DPUCZDX8G for Zynq UltraScale+ MPSoCs

Product Guide

PG338 (v3.4) January 20, 2022

Xilinx is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing non-inclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this link for more information.

Table of Contents

Chapter 1: Introduction **4**
- Features 4
- IP Facts 5

Chapter 2: Overview **6**
- Core Overview 6
- Hardware Architecture 7
- Development Tools 9
- Example System with the DPUCZDX8G 10
- Vitis AI Development Kit 11

Chapter 3: Product Specification **13**
- Port Descriptions 13
- Register Space 14
- Interrupts 19

Chapter 4: DPU Configuration **21**
- Introduction 21
- Configuration Options 24
- Advanced Tab 31
- Summary Tab 33
- DPUCZDX8G Performance on Different Devices 34
- Performance of Different Models 34
- I/O Bandwidth Requirements 36
- Clocking 37
- Reset 42

Chapter 5: Development Flow **44**
- Customizing and Generating the Core in the Zynq UltraScale+ MPSoC 44
- Customizing and Generating the Core in the Vitis IDE 51

Chapter 6: Example Design **56**

Introduction..... 56
Vivado DPU TRD Flow..... 56
Vitis DPU TRD Flow..... 57

Appendix A: Additional Resources and Legal Notices..... 58
Xilinx Resources..... 58
Documentation Navigator and Design Hubs..... 58
References..... 58
Revision History..... 59
Please Read: Important Legal Notices..... 61

Introduction

The DPUCZDX8G is the deep learning processing unit (DPU) designed for the Zynq UltraScale+ MPSoC. It is a configurable computation engine optimized for convolutional neural networks. The degree of parallelism utilized in the engine is a design parameter and can be selected according to the target device and application. 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.

Features

The DPUCZDX8G has the following features:

- One AXI slave interface for accessing configuration and status registers.
- One AXI master interface for accessing instructions.
- Supports individual configuration of each channel.

Some highlights of DPUCZDX8G functionality include:

- Configurable hardware architecture core includes: B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096
- Maximum of four homogeneous cores
- Convolution and Transposed Convolution
- Depthwise convolution and Depthwise transposed convolution
- Max pooling
- Average pooling
- ReLU, ReLU6, and Leaky ReLU
- Elementwise-Sum and Elementwise-Multiply
- Dilation
- Reorg
- Fully connected layer
- Softmax

- Concat, Batch Normalization : supported by tool-chain

IP Facts

DPUCZDX8G IP Facts Table	
Core Specifics	
Supported Device Family	Zynq® UltraScale+™ MPSoC Family
Supported User Interfaces	Memory-mapped AXI interfaces
Resources	See Chapter 4: DPU Configuration.
Provided with Core	
Design Files	Encrypted RTL
Example Design	Verilog
Constraints File	Xilinx Design Constraints (XDC)
Supported S/W Driver	Included in PetaLinux
Tested Design Flows	
Design Entry	Vivado® Design Suite and Vitis™ unified software platform
Simulation	N/A
Synthesis	Vivado® Synthesis
Xilinx Support web page	

Notes:

1. Linux OS and driver support information are available from DPUCZDX8G TRD or Vitis™ AI development kit.
2. For the supported tool versions, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973).
3. The DPUCZDX8G is driven by instructions generated by the Vitis AI compiler. When the target neural network (NN), DPUCZDX8G hardware architecture, or AXI data width is changed, the related .xmodel file which contains DPUCZDX8G instructions must be regenerated.

