

DPU on PYNQ-Z2 (1)

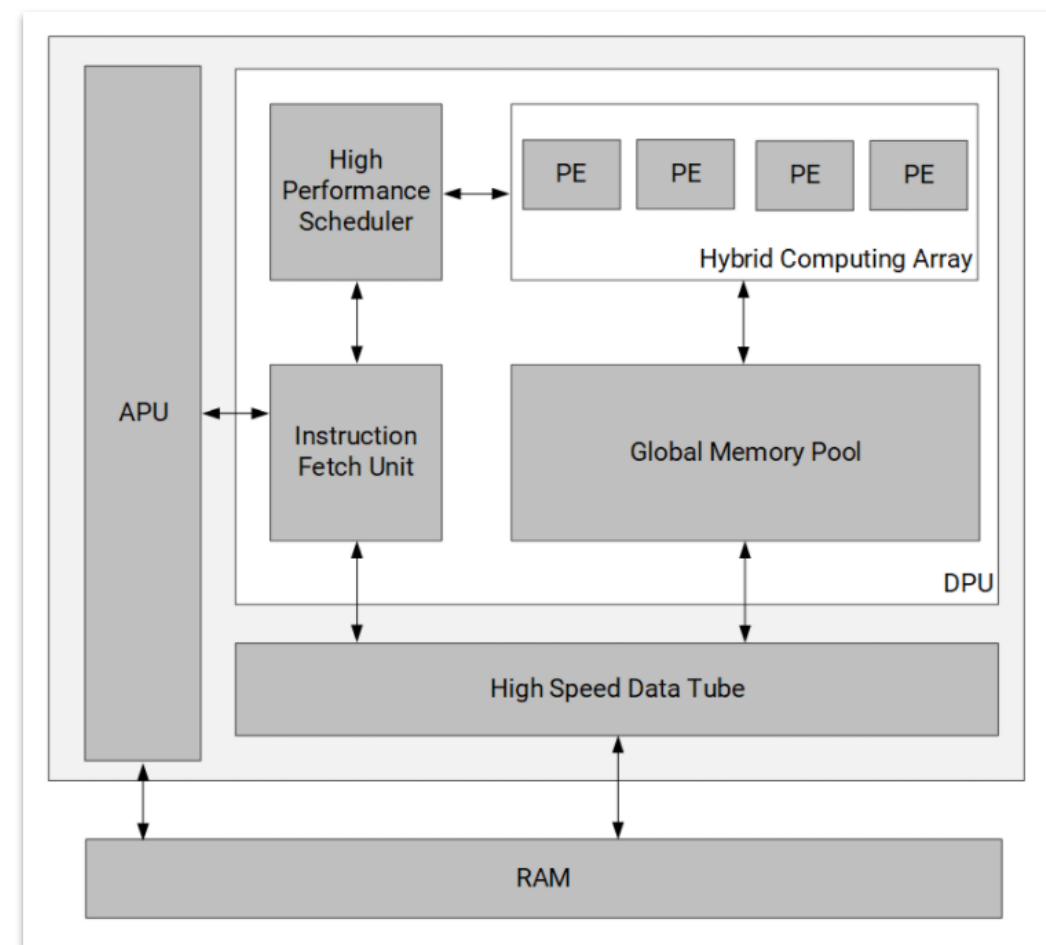
Vivado Project

Outline

- DPU Introduce
- Vivado project Build

DPU Introduction

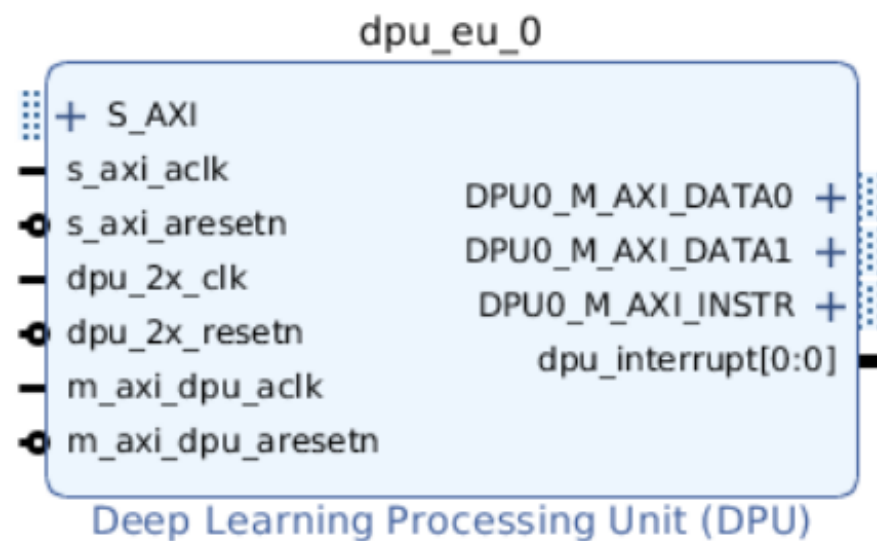
- The Xilinx® Deep Learning Processing Unit (DPU) is a configurable computation engine optimized for convolutional neural networks.
- It includes a set of highly optimized instructions, and supports most convolutional neural networks, such as VGG, ResNet, GoogLeNet, YOLO, SSD, MobileNet, FPN, and others.
- The DPU requires instructions to implement a neural network and accessible memory locations for input images as well as temporary and output data. A program running on the application processing unit (APU) is also required to service interrupts and coordinate data transfers.



DPU Top-Level Block Diagram

DPU Introduction

- The DPU has the following features:
 - One AXI slave interface for accessing configuration and status registers.
 - One AXI master interface for accessing instructions.
 - Supports configurable AXI master interface with 64 or 128 bits for accessing data depending on the target device.
 - Some highlights of DPU functionality include:
 - Configurable hardware architecture core includes: B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096
 - Maximum of three homogeneous cores
 - Convolution and deconvolution
 - Depthwise convolution
 - Max pooling
 - Average pooling
 - ReLU, ReLU6, and Leaky ReLU
 - Concat
 - Elementwise-Sum
 - Dilation
 - Reorg
 - Fully connected layer
 - Softmax (Not Support at Zynq-7000)
 - Batch Normalization
 - Split



DPU IO ports

DPU Introduction

Table 1: DPU Signal Description

Signal Name	Interface Type	Width	I/O	Description
S_AXI	Memory mapped AXI slave interface	32	I/O	32-bit memory mapped AXI interface for registers.
s_axi_aclk	Clock	1	I	AXI clock input for S_AXI
s_axi_aresetn	Reset	1	I	Active-Low reset for S_AXI
dpu_2x_clk	Clock	1	I	Input clock used for DSP blocks in the DPU. The frequency is twice that of m_axi_dpu_aclk.
dpu_2x_resetn	Reset	1	I	Active-Low reset for DSP blocks
m_axi_dpu_aclk	Clock	1	I	Input clock used for DPU general logic.
m_axi_dpu_aresetn	Reset	1	I	Active-Low reset for DPU general logic
DPUx_M_AXI_INSTR	Memory mapped AXI master interface	32	I/O	32-bit memory mapped AXI interface for DPU instructions.
DPUx_M_AXI_DATA0	Memory mapped AXI master interface	128	I/O	128-bit memory mapped AXI interface for DPU data.
DPUx_M_AXI_DATA1	Memory mapped AXI master interface	128	I/O	128-bit memory mapped AXI interface for DPU data.
dpu_interrupt	Interrupt	1~3	O	Active-High interrupt output from DPU. The data width is determined by the number of DPU cores.
SFM_M_AXI (optional)	Memory mapped AXI master interface	128	I/O	128-bit memory mapped AXI interface for softmax data.
sfm_interrupt (optional)	Interrupt	1	O	Active-High interrupt output from softmax module.
dpu_2x_clk_ce (optional)	Clock enable	1	O	Clock enable signal for controlling the input DPU 2x clock when DPU 2x clock gating is enabled.

DPU Introduction

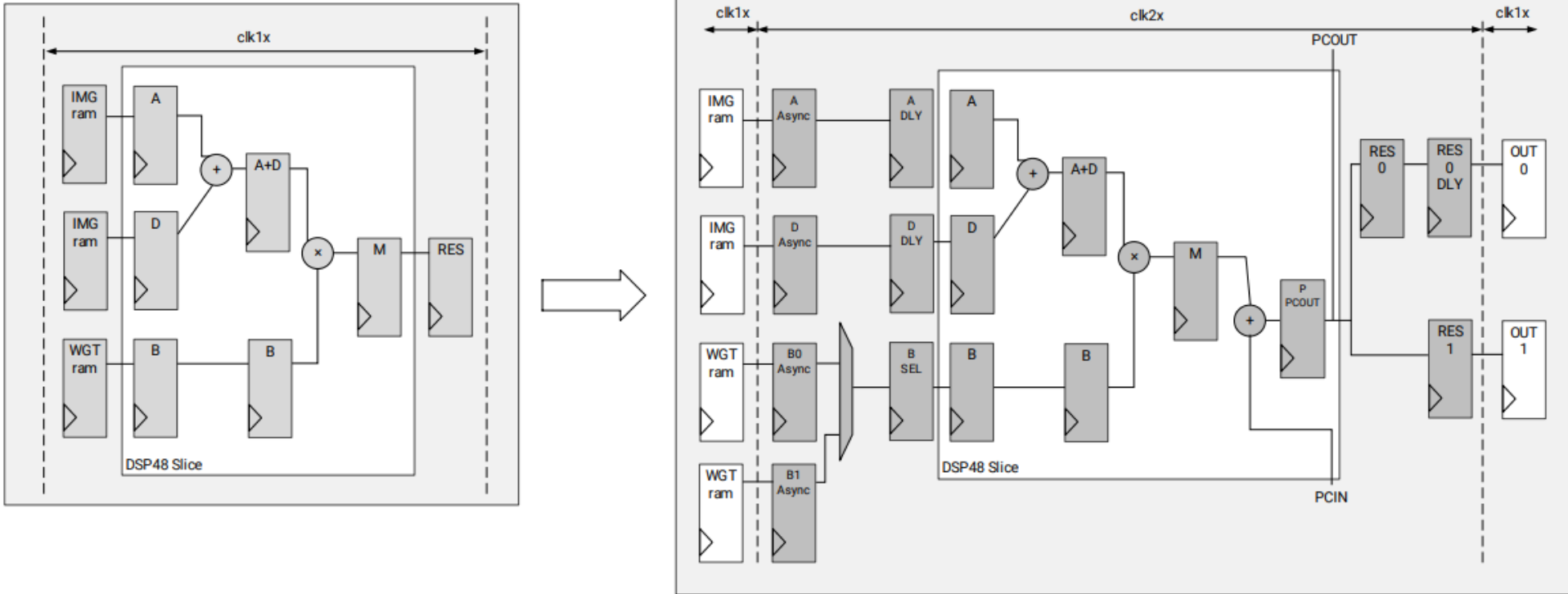
Table 7: Deep Neural Network Features and Parameters Supported by the DPU

Features	Description	
Convolution	Kernel Sizes	W: 1–16 H: 1–16
	Strides	W: 1–4 H:1–4
	Padding_w	1: kernel_w - 1
	Padding_h	1: kernel_h - 1
	Input Size	Arbitrary
	Input Channel	1–256 * channel_parallel
	Output Channel	1–256 * channel_parallel
	Activation	ReLU, LeakyReLU, or ReLU6
	Dilation	dilation * input_channel <= 256 * channel_parallel && stride_w == 1 && stride_h == 1
Depthwise Convolution	Kernel Sizes	W: 1–16 H: 1–16
	Strides	W: 1–4 H:1–4
	Padding_w	1: kernel_w - 1
	Padding_h	1: kernel_h - 1
	Input Size	Arbitrary
	Input Channel	1–256 * channel_parallel
	Output Channel	1–256 * channel_parallel
	Activation	ReLU or ReLU6
	Dilation	dilation * input_channel <= 256 * channel_parallel && stride_w == 1 && stride_h == 1
Deconvolution	Kernel Sizes	W: 1–16 H: 1–16

	Stride_w	stride_w * output_channel <= 256 * channel_parallel
	Stride_h	Arbitrary
	Padding_w	1: kernel_w - 1
	Padding_h	1: kernel_h - 1
	Input Size	Arbitrary
	Input Channel	1–256 * channel_parallel
	Output Channel	1–256 * channel_parallel
	Activation	ReLU or LeakyReLU
	Max Pooling	Kernel Sizes
Strides		W: 1–4 H:1–4
Padding		W: 1–4 H: 1–4
Elementwise-sum	Input channel	1–256 * channel_parallel
	Input size	Arbitrary
Concat	Output channel	1–256 * channel_parallel
Reorg	Strides	stride * stride * input_channel <= 256 * channel_parallel
FC	Input_channel	Input_channel <= 2048 * channel_parallel
	Output_channel	Arbitrary

DPU Introduction- DPU with Enhanced Usage of DSP

- DSP Double Data Rate (DDR) technique is used to improve the performance achieved with the device.
- Therefore, two input clocks for the DPU are needed, one for general logic, and the other for DSP slices.



Difference between DPU without DSP DDR and DPU Enhanced Usage

DPU Introduction-DPU Convolution Architecture

Table 8: Parallelism for Different Convolution Architectures

Convolution Architecture	Pixel Parallelism (PP)	Input Channel Parallelism (ICP)	Output Channel Parallelism (OCP)	Peak Ops (operations/per clock)
B512	4	8	8	512
B800	4	10	10	800
B1024	8	8	8	1024
B1152	4	12	12	1150
B1600	8	10	10	1600
B2304	8	12	12	2304
B3136	8	14	14	3136
B4096	8	16	16	4096

In each clock cycle, the convolution array performs a multiplication and an accumulation, which are counted as two operations.

Thus, the peak number of operations per cycle is equal to $PP \cdot ICP \cdot OCP \cdot 2$.

