP)
- Synthesis
- Implementation
- Simulation
- Hw_Handoff

Synthesis Options

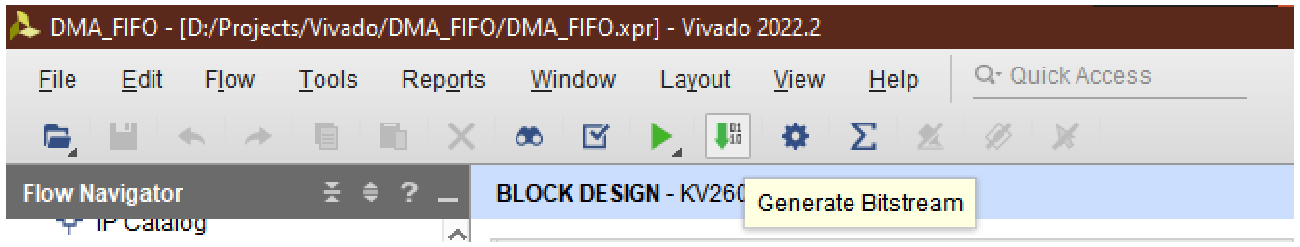
[✓ Global]
Out of context per IP
Out of context per Block Design

Run Settings

[✓ On local host:] Number of jobs: 2
Generate scripts only
Do not launch

Hardware design
Block design (Cont.)

- Select S_AXI_LITE
- Select S_AXI_HP0_FPD and set the master interface to M_AXI_MM2S
- Select S_AXI_HP2_FPD and set the master interface to M_AXI_S2MM
- Add AXI4-Stream Data FIFO to the design
- Make the following connection:
  - S_AXIS → M_AXIS_MM2S
  - M_AXIS → S_AXIS_S2MM
  - s_axis_aclk → pl_clk0
  - s_axis_aresetn → axi_resetn
- Finally, create HDL wrapper
- Generate output product, set the synthesis option to global
- Then, generate bitstream



Hardware design
File location

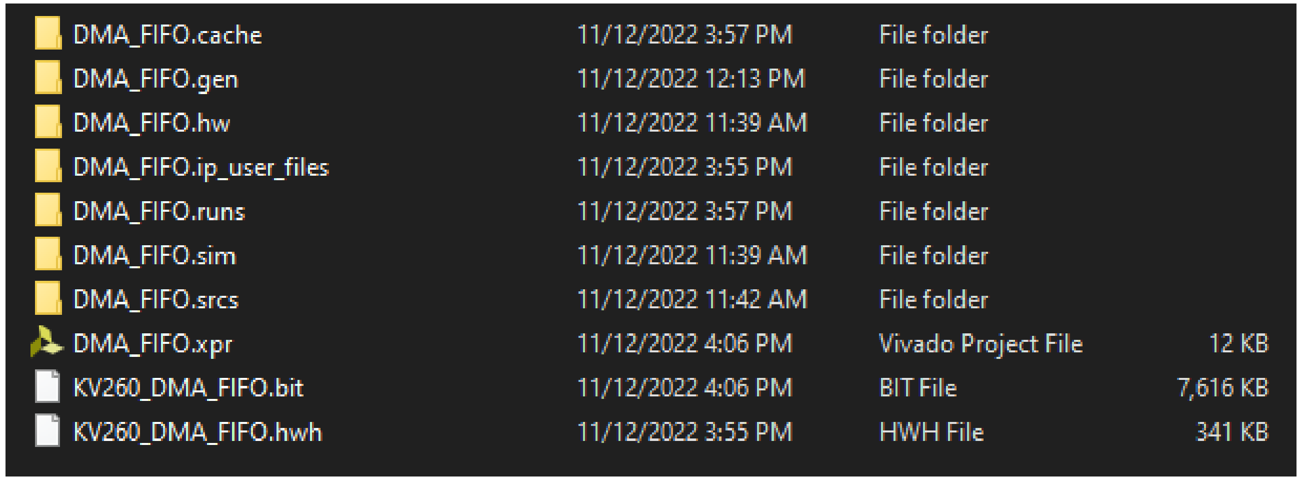
- You’ll need 2 files, one will be the hwh file located here

“DMA_FIFO\DMA_FIFO.gen\sources_1\bd\KV260_DMA_FIFO\hw_handoff”

- And the other is the bit stream file, located here

“DMA_FIFO\DMA_FIFO.runs\impl_1”