Overview

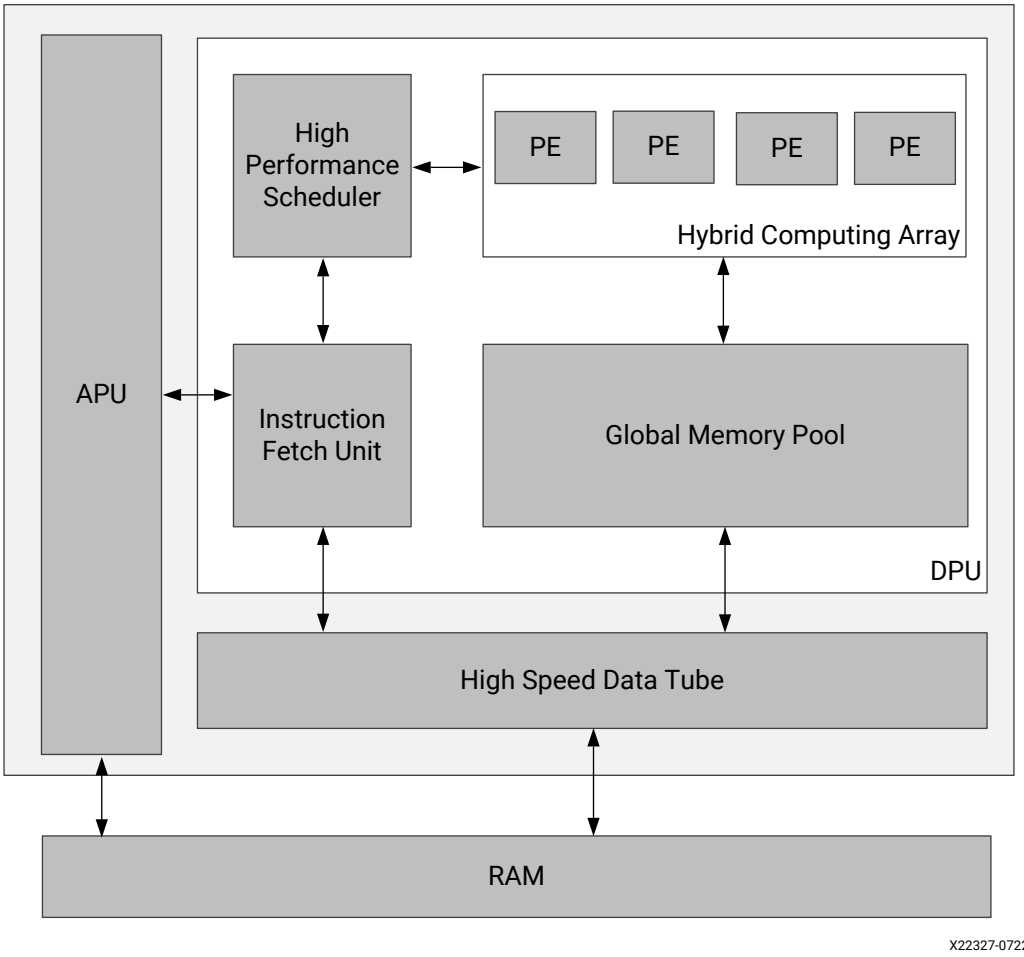
Core Overview

The Xilinx® DPUCZDX8G is a programmable engine optimized for convolutional neural networks. It is composed of a high performance scheduler module, a hybrid computing array module, an instruction fetch unit module, and a global memory pool module. The DPUCZDX8G uses a specialized instruction set, which allows for the efficient implementation of many convolutional neural networks. Some examples of convolutional neural networks which have been deployed include VGG, ResNet, GoogLeNet, YOLO, SSD, MobileNet, and FPN among others.

The DPUCZDX8G IP can be implemented in the programmable logic (PL) of the selected Zynq® UltraScale+™ MPSoC device with direct connections to the processing system (PS). The DPUCZDX8G 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.

The top-level block diagram of the DPUCZDX8G is shown in the following figure.

Figure 1: **DPUCZDX8G Top-Level Block Diagram**



where,

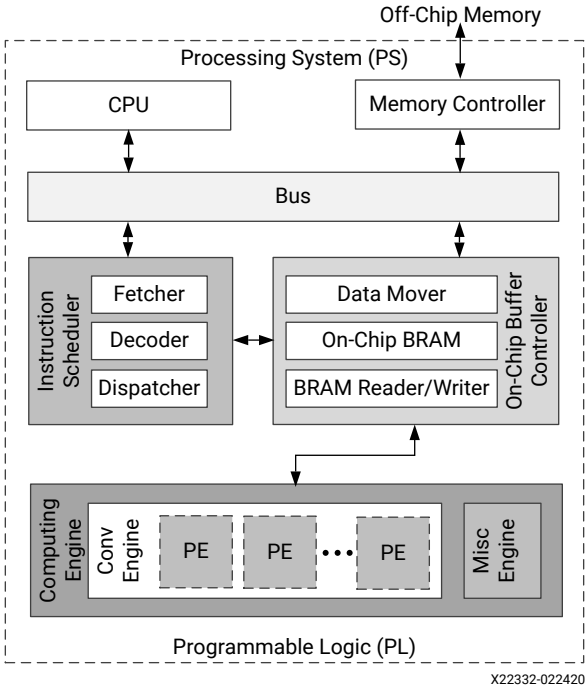
- APU - Application Processing Unit
- PE - Processing Engine
- DPU - Deep Learning Processing Unit
- RAM - Random Access Memory