DPU Introduction-RAM Usage

Table 9: Number of BRAM36K Blocks in Different Architectures for Each DPU Core

DPU Architecture	Low RAM Usage	High RAM Usage
B512 (4x8x8)	73.5	89.5
B800 (4x10x10)	91.5	109.5
B1024 (8x8x8)	105.5	137.5
B1152 (4x12x12)	123	145
B1600 (8x10x10)	127.5	163.5
B2304 (8x12x12)	167	211
B3136 (8x14x14)	210	262
B4096 (8x16x16)	257	317.5

DPU Introduction-Channel Augmentation

- Channel augmentation is an optional feature for improving the efficiency of the DPU when handling input channels much lower than the available channel parallelism.
- When the number of input channels is larger than the channel parallelism, then enabling channel augmentation will not make a difference
- In summary, the channel augmentation can improve the total efficiency for most CNNs, but it will cost extra logic resources.

Table 10: Extra LUTs of DPU with Channel Augmentation

DPU Architecture	Extra LUTs with Channel Augmentation
B1024 (8x8x8)	2670
B1152 (4x12x12)	2189
B4096 (8x16x16)	1550
B512 (4x8x8)	1475
B800 (4x10x10)	2796
B1600 (8x10x10)	2832
B2304 (8x12x12)	1697
B3136 (8x14x14)	1899

DPU Introduction- DSP Usage

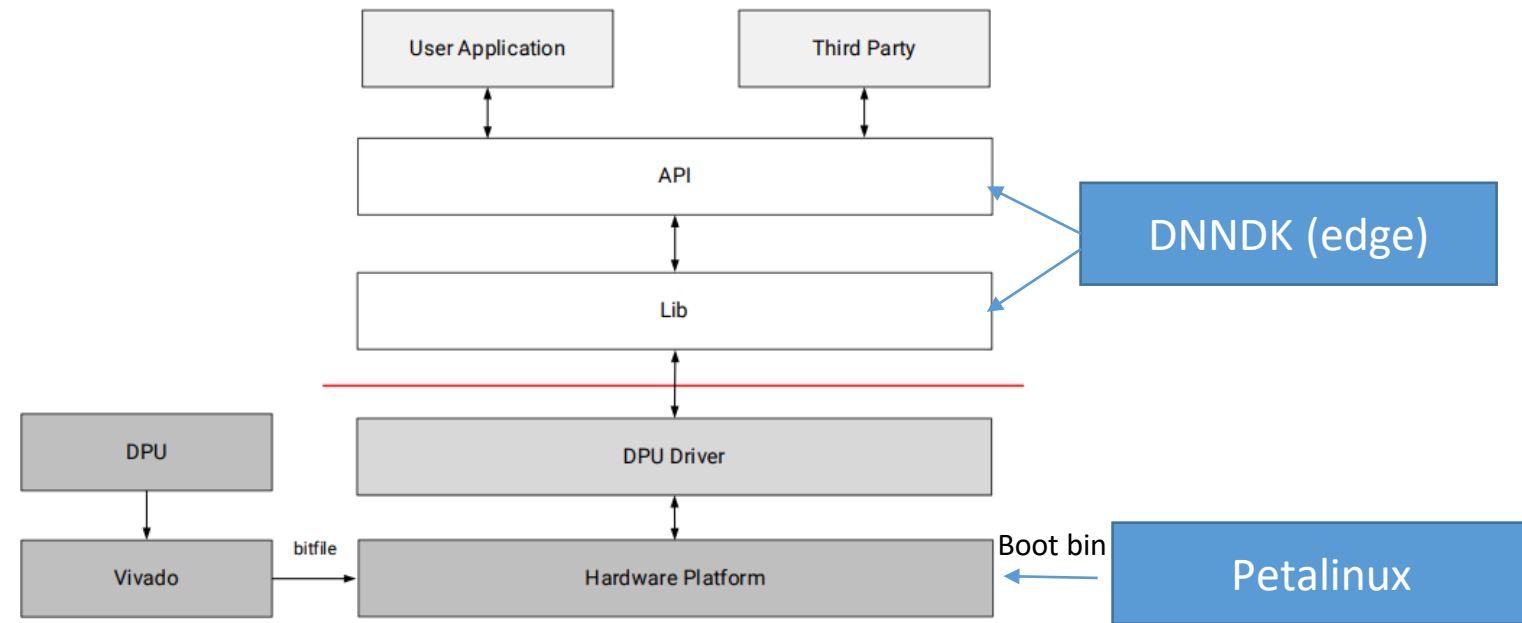
- This allows you to select whether DSP48E slices will be used for accumulation in the DPU convolution module.
- In low DSP usage mode, the DPU IP will use DSP slices only for multiplication in the convolution.
- In high DSP usage mode, the DSP slice will be used for both multiplication and accumulation.
- Thus, the high DSP usage consumes more DSP slices and less LUTs

Table 11: Resources for Different DSP Usage

High DSP Usage					Low DSP Usage				
Arch	LUT	Register	BRAM	DSP	Arch	LUT	Register	BRAM	DSP
B512	20055	28849	69.5	98	B512	21171	33572	69.5	66
B800	21490	34561	87	142	B800	22900	33752	87	102
B1024	24349	46241	101.5	194	B1024	26341	49823	101.5	130
B1152	23527	46906	117.5	194	B1152	25250	49588	117.5	146
B1600	26728	56267	123	282	B1600	29270	60739	123	202
B2304	39562	67481	161.5	386	B2304	32684	72850	161.5	290
B3136	32190	79867	203.5	506	B3136	35797	86132	203.5	394
B4096	37266	92630	249.5	642	B4096	41412	99791	249.5	514

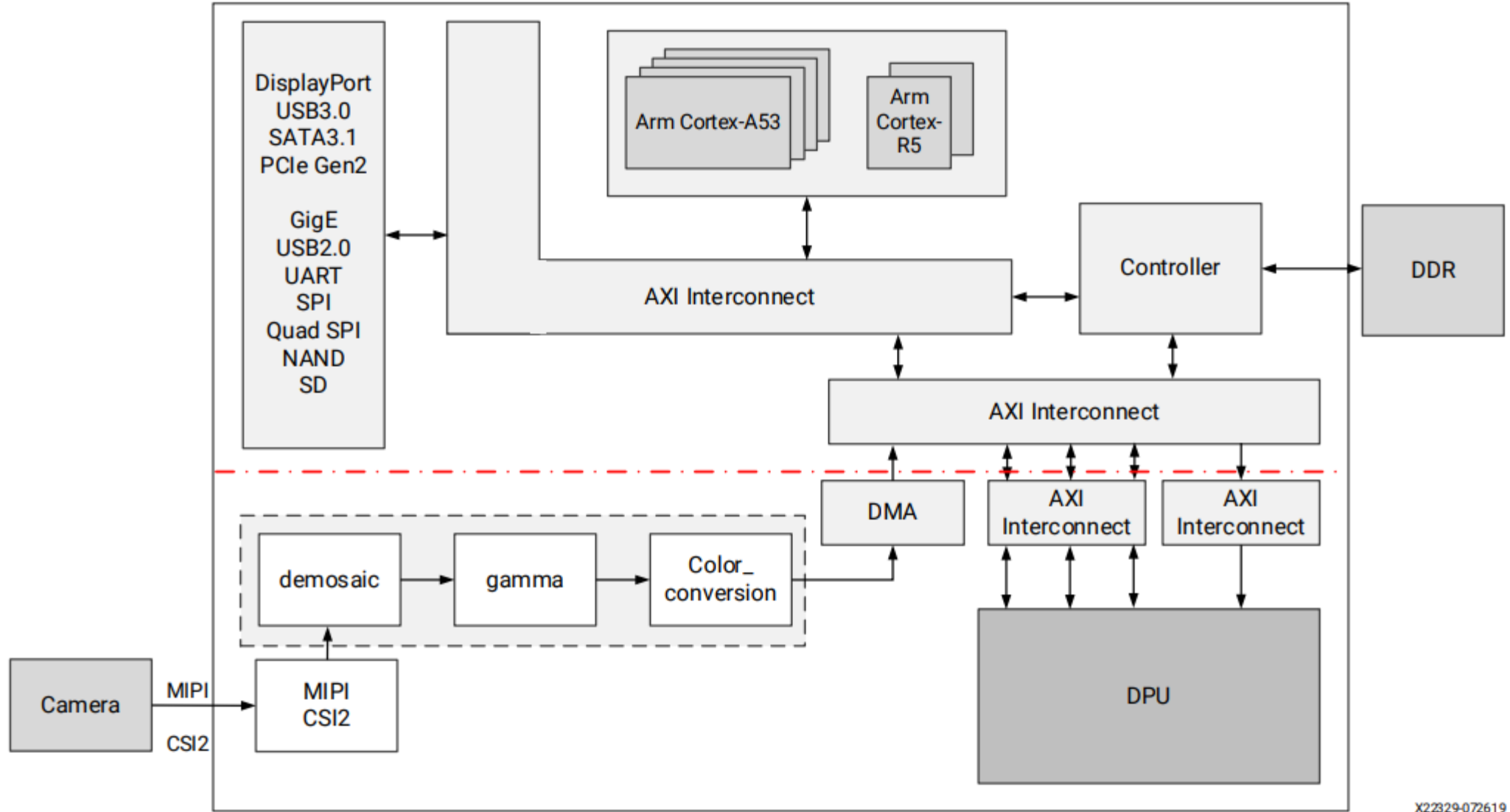
DPU Introduction- DPU Development Flow

- The DPU requires a device driver which is included in the Xilinx Deep Neural Network Development Kit (DNNDK) toolchain.
- Vivado to generate the bitstream. Then, download the bitstream to the target board and install the DPU driver.



Basic Development Flow

DPU Introduction- Example System with DPU

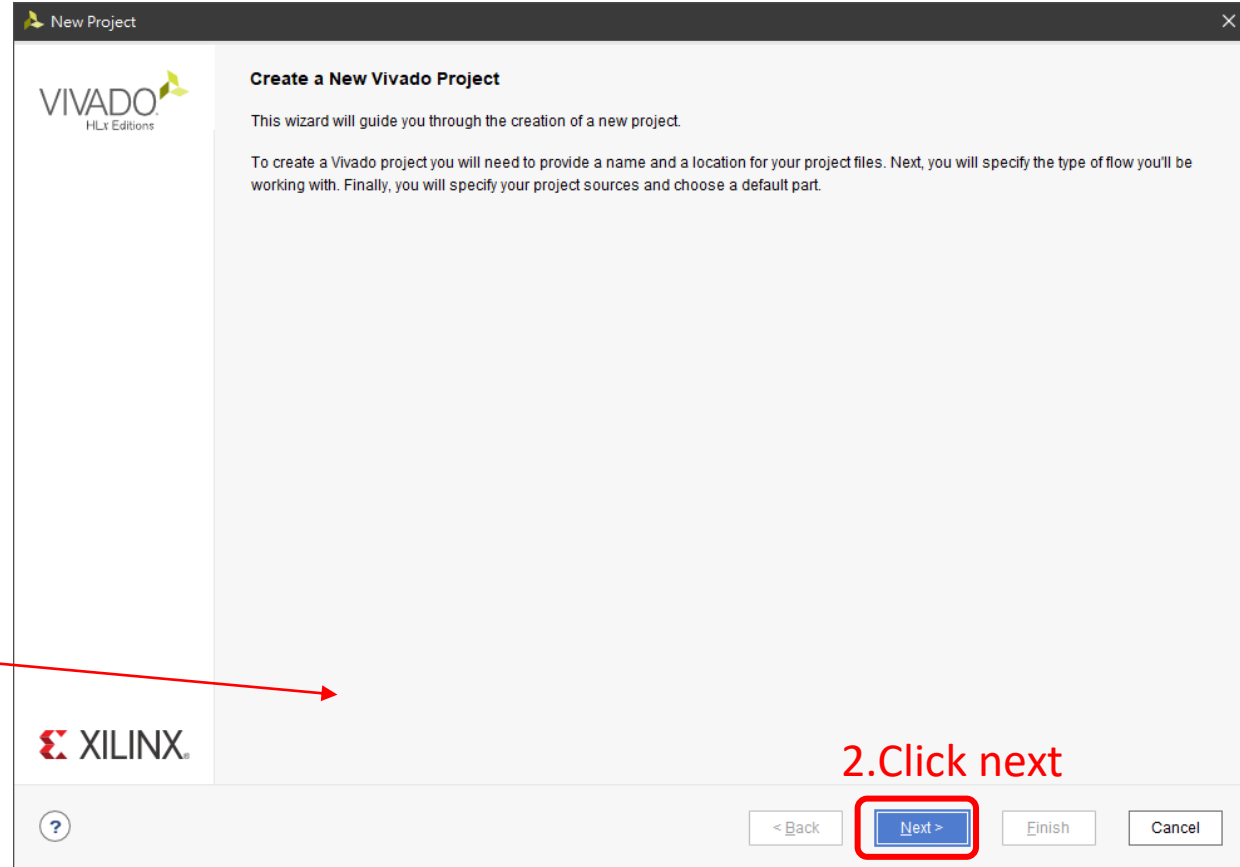
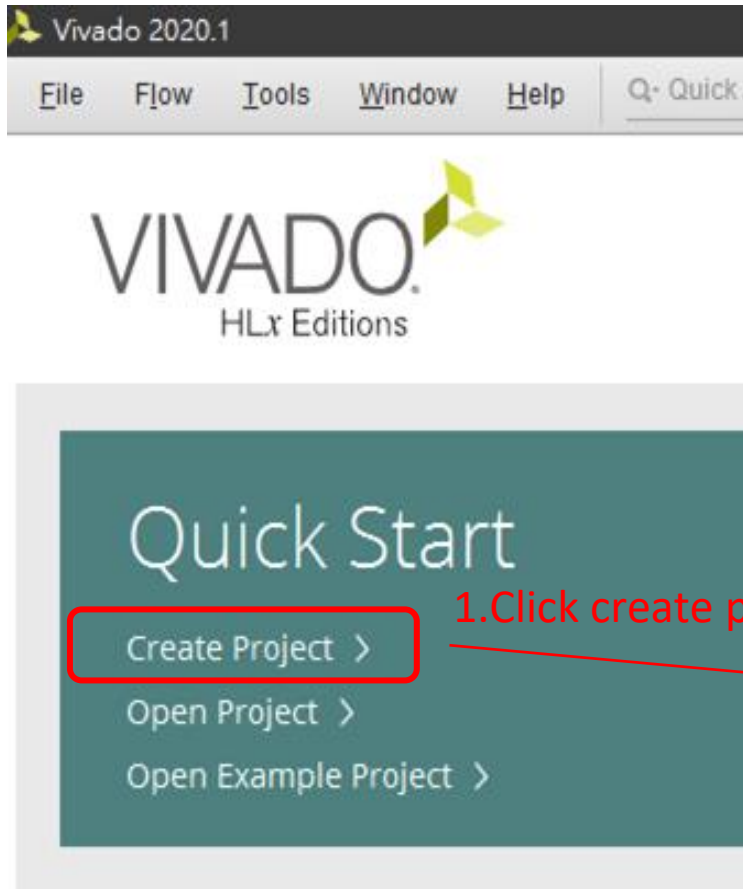


X22329-072619

Example System with DPU

Vivado Project Build

Open vivado and create new project



Change 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.