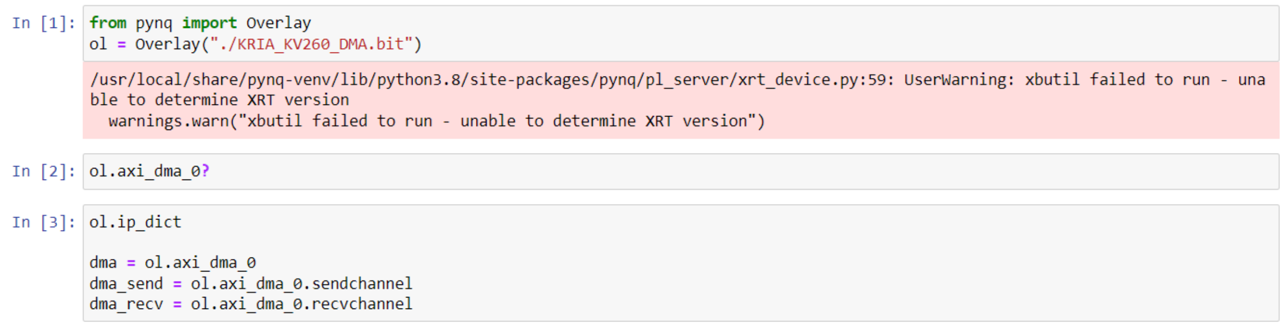
- Move them to the global folder and make sure they have the same name



DMA_FIFO.cache	11/12/2022 3:57 PM	File folder	
DMA_FIFO.gen	11/12/2022 12:13 PM	File folder	
DMA_FIFO.hw	11/12/2022 11:39 AM	File folder	
DMA_FIFO.ip_user_files	11/12/2022 3:55 PM	File folder	
DMA_FIFO.runs	11/12/2022 3:57 PM	File folder	
DMA_FIFO.sim	11/12/2022 11:39 AM	File folder	
DMA_FIFO.srcs	11/12/2022 11:42 AM	File folder	
DMA_FIFO.xpr	11/12/2022 4:06 PM	Vivado Project File	12 KB
KV260_DMA_FIFO.bit	11/12/2022 4:06 PM	BIT File	7,616 KB
KV260_DMA_FIFO.hwh	11/12/2022 3:55 PM	HWH File	341 KB

Using the DMA

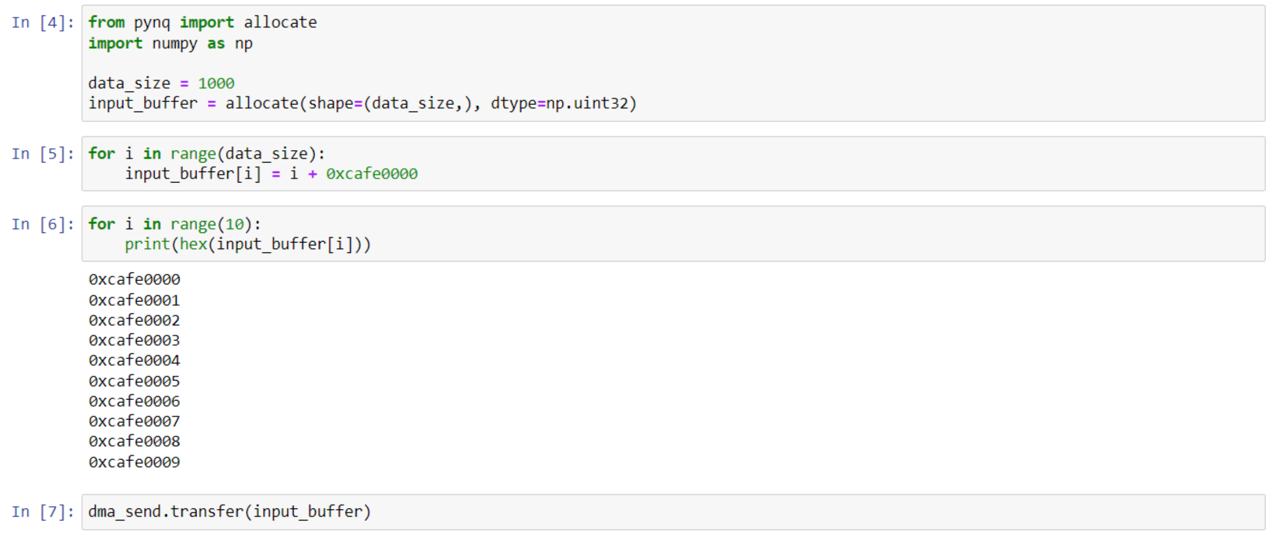
- The following segment can be used to test the functionality of the design.
- The test is very simple, consists of sending an array of data to the DMA through the send channel and receiving the same data through the receive channel
- Make sure to free the memory buffers to avoid memory leaks.



```
In [1]: from pynq import Overlay
        ol = Overlay("./KRIA_KV260_DMA.bit")

/usr/local/share/pynq-venv/lib/python3.8/site-packages/pynq/pl_server/xrt_device.py:59: UserWarning: xbutil failed to run - una
ble to determine XRT version
  warnings.warn("xbutil failed to run - unable to determine XRT version")

In [2]: ol.axi_dma_0?

In [3]: ol.ip_dict

        dma = ol.axi_dma_0
        dma_send = ol.axi_dma_0.sendchannel
        dma_recv = ol.axi_dma_0.recvchannel
```