Hardware Architecture

The detailed hardware architecture of the DPUCZDX8G is shown in the following figure. After start-up, the DPUCZDX8G fetches instructions from the off-chip memory to control the operation of the computing engine. The instructions are generated by the Vitis™ AI compiler, where substantial optimizations are performed.

On-chip memory is used to buffer input, intermediate, and output data to achieve high throughput and efficiency. The data is reused as much as possible to reduce the external memory bandwidth. A deep pipelined design is used for the computing engine. The processing elements (PE) take full advantage of the fine-grained building blocks such as multipliers, adders, and accumulators in Xilinx devices.

Figure 2: **DPUCZDX8G Hardware Architecture**

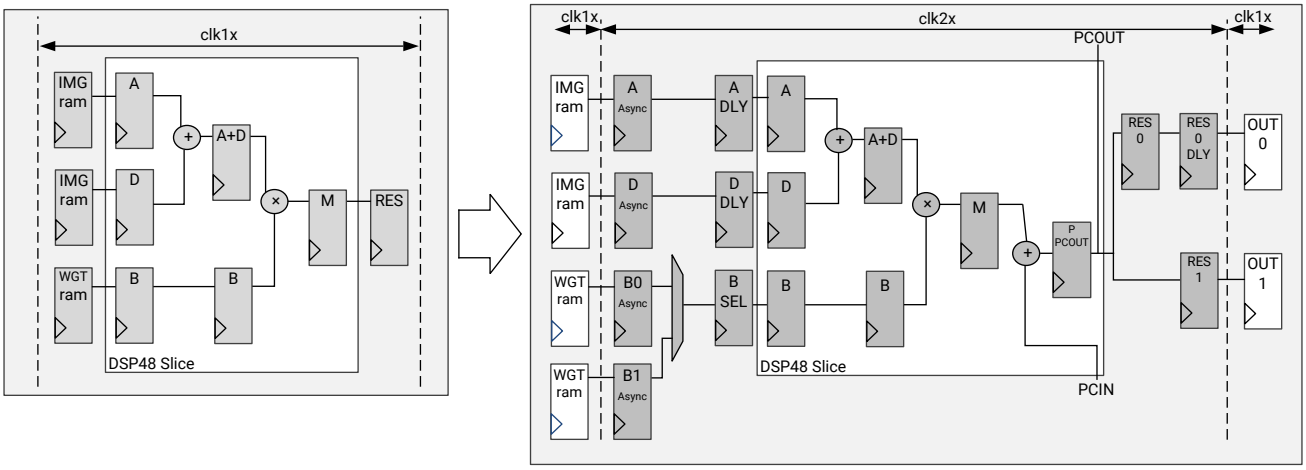


DPUCZDX8G with Enhanced Usage of DSP

A DSP Double Data Rate (DDR) technique is used to improve the performance achieved with the device. Therefore, two input clocks for the DPUCZDX8G are needed: One for general logic and another at twice the frequency for DSP slices. The difference between a DPUCZDX8G not using the DSP DDR technique and a DPUCZDX8G enhanced usage architecture is shown here.

Note: All DPUCZDX8G architectures referred to in this document refer to DPUCZDX8G enhanced usage, unless otherwise specified.

Figure 3: **Difference between DPUCZDX8G without DSP DDR and DPUCZDX8G Enhanced Usage**



Development Tools

Two flows are supported for integrating the DPUCZDX8G into your project: the Vivado flow and the Vitis™ flow.

The Xilinx Vivado® Design Suite is required to integrate the DPUCZDX8G into your projects for the Vivado flow. Vivado Design Suite 2021.2 or later version is recommended. Contact your local sales representative if the project requires an older version of Vivado.