1.Setup project name and directory

Project name:

Project location:

Create project subdirectory

Project will be created at: D:/pynq2_dpu

2.Click next

Select RTL project and next

New Project

Project Type
Specify the type of project to create.

1. Select RTL Project and check Do not specify source at this time

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

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, XST or ISE Project File.

Example Project
Create a new Vivado project from a predefined template.

2. Click Next

? < Back **Next >** Finish Cancel

Select pynq-z2 Board File

New Project

Default Part
Choose a default Xilinx part or board for your project.

1. Click boards

2. Select pynq-z2





3. click next

Parts **Boards**

[Reset All Filters](#) Install/Update Boards

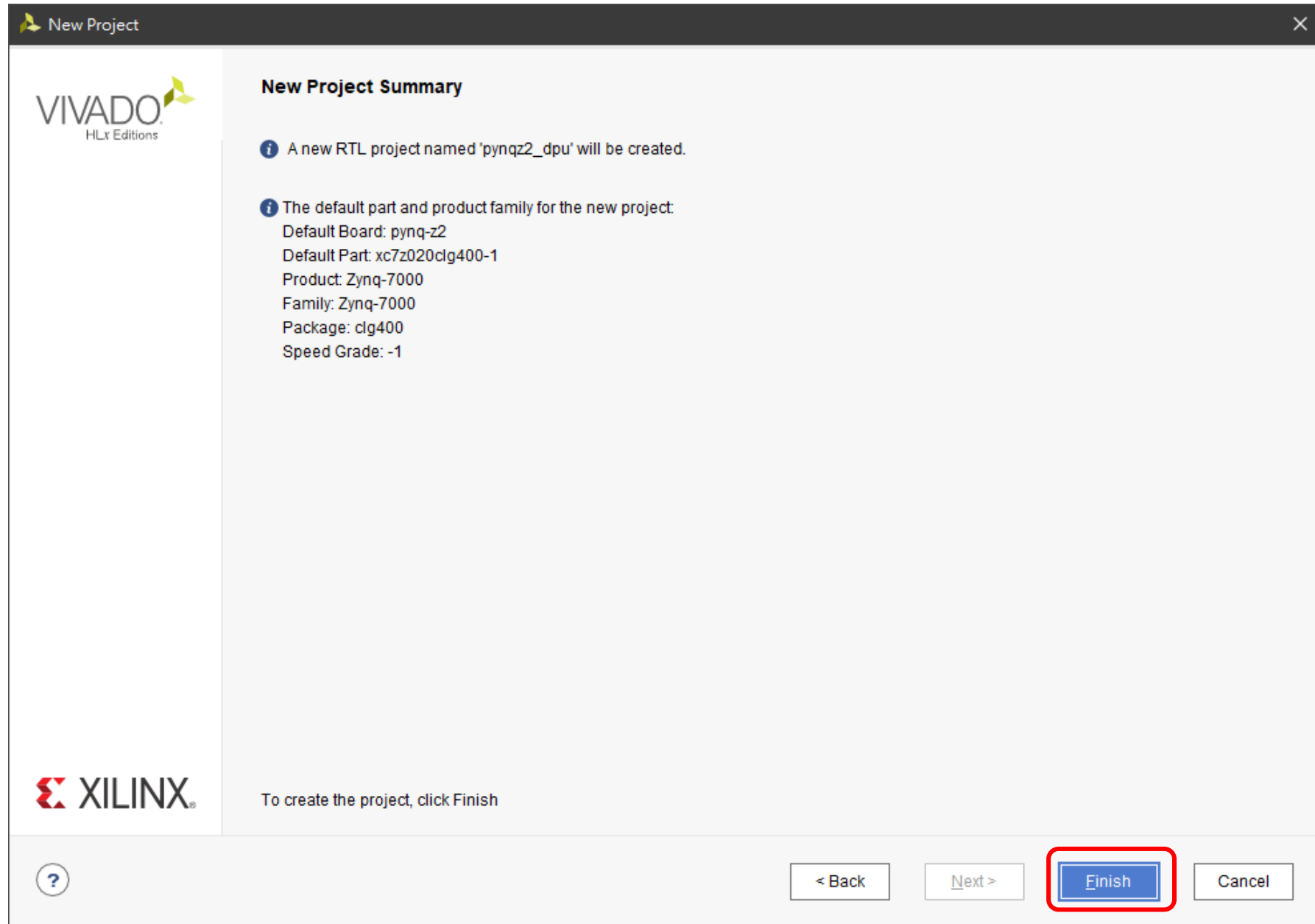
Vendor: All Name: All Board Rev: Latest

Search: Q

Display ...	Preview	Vendor	File Versi...	Part	I/O Pin C...	Board Rev	Available ...	LUT Ele...	FlipFlops	Block RA...	Ultra RAMs
ZedBoard Z... Add Daught...		em.avnet...	1.4	xc7z020c...	484	d	200	53200	106400	140	0
pynq-z2		tul.com.tw	1.0	xc7z020c...	400	1.0	125	53200	106400	140	0
Artix-7 AC70... Add Daught...		xilinx.com	1.4	xc7a200t...	676	1.1	400	134600	269200	365	0
Alveo U200		xilinx.com	1.3	Accelerat...	2104	1.0	676	1182240	2364480	2160	960

[? Back](#) **Next >** [Finish](#) [Cancel](#)

Project Summary should be look like this



Add DPU IP repository

[zcu102-dpu-trd-2019-1-timer Download link](#)

1. Click settings

2. Select IP/Repository

3. Click to select DPU IP path

4. Select *zcu102-dpu-trd-2019-1-timer\pl\srcs\dpu_ip*

5. Select

Name	Constraints	Status
> D:\synth_1	constrs_1	Not started
>> impl_1	constrs_1	Not started

Creat Block design

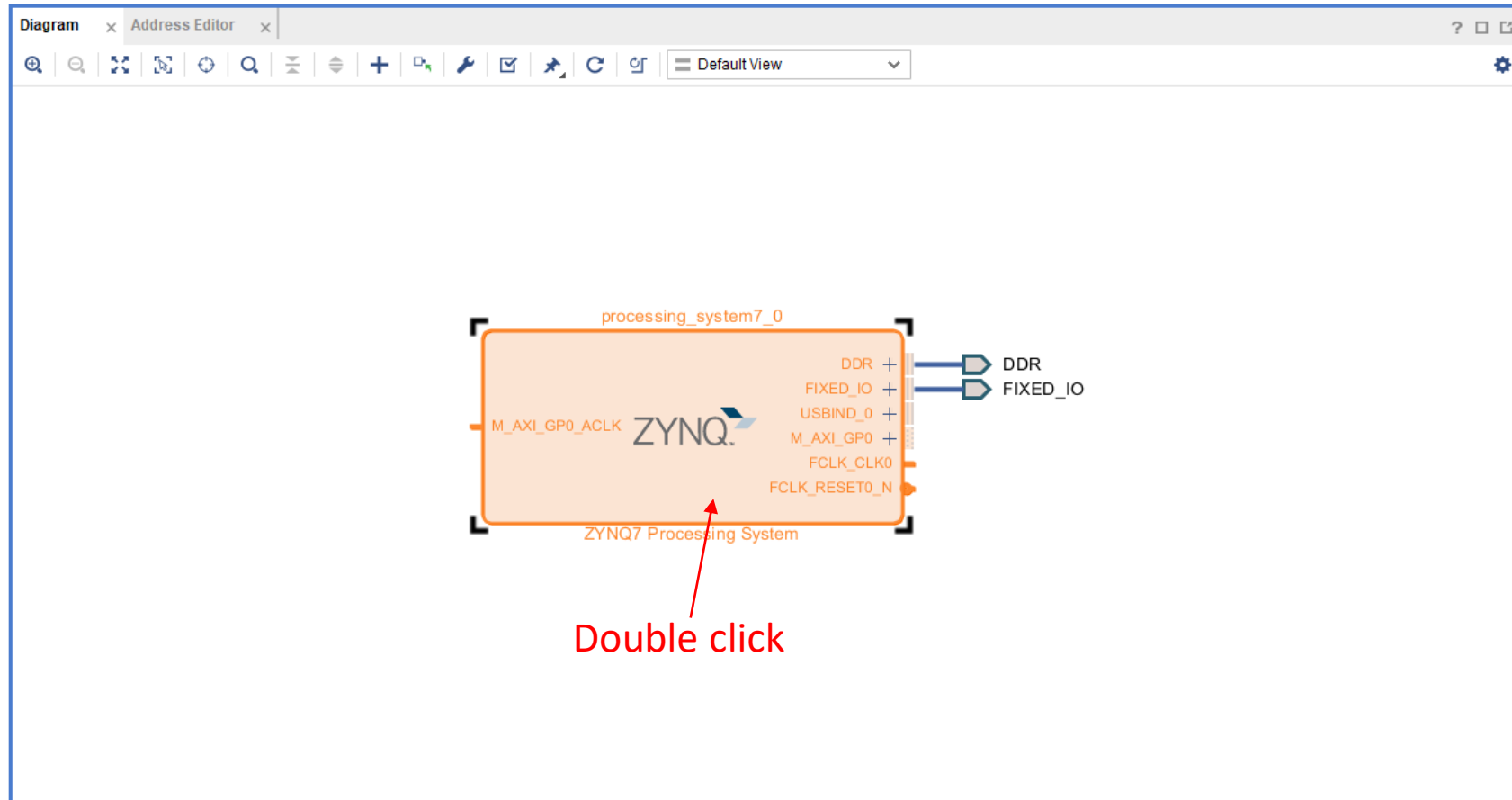
The image shows a software interface with a sidebar on the left and a main workspace on the right. The sidebar is titled 'Flow Navigator' and contains several sections: 'PROJECT MANAGER' (Settings, Add Sources, Language Templates, IP Catalog), 'IP INTEGRATOR' (Create Block Design, Open Block Design, Generate Block Design), 'SIMULATION' (Run Simulation), 'RTL ANALYSIS' (Open Elaborated Design), 'SYNTHESIS' (Run Synthesis, Open Synthesized Design), 'IMPLEMENTATION' (Run Implementation, Open Implemented Design), and 'PROGRAM AND DEBUG' (Generate Bitstream, Open Hardware Manager). The 'Create Block Design' option under 'IP INTEGRATOR' is highlighted with a red box. A red arrow points from this box to a dialog box titled 'Create Block Design'. The dialog box contains the following fields and buttons: 'Design name:' with the text 'pynqz2_dpu' in a text box; 'Directory:' with a dropdown menu showing '<Local to Project>'; 'Specify source set:' with a dropdown menu showing 'Design Sources'; a blue 'OK' button; and a 'Cancel' button. The 'OK' button is also highlighted with a red box. The main workspace shows a 'BLOCK DESIGN -' window with 'Sources' and 'Des' tabs, a search bar, and a list containing 'design_1'. Below the workspace are 'Properties' and 'Tcl Console' panels. The 'Tcl Console' shows a list of commands: set_property, set_property, update_ip_ca, INFO: [IP_F1, INFO: [IP_F1, create_bd_de, Wrote : <D:, create_bd_de, and update_compi.

Add Zynq processor and click Run Block Automation

The image shows a sequence of steps in the Xilinx Vivado IDE:

- Step 1:** The 'Add IP' menu is open, and the 'ZYNQ7 Processing System' is selected in the search results. A red box highlights the 'Add IP' button and the search result.
- Step 2:** The 'Run Block Automation' dialog box is displayed. A red box highlights the 'Run Block Automation' button in the top toolbar. Another red box highlights the 'OK' button at the bottom of the dialog.
- Step 3:** The final result is a block diagram showing the 'processing_system7_0' block. The block is labeled 'ZYNQ7 Processing System' and has several pins: 'M_AXI_GP0_ACLK', 'DDR', 'FIXED_IO', 'USBIND_0', 'M_AXI_GP0', 'FCLK_CLK0', and 'FCLK_RESET0_N'. Red arrows indicate the flow from the search results to the dialog box and then to the final block diagram.