Using the DMA

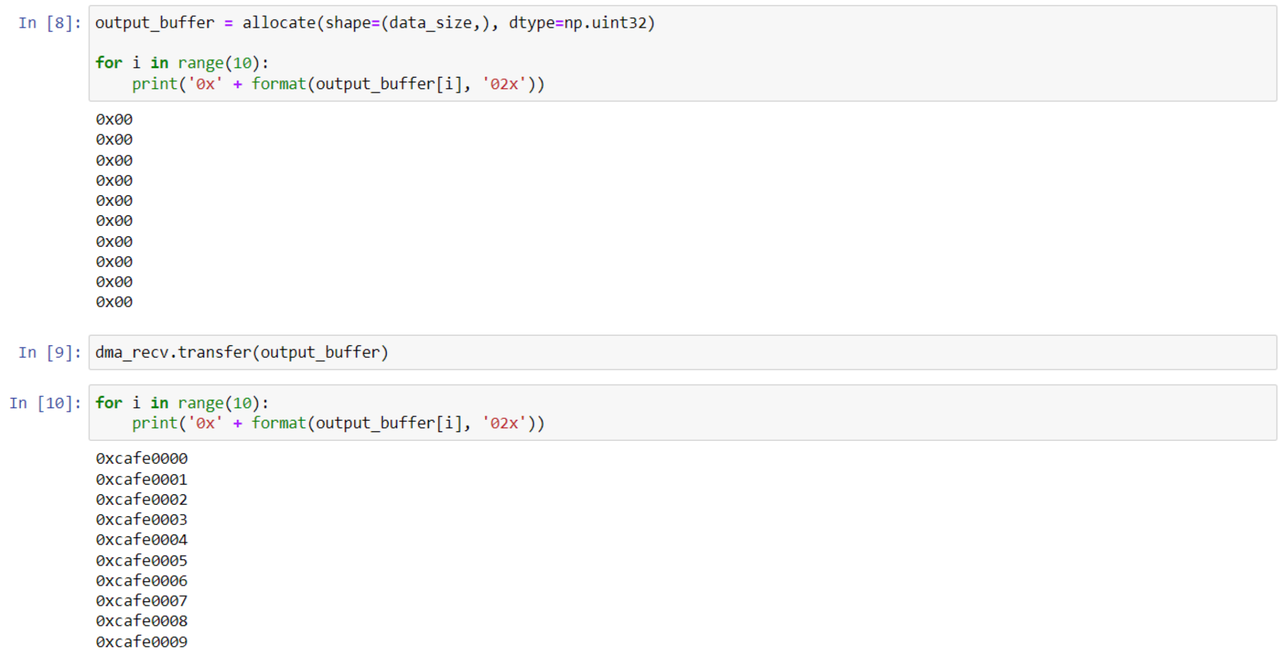
- The following segment can be used to test the functionality of the design.
- The test is very simple, consists of sending an array of data to the DMA through the send channel and receiving the same data through the receive channel
- Make sure to free the memory buffers to avoid memory leaks.



```
In [4]: from pynq import allocate
        import numpy as np

        data_size = 1000
        input_buffer = allocate(shape=(data_size,), dtype=np.uint32)

In [5]: for i in range(data_size):
            input_buffer[i] = i + 0xcafe0000

In [6]: for i in range(10):
            print(hex(input_buffer[i]))

        0xcafe0000
        0xcafe0001
        0xcafe0002
        0xcafe0003
        0xcafe0004
        0xcafe0005
        0xcafe0006
        0xcafe0007
        0xcafe0008
        0xcafe0009

In [7]: dma_send.transfer(input_buffer)
```

Using the DMA

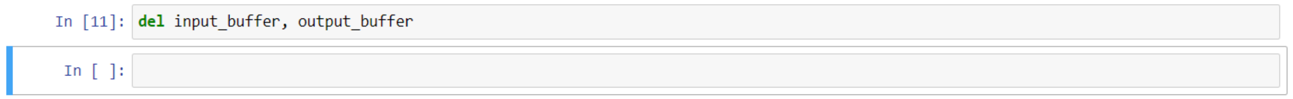
- The following segment can be used to test the functionality of the design.
- The test is very simple, consists of sending an array of data to the DMA through the send channel and receiving the same data through the receive channel
- Make sure to free the memory buffers to avoid memory leaks.



```
In [8]: output_buffer = allocate(shape=(data_size,), dtype=np.uint32)

        for i in range(10):
            print('0x' + format(output_buffer[i], '02x'))

0x00
0x00
0x00
0x00
0x00
0x00
0x00
0x00
0x00
0x00

In [9]: dma_recv.transfer(output_buffer)

In [10]: for i in range(10):
             print('0x' + format(output_buffer[i], '02x'))

0xcafe0000
0xcafe0001
0xcafe0002
0xcafe0003
0xcafe0004
0xcafe0005
0xcafe0006
0xcafe0007
0xcafe0008
0xcafe0009
```

Using the DMA

- The following segment can be used to test the functionality of the design.
- The test is very simple, consists of sending an array of data to the DMA through the send channel and receiving the same data through the receive channel
- Make sure to free the memory buffers to avoid memory leaks.



```
In [11]: del input_buffer, output_buffer

In [ ]:
```