The Vitis unified software platform 2021.2 or later is required to integrate the DPUCZDX8G for the Vitis flow.

Device Resources

The DPUCZDX8G logic resource usage is scalable across UltraScale+™ MPSoC devices. For more information on resource utilization, see the Chapter 4: DPU Configuration.

Related Information

DPU Configuration

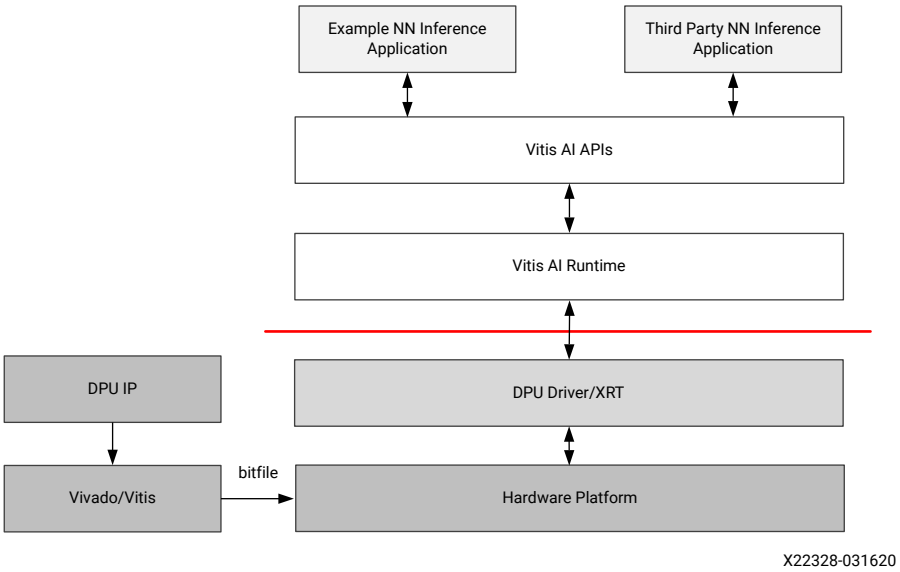
DPUCZDX8G Development Flow

The DPUCZDX8G requires a device driver which is included in the Xilinx Vitis™ AI development kit.

Free developer resources can be obtained from the Xilinx website: https://github.com/Xilinx/Vitis-AI.

The *Vitis AI User Guide* (UG1414) describes how to use the DPUCZDX8G with the Vitis AI tools. The basic development flow is shown in the following figure. First, use Vivado or Vitis to generate the bitstream. Then, download the bitstream to the target board and install the related driver. For instructions on installing the related driver and dependent libraries, see the *Vitis AI User Guide* (UG1414).