Configure processing_system7_0



Configure processing_system7_0

Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation Presets IP Location Import XPS Settings

Page Navigator

Zynq Block Design

PS-PL Configuration

Search: Q-

Name	Select	Description
General		
AXI Non Secure Enablement	0	Enable AXI Non Secure Transaction
GP Slave AXI Interface		
HP Slave AXI Interface		
S AXI HP0 interface	<input checked="" type="checkbox"/>	Enables AXI high performance slave interface
S AXI HP1 interface	<input checked="" type="checkbox"/>	Enables AXI high performance slave interface
S AXI HP2 interface	<input checked="" type="checkbox"/>	Enables AXI high performance slave interface
S AXI HP2 DATA WIDTH	32	Allows HP2 to be used in 32/64 bit data width
S AXI HP3 interface	<input type="checkbox"/>	Enables AXI high performance slave interface
ACP Slave AXI Interface		
DMA Controller		
PS-PL Cross Trigger interface	<input type="checkbox"/>	Enables PL cross trigger signals to PS and

PS-PL configuration -> HP Slave AXI Interface
-> S AXI HP0~HP2
Change S AXI HP2 Data WIDTH 64 -> 32

Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation Presets IP Location Import XPS Settings

Page Navigator

Zynq Block Design

PS-PL Configuration

Peripheral I/O Pins

MIO Configuration

Clock Configuration

DDR Configuration

SMC Timing Calculation

Interrupts

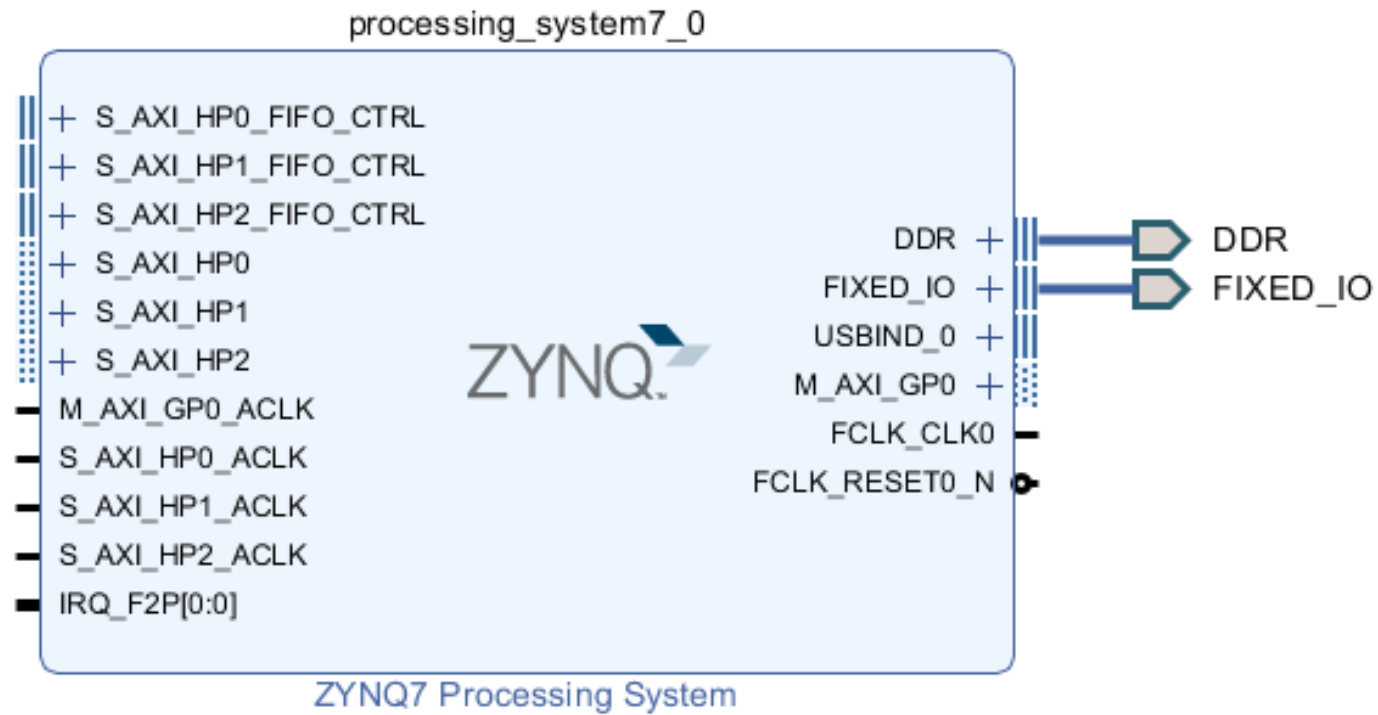
Search: Q-

Interrupt Port	ID	Description
<input checked="" type="checkbox"/> Fabric Interrupts		Enable PL Interrupts to PS and vice versa
PL-PS Interrupt Ports		
<input checked="" type="checkbox"/> IRQ_F2P[15:0]	[91:84], [68:6	Enables 16-bit shared interrupt port from the PL. MSB is assigned the hi
<input type="checkbox"/> Core0_nFIQ	28	Enables fast private interrupt signal for CPU0 from the PL
<input type="checkbox"/> Core0_nIRQ	31	Enables private interrupt signal for CPU0 from the PL
<input type="checkbox"/> Core1_nFIQ	28	Enables fast private interrupt signal for CPU1 from the PL
<input type="checkbox"/> Core1_nIRQ	31	Enables private interrupt signal for CPU1 from the PL
PS-PL Interrupt Ports		

OK Cancel

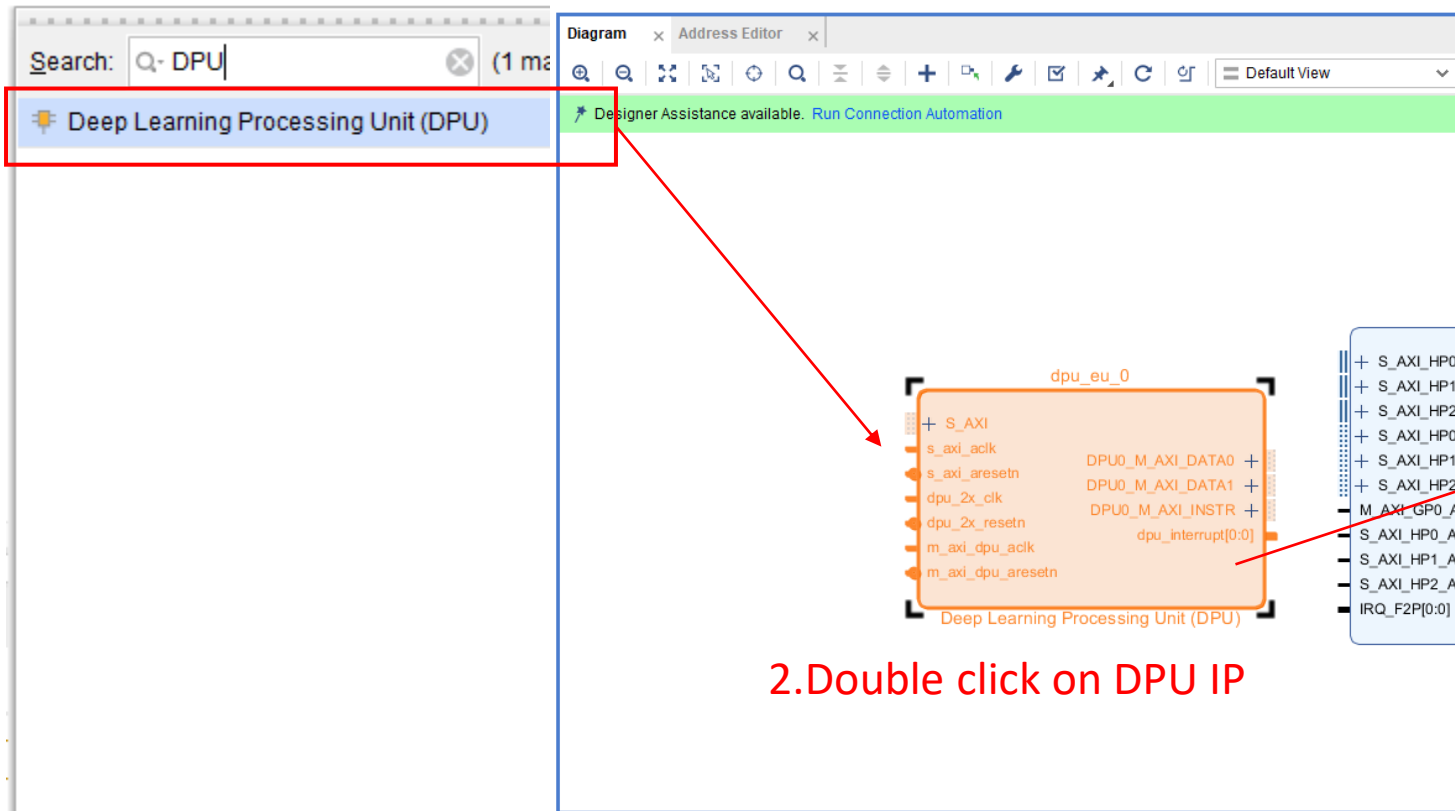
Interrupts-> Fabric Interrupts -> IRQ_F2P[15:0]
Click Ok and ignore warnings.