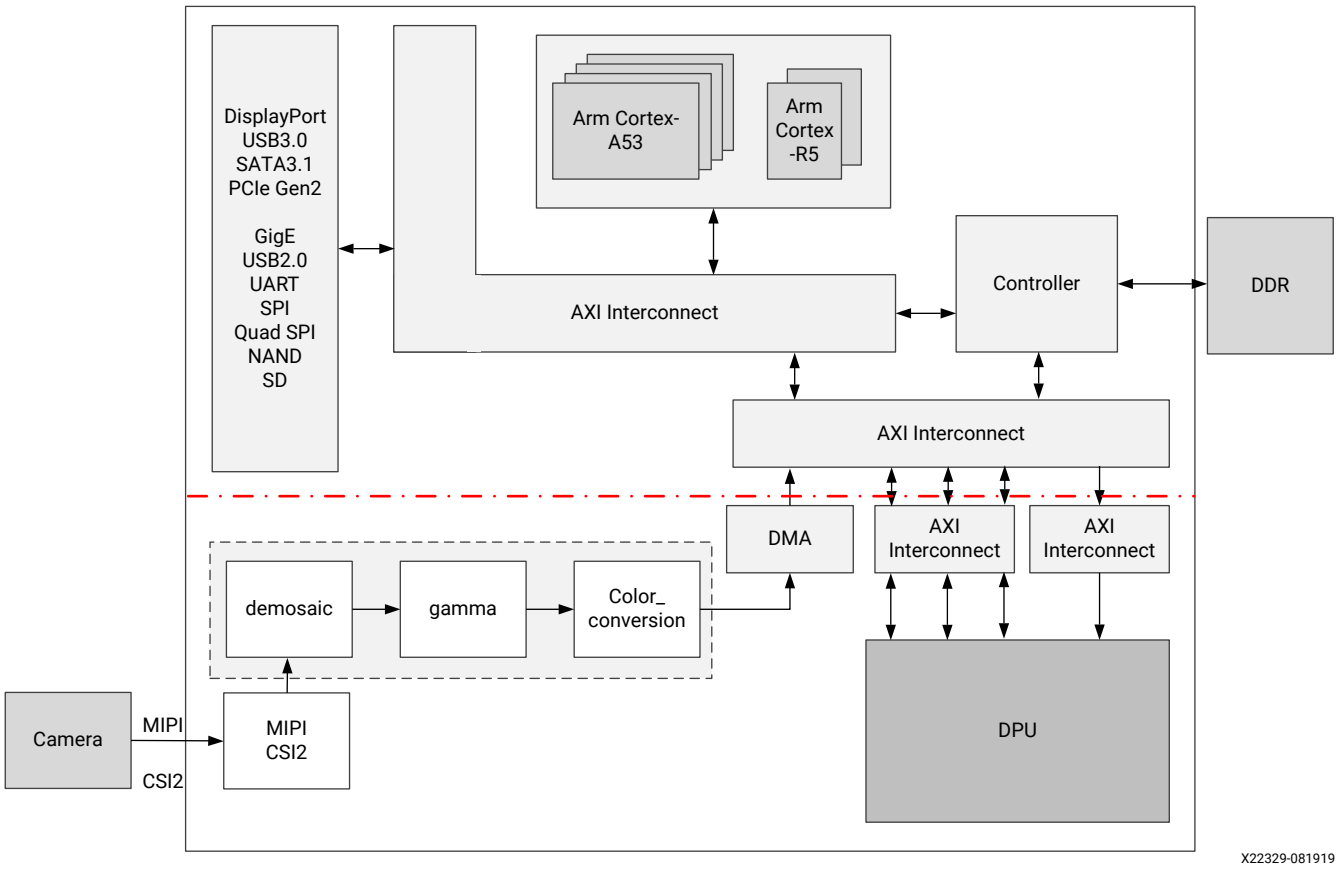
Figure 4: **HW/SW Stack**



Example System with the DPUCZDX8G

The figure below shows an example system block diagram with the Xilinx® UltraScale+™ MPSoC using a camera input. The DPUCZDX8G is integrated into the system through an AXI interconnect to perform deep learning inference tasks such as image classification, object detection, and semantic segmentation.