It should look like this

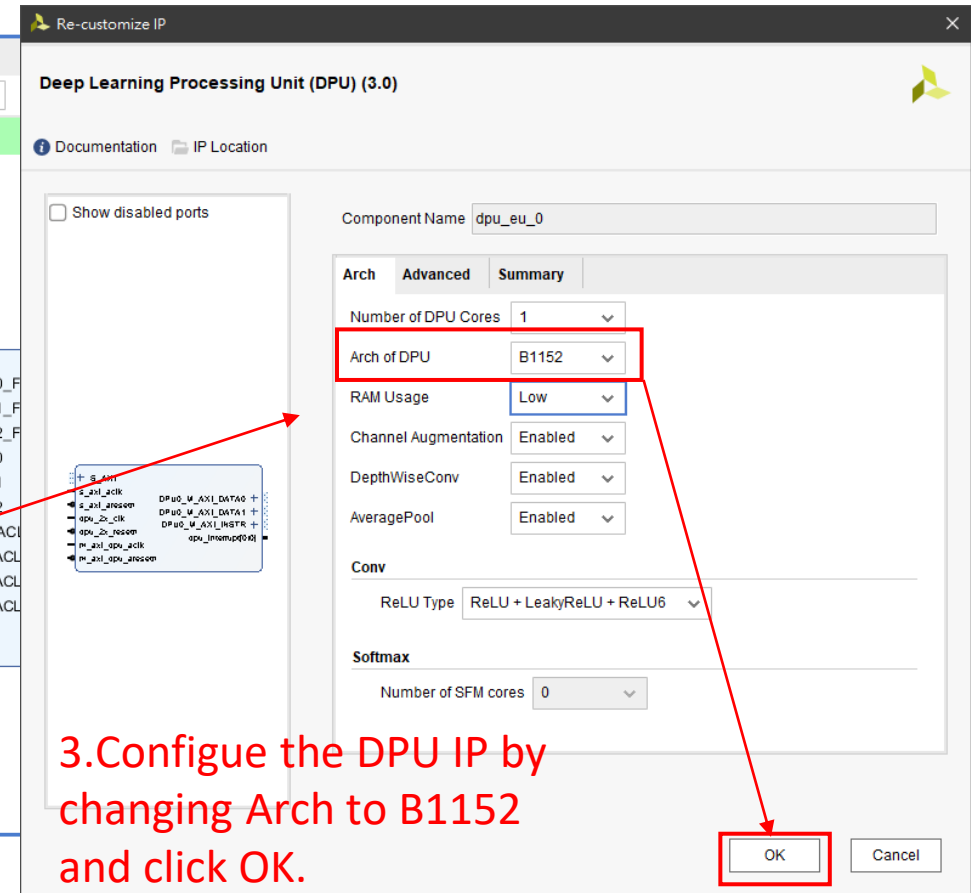


Add DPU IP

1. Search for DPU IP and add it into design



2. Double click on DPU IP



3. Configure the DPU IP by changing Arch to B1152 and click OK.

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

Connect DPU with PS

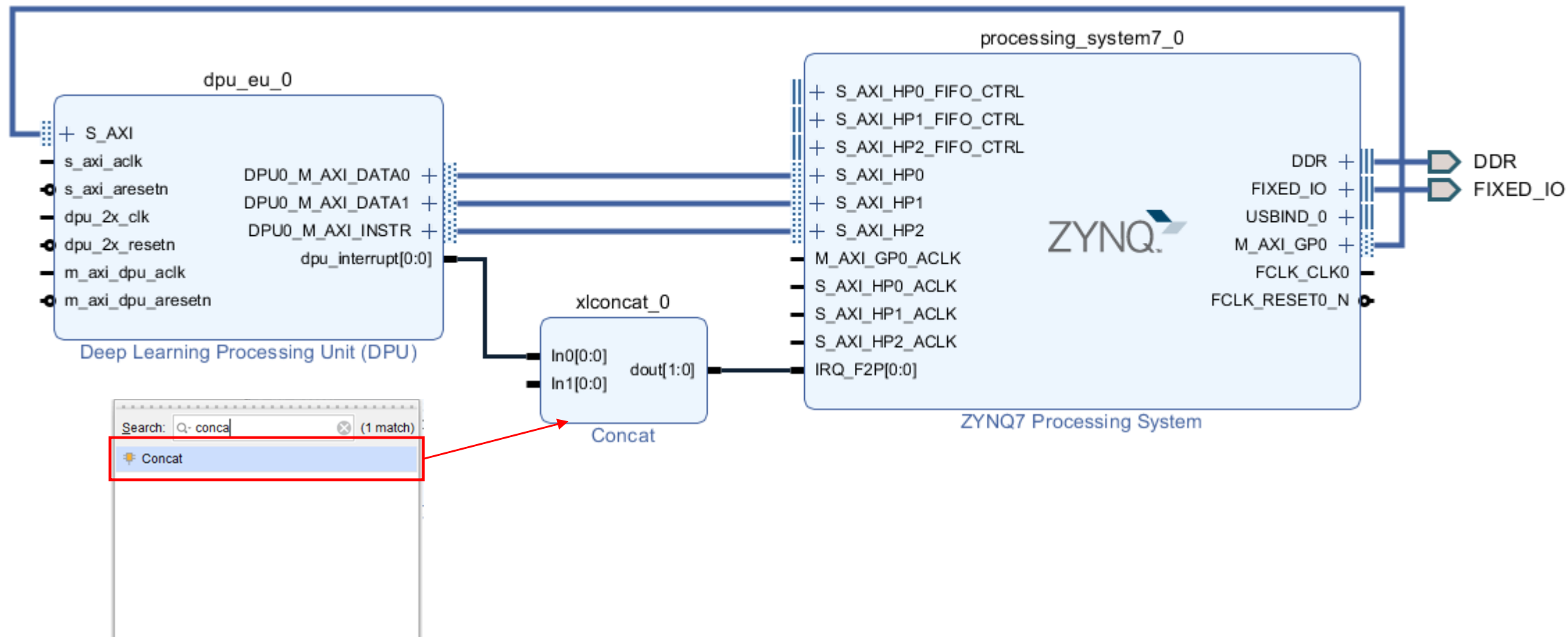
DPU0_M_AXI_DATA0 -> S_AXI_HP0

DPU0_M_AXI_DATA1 -> S_AXI_HP1

DPU0_M_AXI_INSTR -> S_AXI_HP2

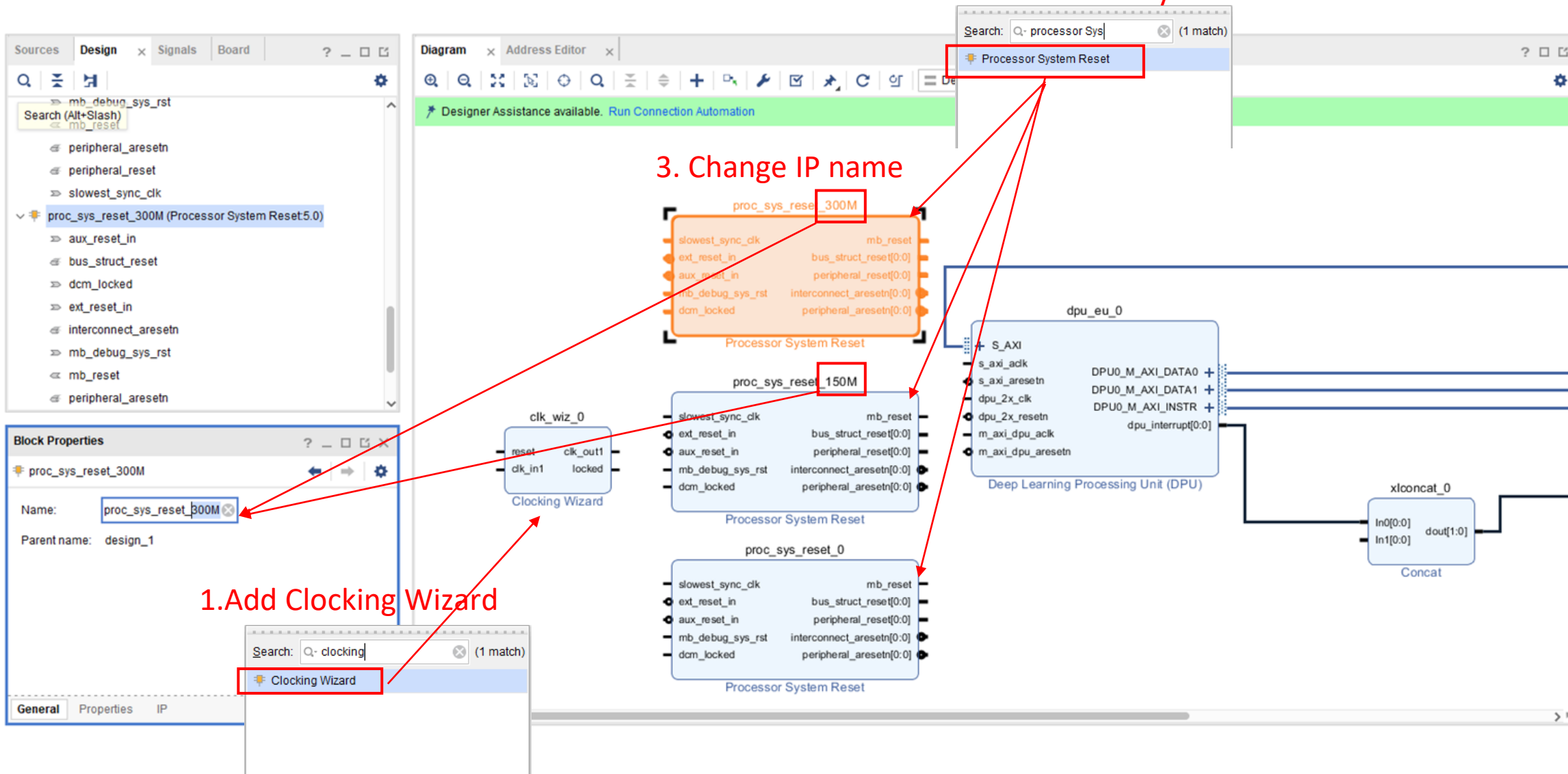
(DPU) S_AXI -> M_AXI_GP0

dpu_interrupt -> (concat) in0, (concat)dout -> (PS) IRQ_F2P



Add other require IP

2. Add Processor System Reset



3. Change IP name

1. Add Cloning Wizard

Configure clocking wizard



1. Double Click

2. Select Output Clocks

3. Configure clk_out

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)
		Requested	Actual	Requested	Actual	
<input checked="" type="checkbox"/> clk_out1	clk_out_150M	150.000	150.00000	0.000	0.000	50.00
<input checked="" type="checkbox"/> clk_out2	clk_out_300M	300.000	300.00000	0.000	0.000	50.00
<input type="checkbox"/> clk_out3	clk_out3	100.000	N/A	0.000	N/A	50.00
<input type="checkbox"/> clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.00
<input type="checkbox"/> clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.00
<input type="checkbox"/> clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.00
<input type="checkbox"/> clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.00

4. Scroll Down and Change Reset type to active Low

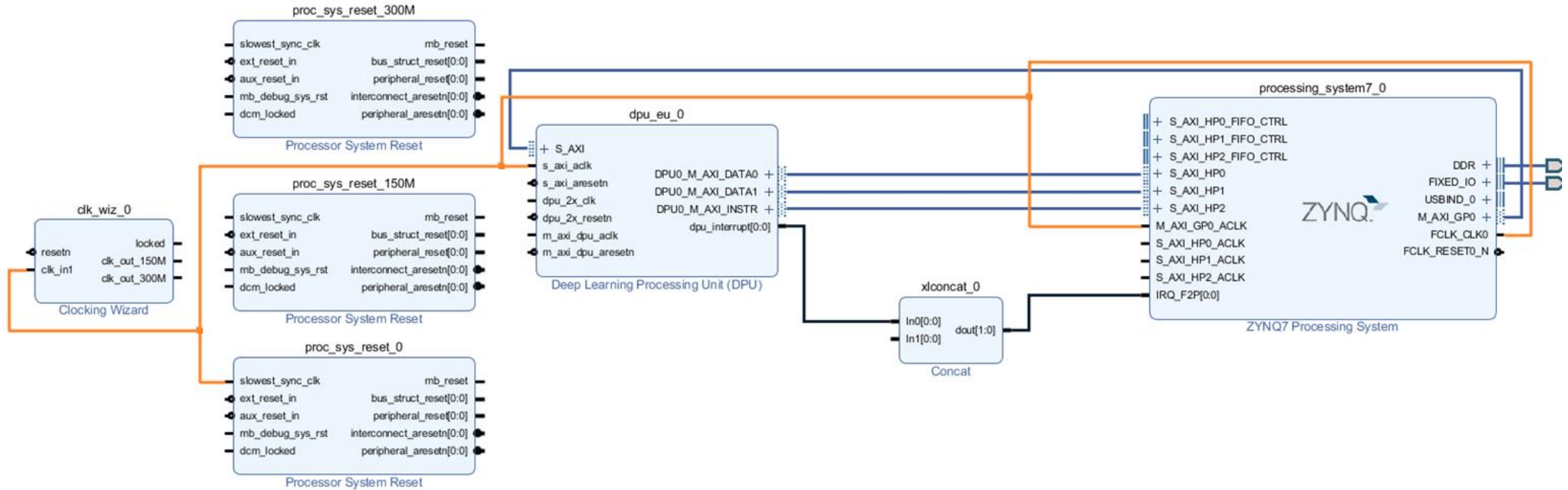
5. Click OK

Output Clock	Sequence Number	Source	Signaling
clk_out1	1	Automatic Control On-Chip	Single-ended
clk_out2	1	Automatic Control Off-Chip	Differential
clk_out3	1	User-Controlled On-Chip	
clk_out4	1	User-Controlled Off-Chip	
clk_out5	1		
clk_out6	1		
clk_out7	1		

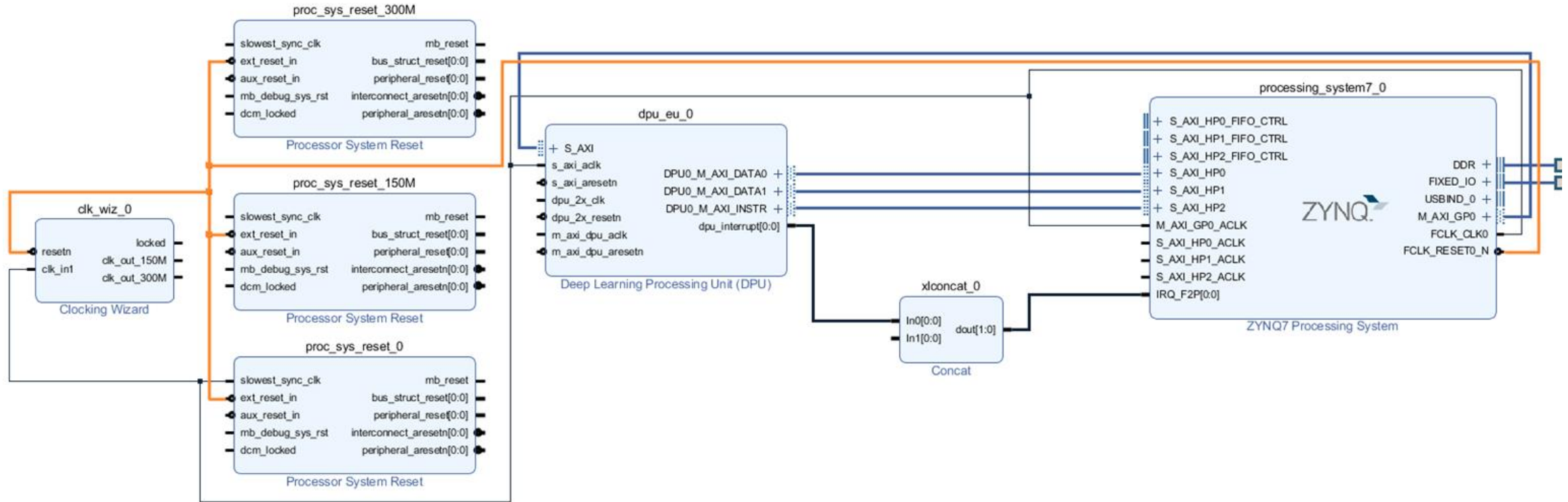
Reset Type: Active High Active Low

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)
		Requested	Actual	Requested	Actual	
<input checked="" type="checkbox"/> clk_out1	clk_out_150M	150.000	150.00000	0.000	0.000	50.00
<input checked="" type="checkbox"/> clk_out2	clk_out_300M	300.000	300.00000	0.000	0.000	50.00

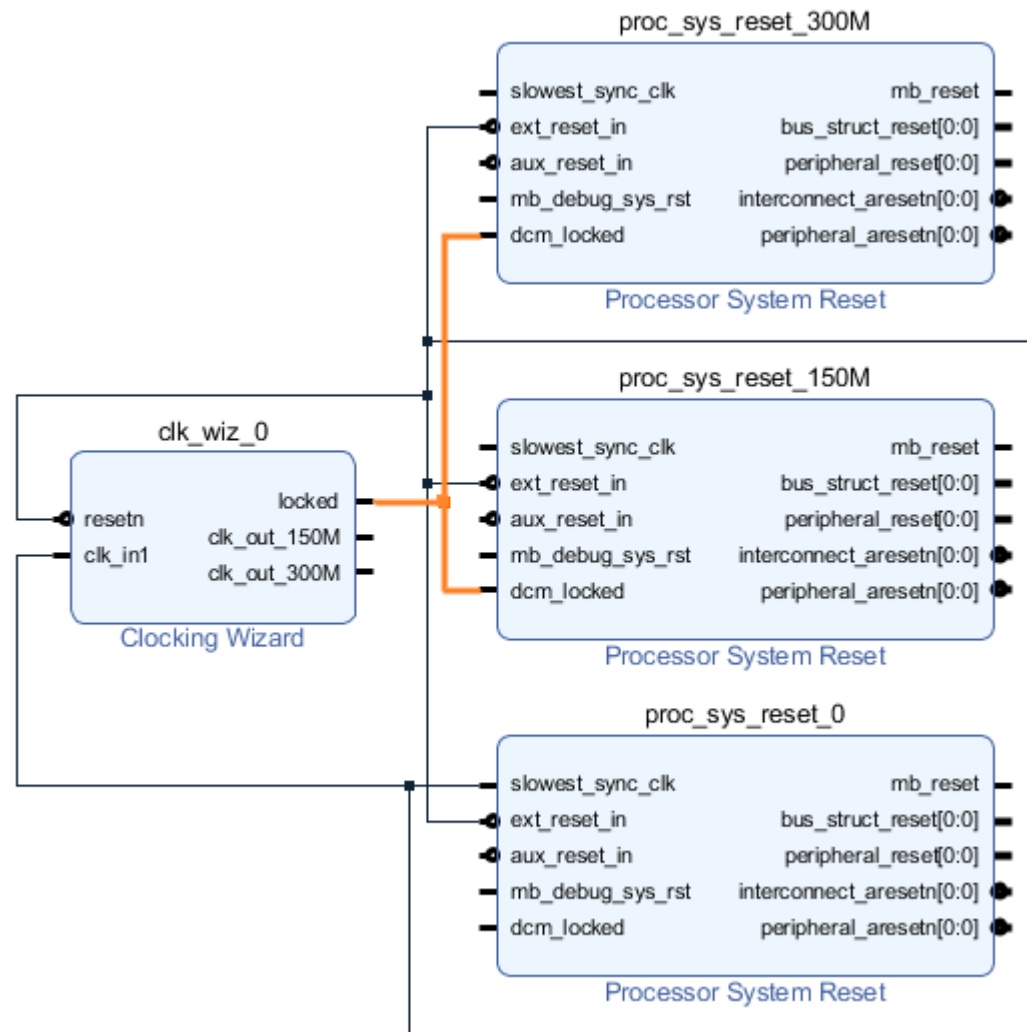
Connect clock



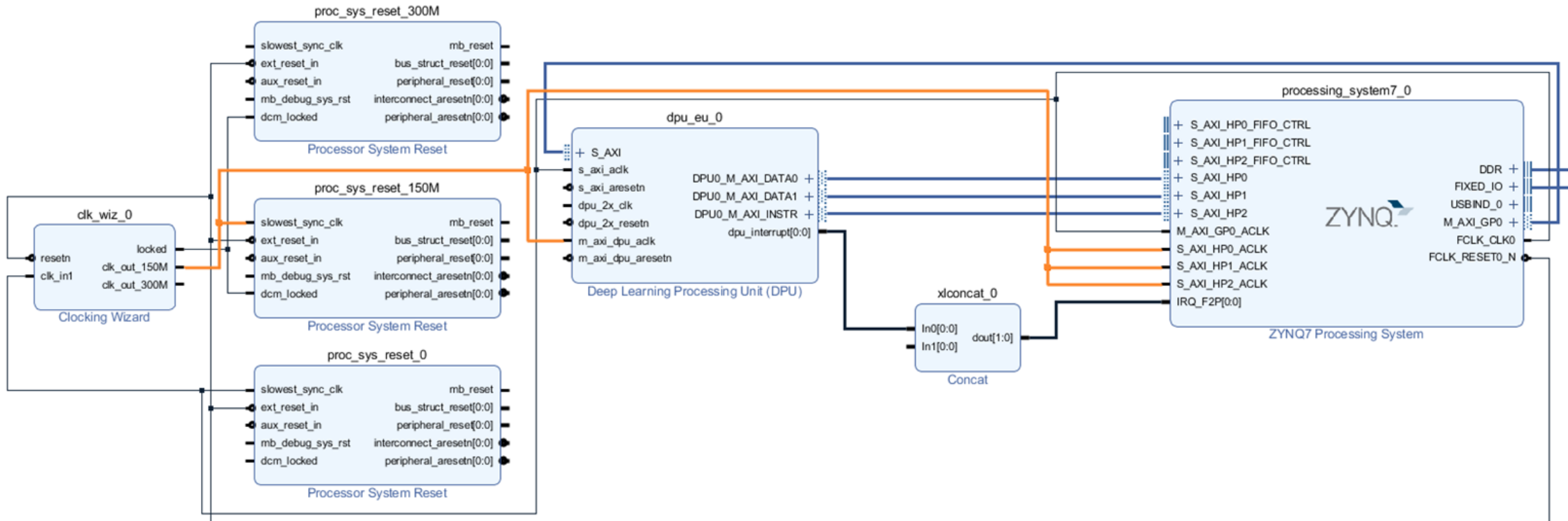
Connect reset



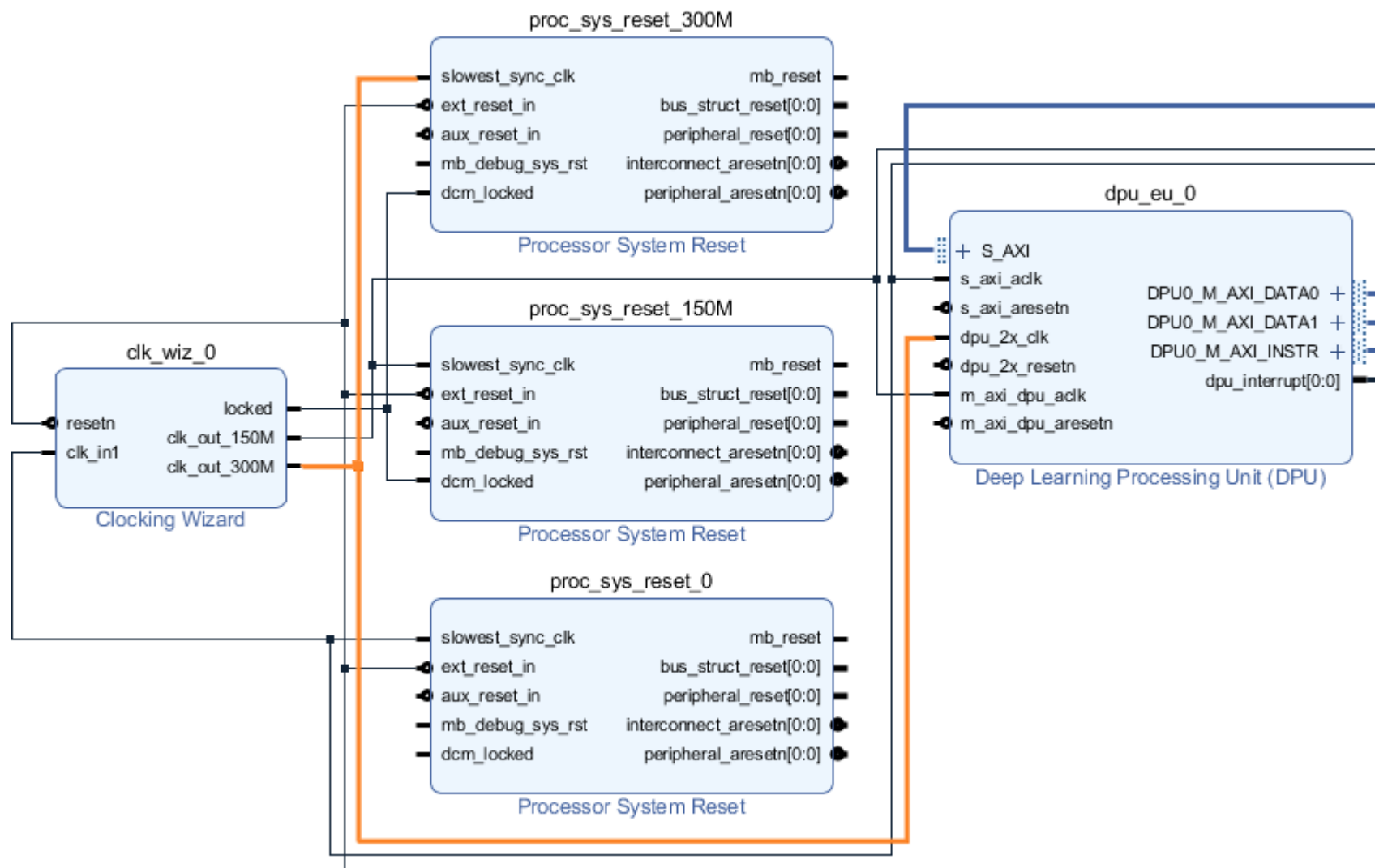
Connect lock



Connect Clock_150Mhz

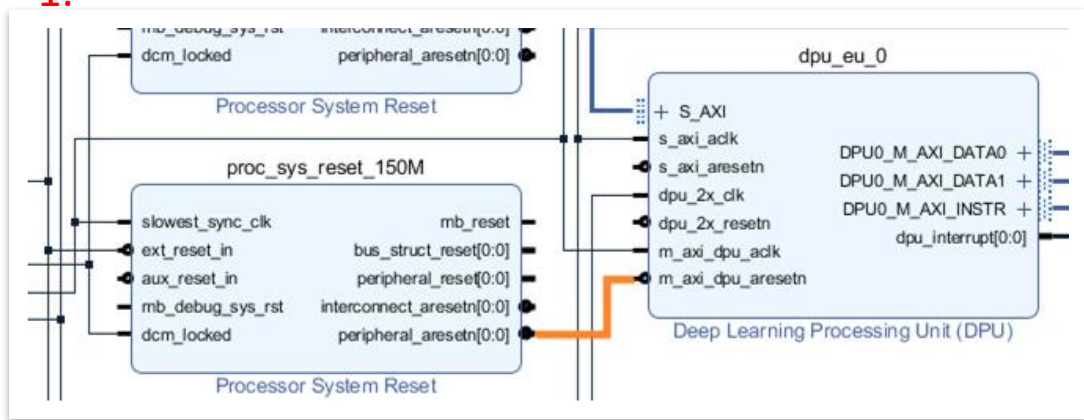


Connect Clock_300Mhz

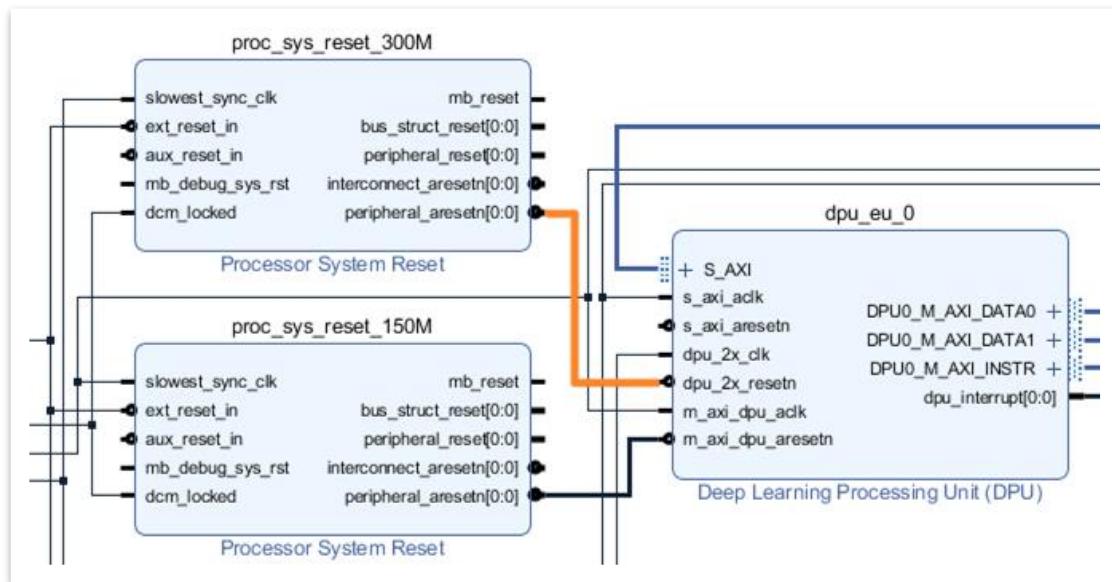


Connect dpu aresetn (Active-low reset)

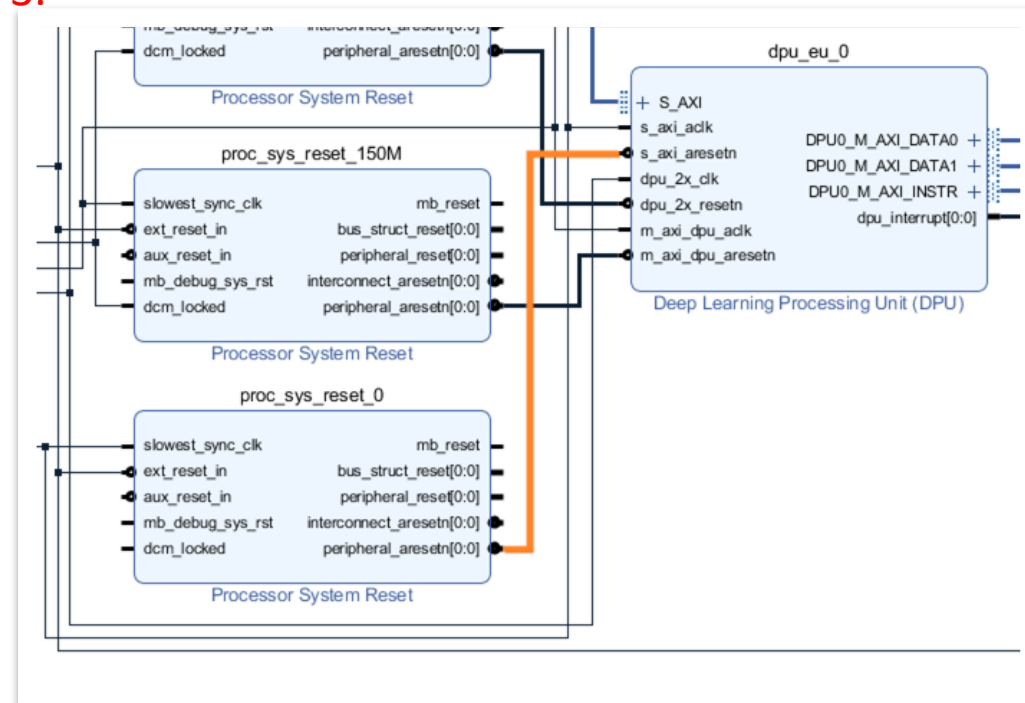
1.



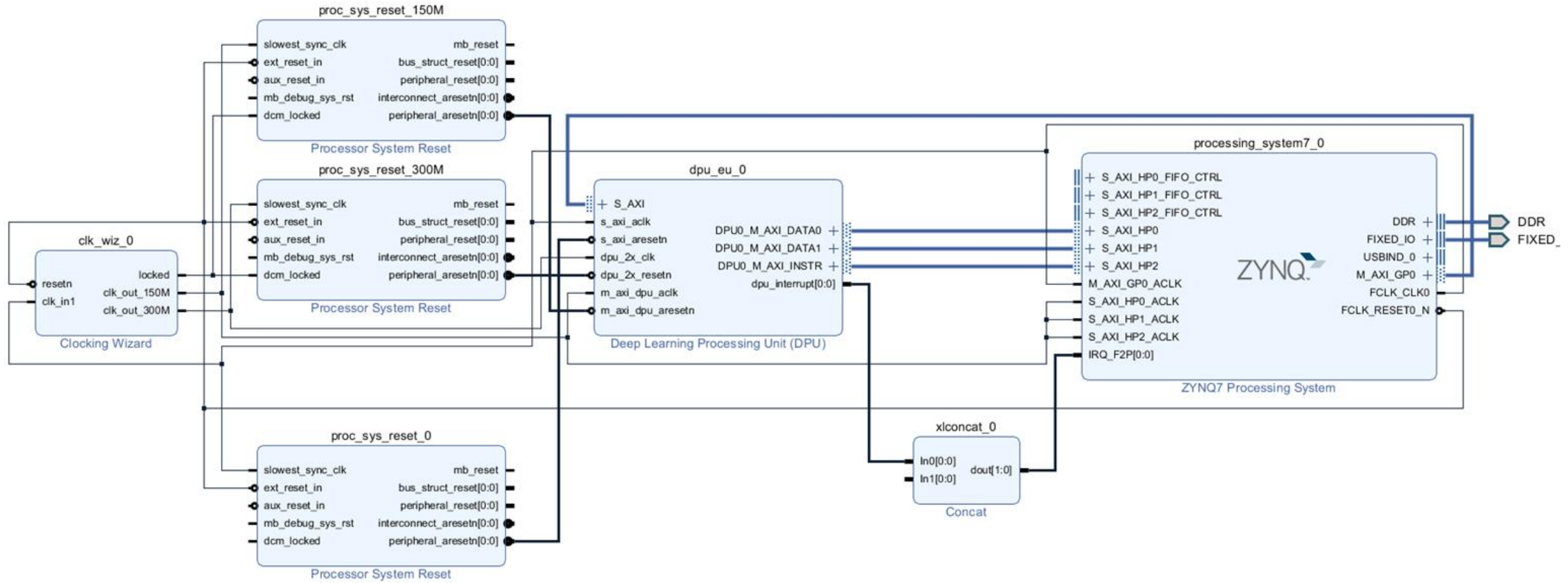
2.