Figure 5: **Example System with Integrated DPUCZDX8G**

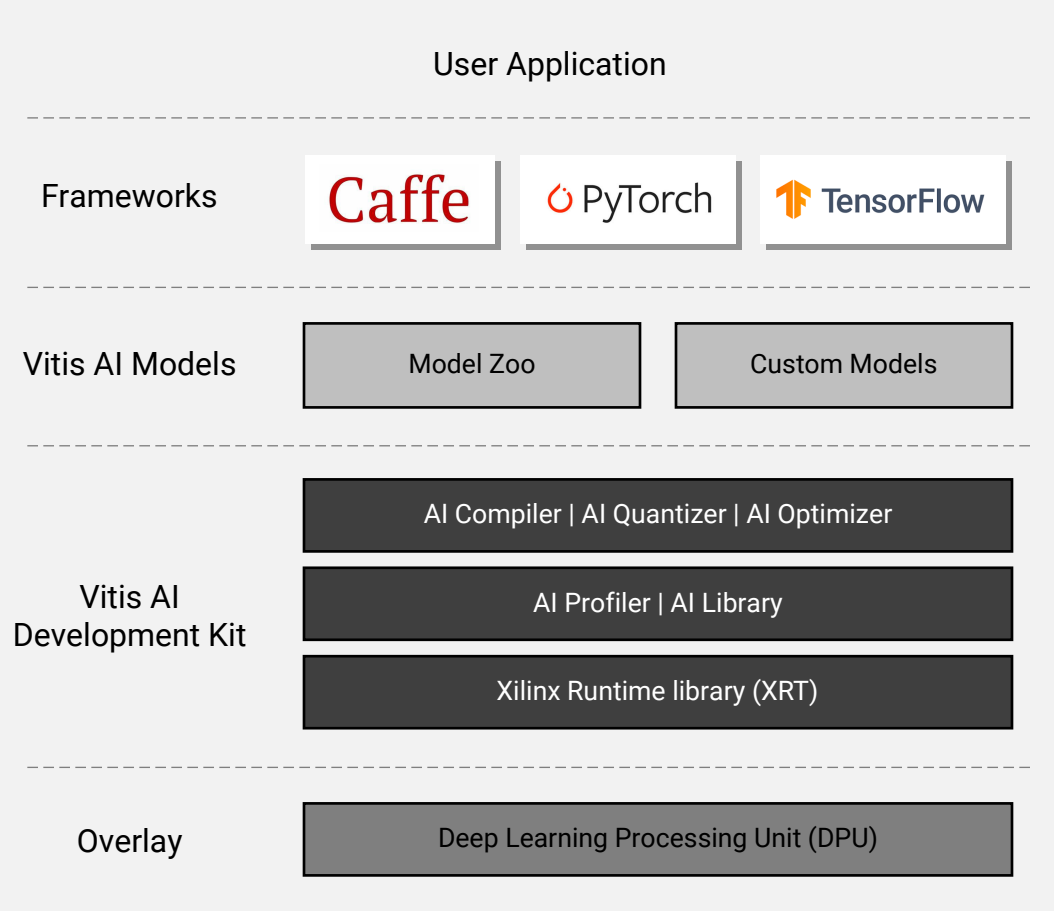


Vitis AI Development Kit

The Vitis AI development environment is used for AI inference on Xilinx hardware platforms. It consists of optimized IP cores, tools, libraries, models, and example designs.

As shown in the following figure, the Vitis AI development kit consists of AI Compiler, AI Quantizer, AI Optimizer, AI Profiler, AI Library, and Xilinx Runtime Library (XRT).

Figure 6: **Vitis AI Stack**



For more information of the Vitis AI development kit, see the *Vitis AI User Guide* (UG1414).

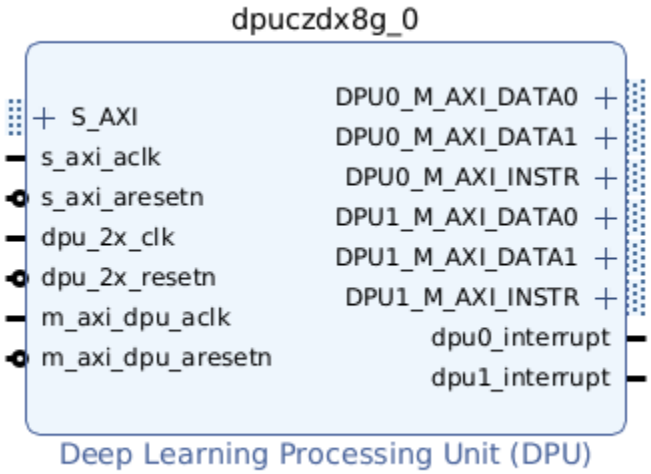
You can download the Vitis AI development kit for free from here.

Product Specification

Port Descriptions

The DPUCZDX8G top-level interfaces are shown in the following figure.

Figure 7: **Two DPU Kernel Ports**



The DPUCZDX8G I/O signals are listed and described in the table below.

Table 1: **DPUCZDX8G 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 DPUCZDX8G. The frequency is twice that of m_axi_dpu_aclk.
dpu_2x_resetn	Reset	1	I	Active-Low reset for DSP blocks

Table 1: **DPUCZDX8G Signal Description** *(cont'd)*

Signal Name	Interface Type	Width	I/O	Description
m_axi_dpu_aclk	Clock	1	I	Input clock used for DPUCZDX8G general logic.
m_axi_dpu_aresetn	Reset	1	I	Active-Low reset for DPUCZDX8G general logic
DPUx_M_AXI_INSTR	Memory mapped AXI master interface	32	I/O	32-bit memory mapped AXI interface for DPUCZDX8G instructions.
DPUx_M_AXI_DATA0	Memory mapped AXI master interface	128	I/O	128-bit for Zynq UltraScale+ MPSoC series.
DPUx_M_AXI_DATA1	Memory mapped AXI master interface	128	I/O	128-bit for Zynq UltraScale+ MPSoC series.
dpux_interrupt	Interrupt	1	O	Active-High interrupt output from DPUCZDX8G.
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 DPUCZDX8G 2x clock when DPUCZDX8G 2x clock gating is enabled.