3.



Final Layout



Assign address

1. Switch to Address editor

2. Click Assign all

3. Change dpu_eu_0 Address to 0x4F00_0000
And make sure address is 0x4F00_0000 ~ 0x4FFF_FFFF

Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
/dpu_eu_0	S_AXI	reg0	0x4F00_0000	16M	0x4FFF_FFFF

Address Space Properties

Data

Use name: Usta

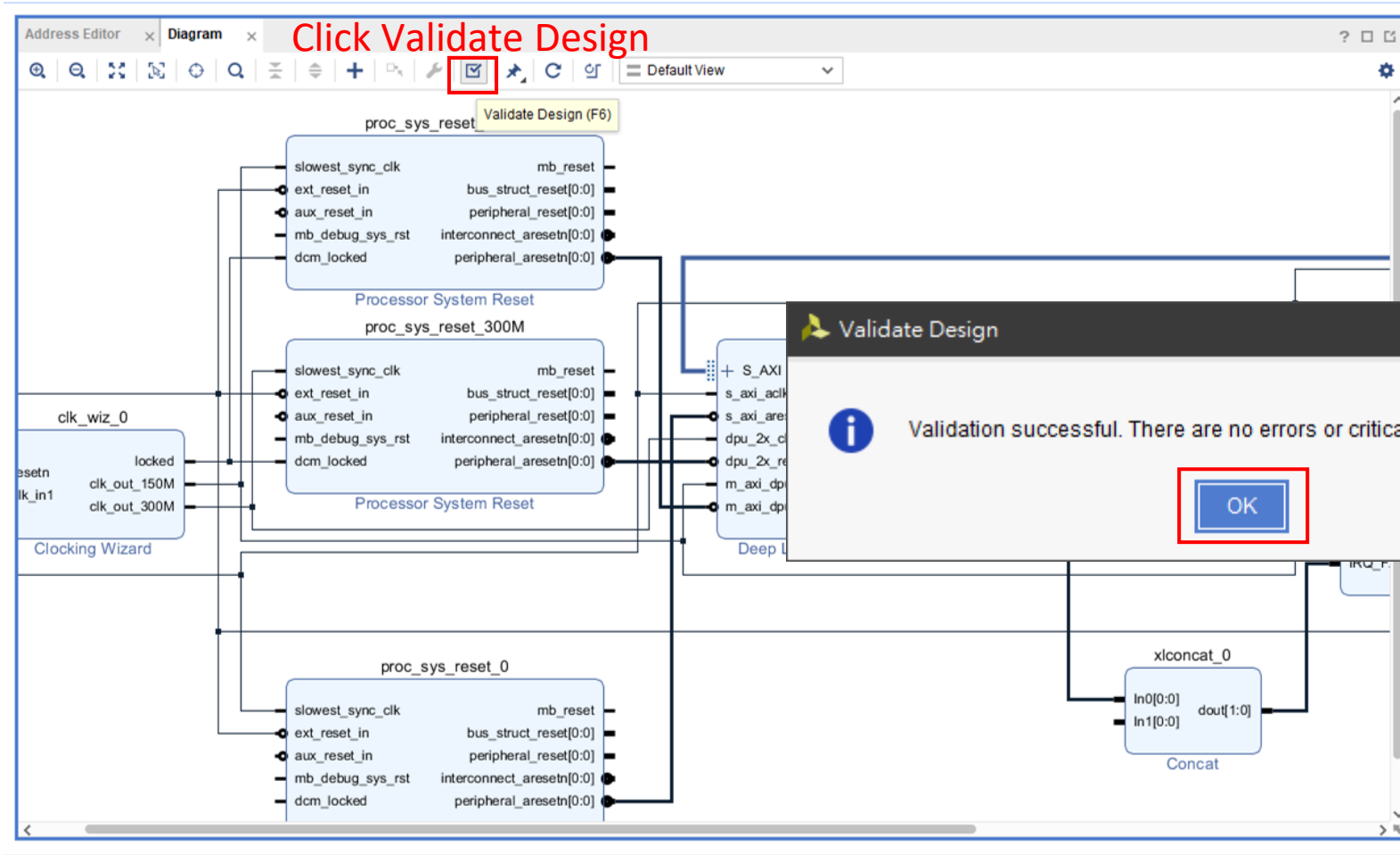
Full name: processing_system7_0/Data

Range: 0x000000000000

Width: 32

Master block: processing_system7_0

Validate Design



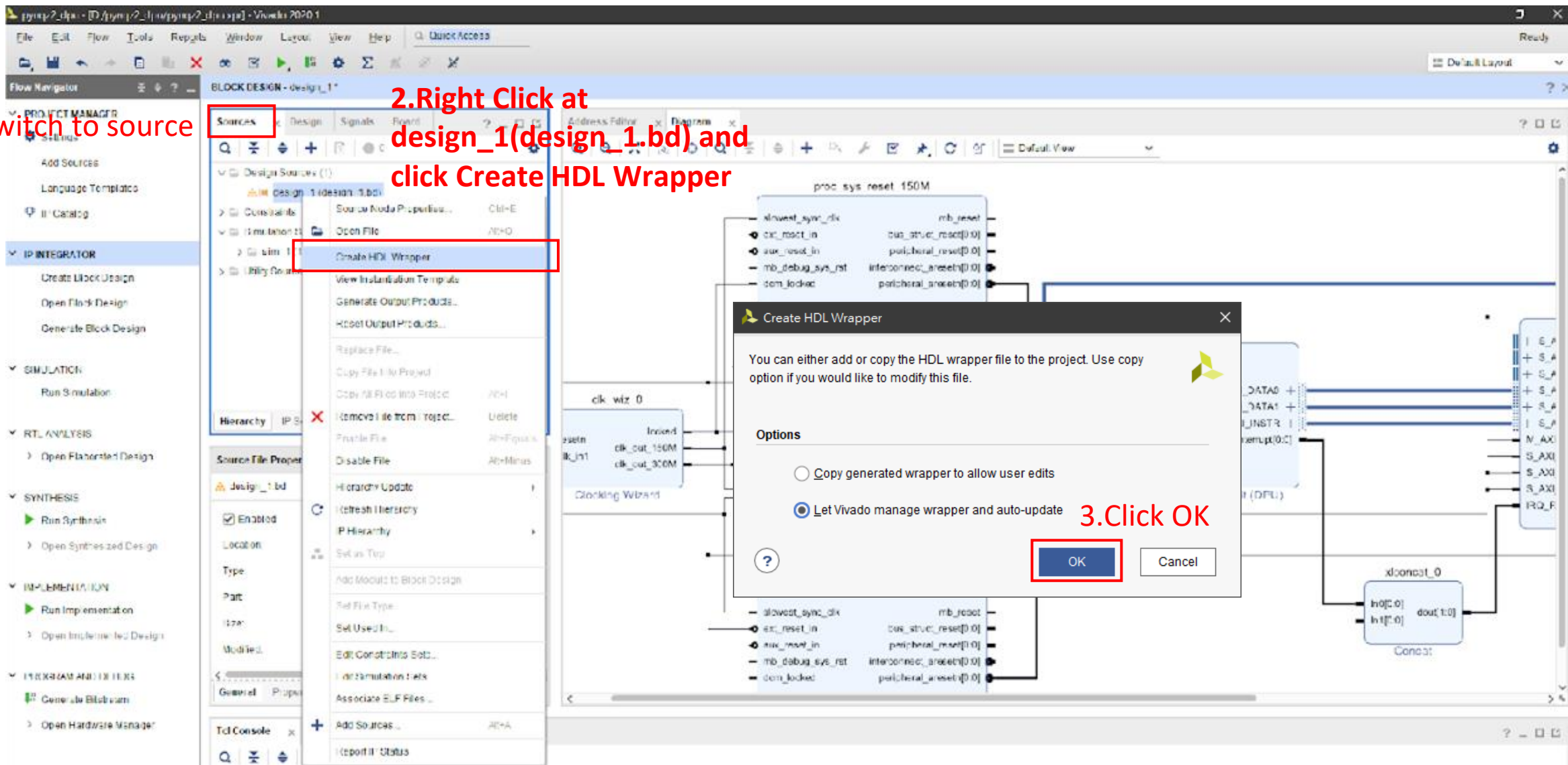
Should Be successful

Create HDL Wrapper

1. Switch to source

2. Right Click at design_1(design_1.bd) and click Create HDL Wrapper

3. Click OK



Generate Bitstream

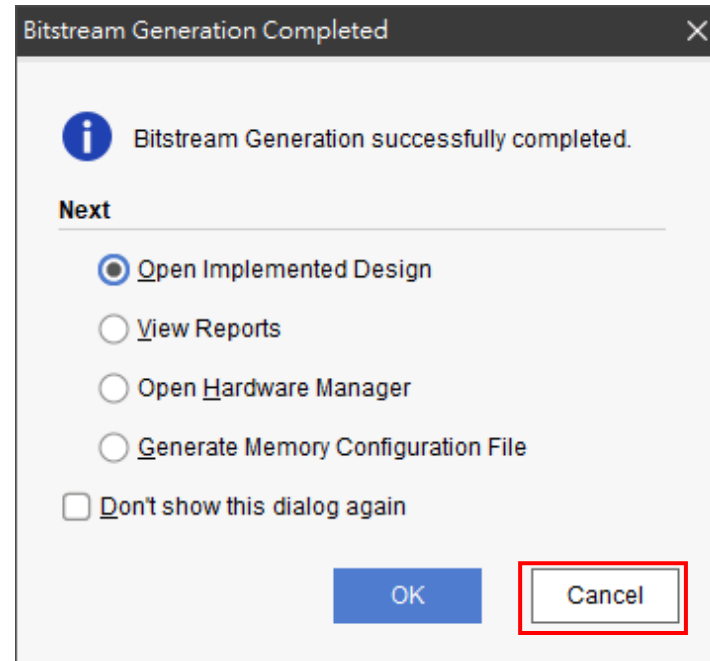
The screenshot shows the Vivado 2020.1 interface with the following elements:

- Left Sidebar:** The 'PROGRAM AND DEVICES' section is expanded, and the 'Generate Bitstream' button is highlighted with a red box. A red arrow points from this button to the 'Launch Runs' dialog box.
- Center Dialogs:**
 - A 'No Implementation Results Available' dialog box is open, with a red box around the 'Yes' button and a red arrow pointing to it. The text '2. Click Yes' is written in red above the button.
 - A 'Launch Runs' dialog box is open in front of it, with a red box around the 'OK' button and a red arrow pointing to it. The text '3. Click OK' is written in red above the button.
- Background:** The main workspace shows a block design diagram with various components like 'proc_sys_reset_150M', 'alovest_syn_rdx', 'mb_reset', 'ext_reset_in', 'bus_struct_reset[0:0]', 'aux_res', 'mb_deb', and 'dem_joc'. The 'Sources' panel on the left shows 'Design Sources (1)' containing 'design_1_wrapper (design_1_wrapper.v) (1)'. The 'Tcl Console' at the bottom is empty.