Notes:

1. The softmax interface only appears when the softmax option in the DPUCZDX8G is enabled.

Register Space

The DPUCZDX8G IP implements registers in programmable logic. The following tables show the DPUCZDX8G IP registers. These registers are accessible from the APU through the S_AXI interface.

reg_dpu_reset

The reg_dpu_reset register controls the resets of all DPUCZDX8G cores integrated in the DPUCZDX8G IP. The lower four bits of this register control the reset of up to four DPUCZDX8G cores. All the reset signals are active-High. The details of reg_dpu_reset are shown in the following table.

Table 2: **reg_dpu_reset**

Register	Address Offset	Width	Type	Description
reg_dpu_reset	0x004	32	R/W	[n] – DPUCZDX8G core n reset

reg_dpu_isr

The reg_dpu_isr register represents the interrupt status of all cores in the DPUCZDX8G IP. The lower four bits of this register shows the interrupt status of up to four DPUCZDX8G cores. The details of reg_dpu_irq are shown in the following table.

Table 3: **reg_dpu_isr**

Register	Address Offset	Width	Type	Description
reg_dpu_isr	0x608	32	R	[n] – DPUCZDX8G core n interrupt status

reg_dpu_start

The reg_dpu_start register is the start signal for a DPUCZDX8G core. There is one start register for each DPUCZDX8G core. The details of reg_dpu_start are shown in the following table.

Table 4: **reg_dpu_start**

Register	Address Offset	Width	Type	Description
reg_dpu0_start	0x220	32	R/W	DPUCZDX8G core0 start signal.
reg_dpu1_start	0x320	32	R/W	DPUCZDX8G core1 start signal.
reg_dpu2_start	0x420	32	R/W	DPUCZDX8G core2 start signal.
reg_dpu3_start	0x520	32	R/W	DPUCZDX8G core3 start signal.

reg_dpu_instr_addr

The reg_dpu_instr_addr register is used to indicate the instruction address of a DPUCZDX8G core. Each DPUCZDX8G core has a reg_dpu_instr_addr register. Only the lower 28-bits are valid. In the DPUCZDX8G processor, the real instruction-fetch address is a 40-bit signal which consists of the lower 28 bits of reg_dpu_instr_addr followed by 12 zero bits. The available instruction address for DPU ranges from 0x1000 to 0xFFFF_FFFF_FFFF_F000. The details of reg_dpu_instr_addr are shown in the following table.

Table 5: **reg_dpu_instr_addr**

Register	Address Offset	Width	Type	Description
reg_dpu0_instr_addr	0x20C	32	R/W	Start address in external memory for DPUCZDX8G core0 instructions. The lower 28-bit is valid.
reg_dpu1_instr_addr	0x30C	32	R/W	Start address in external memory for DPUCZDX8G core1 instructions. The lower 28-bit is valid.

Table 5: **reg_dpu_instr_addr** *(cont'd)*

Register	Address Offset	Width	Type	Description
reg_dpu2_instr_addr	0x40C	32	R/W	Start address in external memory for DPUCZDX8G core2 instructions. The lower 28-bit is valid.
reg_dpu3_instr_addr	0x50C	32	R/W	Start address in external memory for DPUCZDX8G core3 instructions. The lower 28-bit is valid.

reg_dpu_base_addr

The reg_dpu_base_addr register is used to indicate the address of input image and parameters for each DPUCZDX8G in external memory. The width of a DPUCZDX8G base address is 40 bits so it can support an address space up to 1 TB. All registers are 32 bits wide, so two registers are required to represent a 40-bit wide base address. reg_dpu0_base_addr0_l represents the lower 32 bits of base_address0 in DPUCZDX8G core0 and reg_dpu0_base_addr0_h represents the upper eight bits of base_address0 in DPUCZDX8G core0.