Wait for it

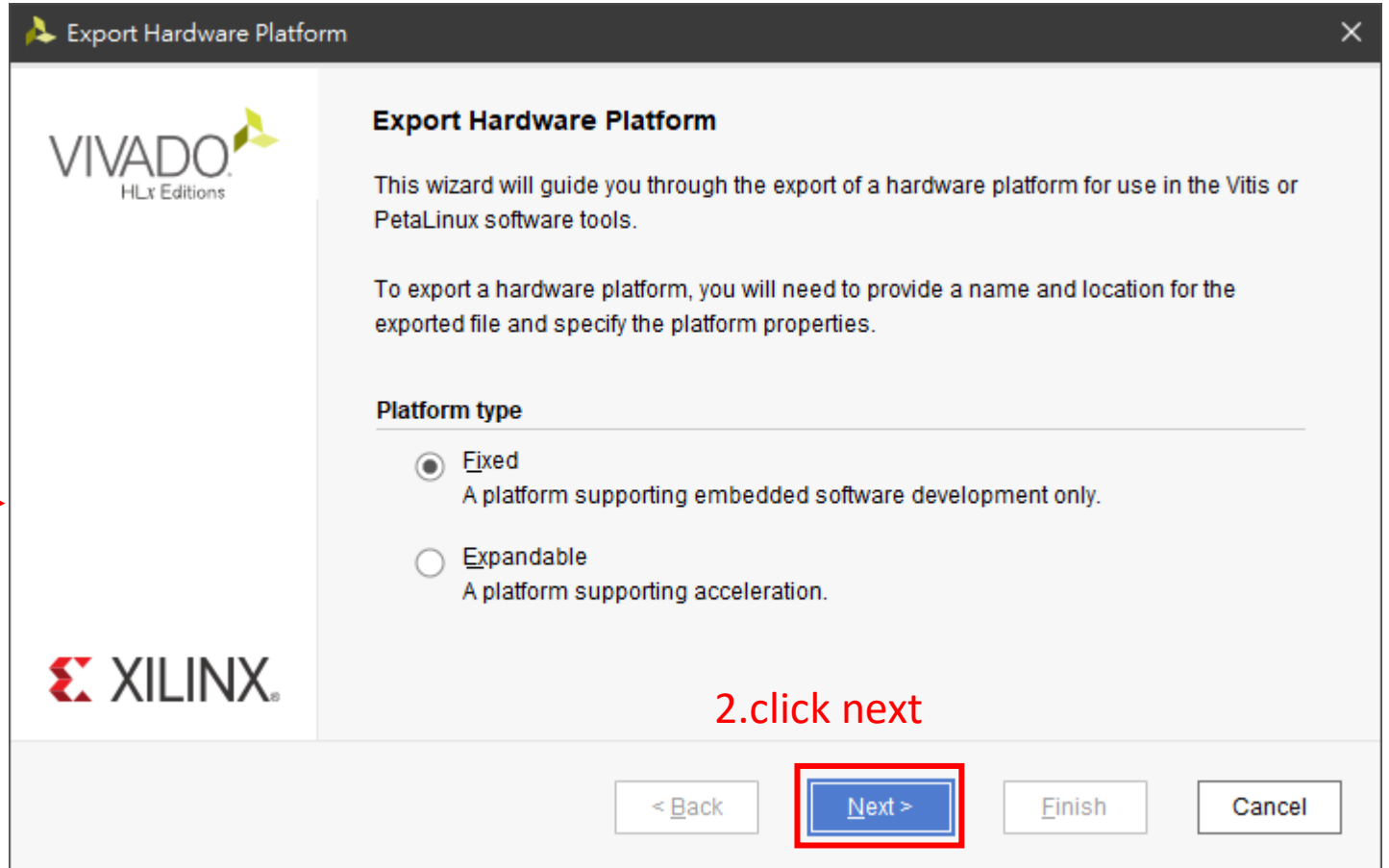
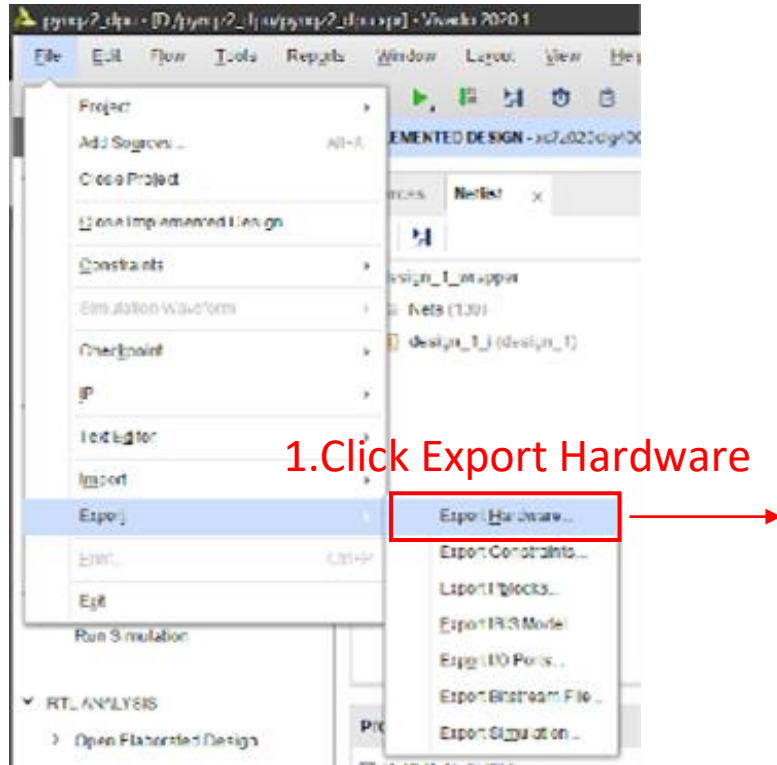
The image shows the Vivado 2020.1 interface during a bitstream generation process. A modal dialog box titled "Generate Bitstream" is open, displaying the message "Generate IP 'processing_system7_0'..." and a progress bar. A red box highlights the "Background" button in the dialog. In the background, a block design diagram is visible, featuring components such as "proc_sys_reset_150M", "proc_sys_reset_300M", "proc_sys_reset_0", "dpu_eu_0", and "xiconcat_0". A red arrow points from the text "Wait for it" to the "Background" button. The top right corner of the Vivado window shows a status bar with the text "Running multiple block runs" and a "Cancel" button, also highlighted with a red box.

Finish

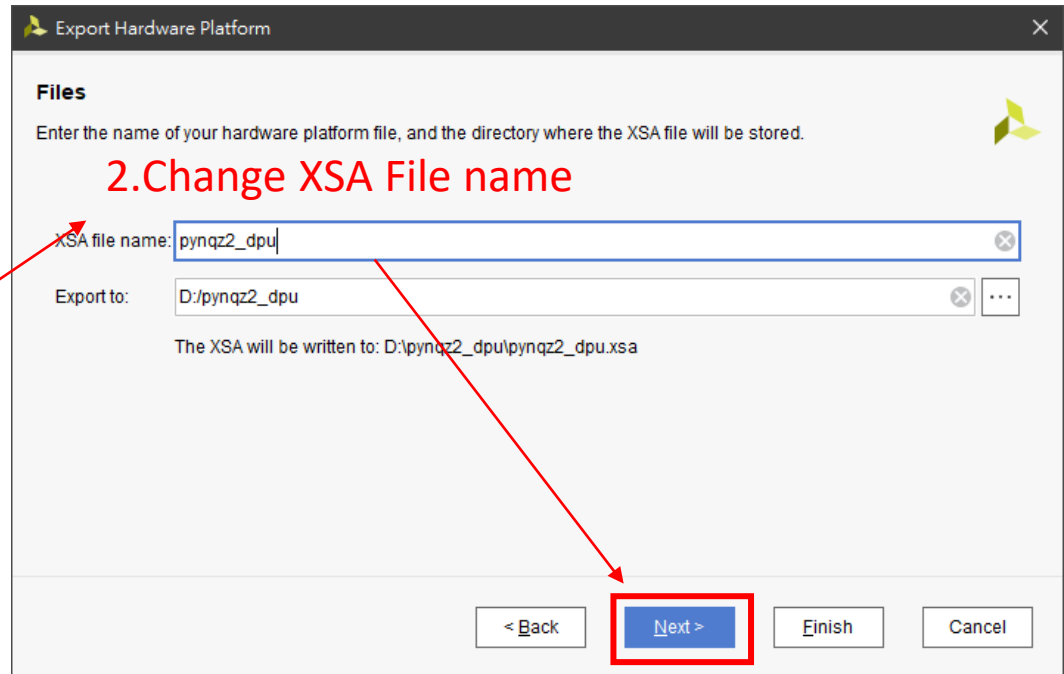
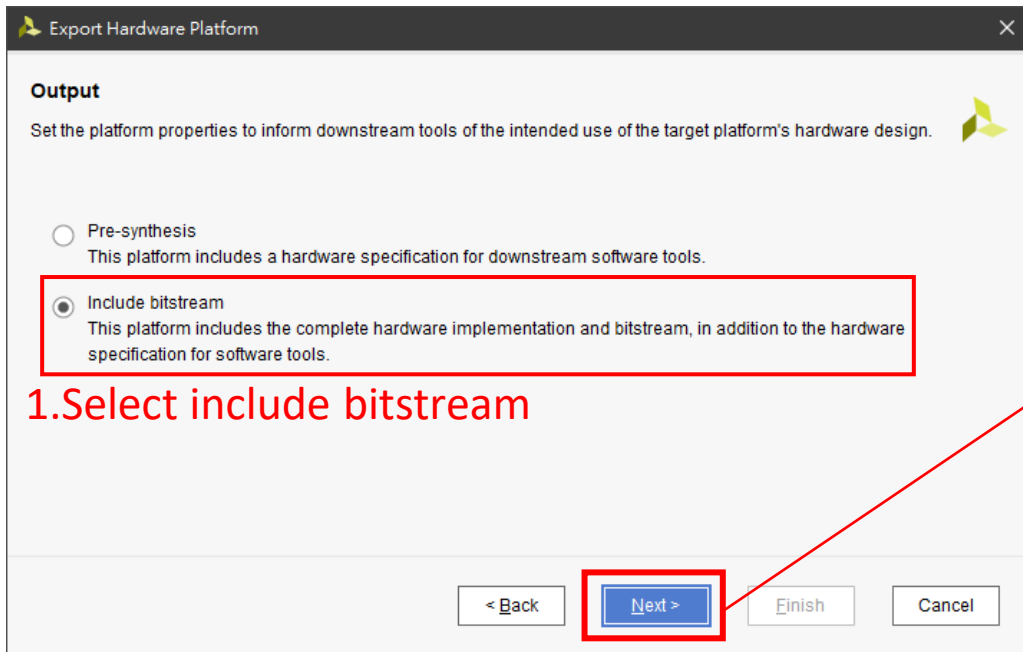


Click cancel when finished
generate Bitstream

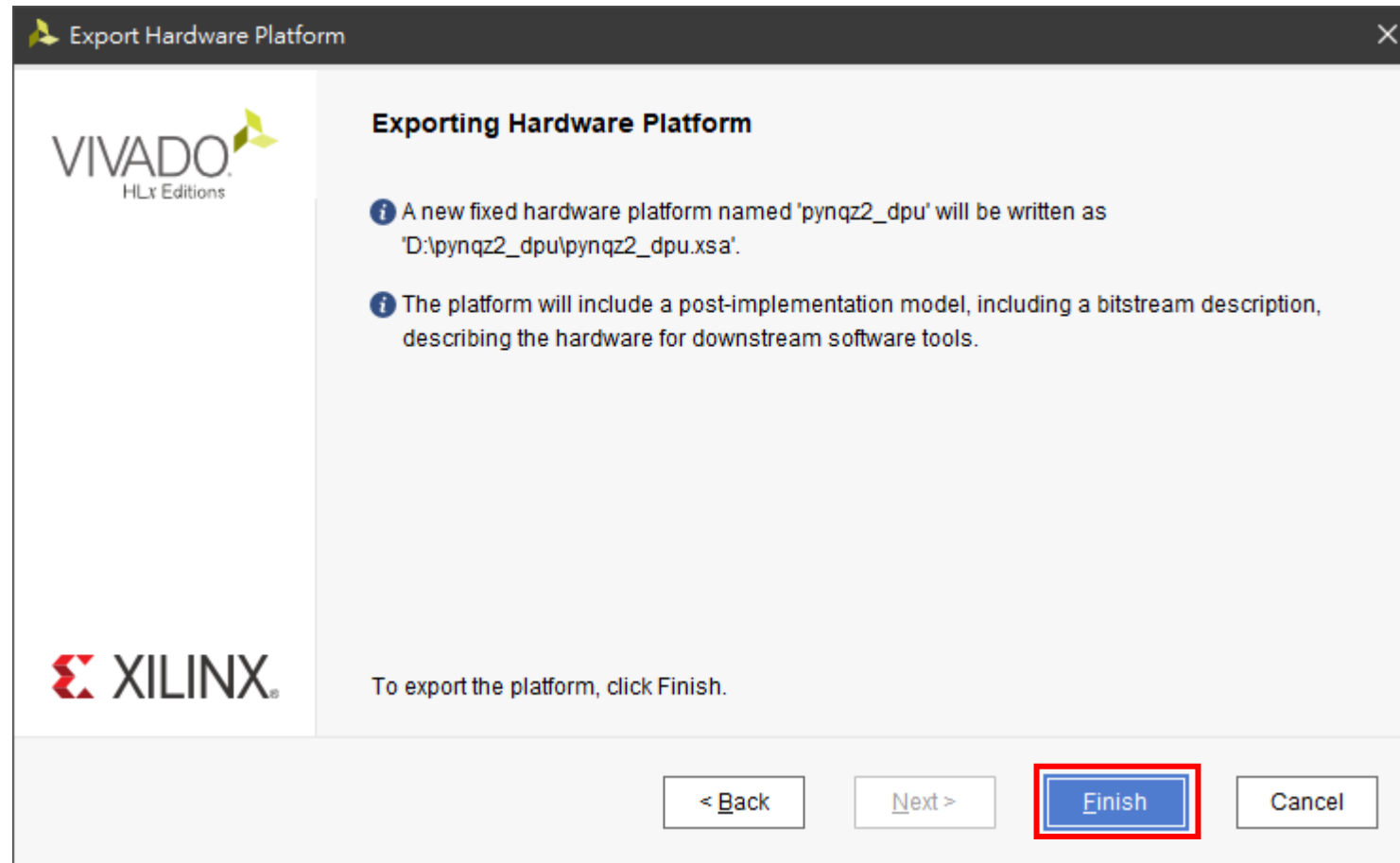
Export Hardware



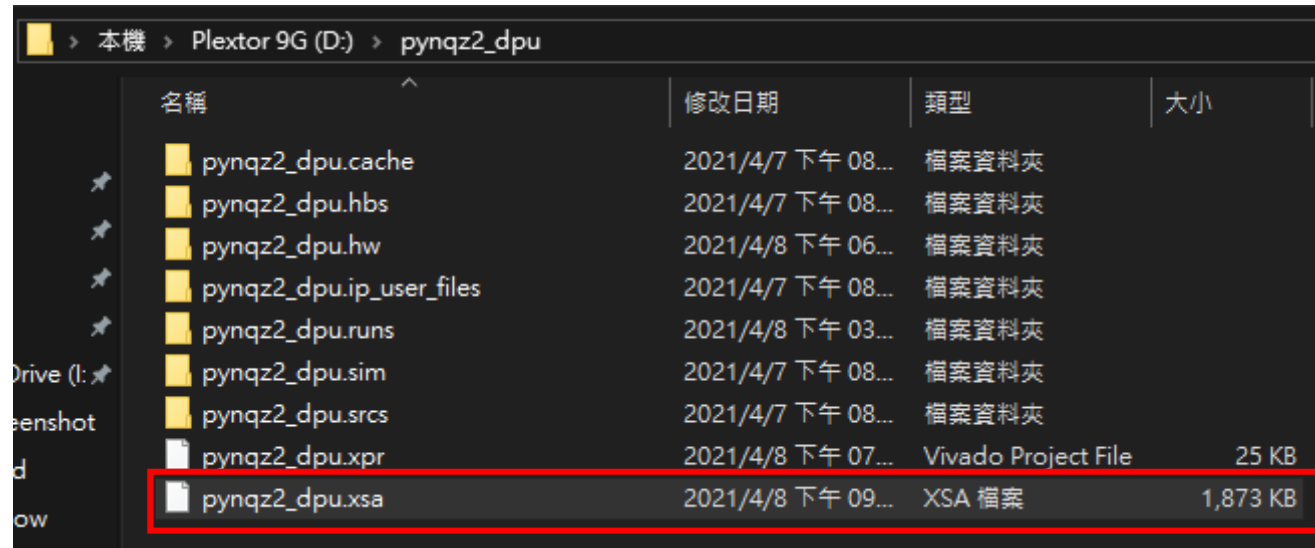
Export Hardware



Click Finish



Done!



Save pynqz2_dpu.xsa