There are eight groups of DPUCZDX8G base addresses for each DPUCZDX8G core and thus 32 groups of DPUCZDX8G base addresses for up to four DPUCZDX8G cores. The details of reg_dpu_base_addr are shown in the following table.

Table 6: **reg_dpu_base_addr**

Register	Address Offset	Width	Type	Description
reg_dpu0_base_addr0_l	0x224	32	R/W	The lower 32 bits of base_address0 of DPUCZDX8G core0.
reg_dpu0_base_addr0_h	0x228	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address0 of DPUCZDX8G core0.
reg_dpu0_base_addr1_l	0x22C	32	R/W	The lower 32 bits of base_address1 of DPUCZDX8G core0.
reg_dpu0_base_addr1_h	0x230	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address1 of DPUCZDX8G core0.
reg_dpu0_base_addr2_l	0x234	32	R/W	The lower 32 bits of base_address2 of DPUCZDX8G core0.
reg_dpu0_base_addr2_h	0x238	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address2 of DPUCZDX8G core0.
reg_dpu0_base_addr3_l	0x23C	32	R/W	The lower 32 bits of base_address3 of DPUCZDX8G core0.
reg_dpu0_base_addr3_h	0x240	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address3 of DPUCZDX8G core0.
reg_dpu0_base_addr4_l	0x244	32	R/W	The lower 32 bits of base_address4 of DPUCZDX8G core0.

Table 6: **reg_dpu_base_addr** *(cont'd)*

Register	Address Offset	Width	Type	Description
reg_dpu0_base_addr4_h	0x248	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address4 of DPUCZDX8G core0.
reg_dpu0_base_addr5_l	0x24C	32	R/W	The lower 32 bits of base_address5 of DPUCZDX8G core0.
reg_dpu0_base_addr5_h	0x250	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address5 of DPUCZDX8G core0.
reg_dpu0_base_addr6_l	0x254	32	R/W	The lower 32 bits of base_address6 of DPUCZDX8G core0.
reg_dpu0_base_addr6_h	0x258	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address6 of DPUCZDX8G core0.
reg_dpu0_base_addr7_l	0x25C	32	R/W	The lower 32 bits of base_address7 of DPUCZDX8G core0.
reg_dpu0_base_addr7_h	0x260	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address7 of DPUCZDX8G core0.
reg_dpu1_base_addr0_l	0x324	32	R/W	The lower 32 bits of base_address0 of DPUCZDX8G core1.
reg_dpu1_base_addr0_h	0x328	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address0 of DPUCZDX8G core1.
reg_dpu1_base_addr1_l	0x32C	32	R/W	The lower 32 bits of base_address1 of DPUCZDX8G core1.
reg_dpu1_base_addr1_h	0x330	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address1 of DPUCZDX8G core1.
reg_dpu1_base_addr2_l	0x334	32	R/W	The lower 32 bits of base_address2 of DPUCZDX8G core1.
reg_dpu1_base_addr2_h	0x338	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address2 of DPUCZDX8G core1.
reg_dpu1_base_addr3_l	0x33C	32	R/W	The lower 32 bits of base_address3 of DPUCZDX8G core1.
reg_dpu1_base_addr3_h	0x340	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address3 of DPUCZDX8G core1.
reg_dpu1_base_addr4_l	0x344	32	R/W	The lower 32 bits of base_address4 of DPUCZDX8G core1.
reg_dpu1_base_addr4_h	0x348	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address4 of DPUCZDX8G core1.
reg_dpu1_base_addr5_l	0x34C	32	R/W	The lower 32 bits of base_address5 of DPUCZDX8G core1.
reg_dpu1_base_addr5_h	0x350	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address5 of DPUCZDX8G core1.
reg_dpu1_base_addr6_l	0x354	32	R/W	The lower 32 bits of base_address6 of DPUCZDX8G core1.

Table 6: **reg_dpu_base_addr** *(cont'd)*

Register	Address Offset	Width	Type	Description
reg_dpu1_base_addr6_h	0x358	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address6 of DPUCZDX8G core1.
reg_dpu1_base_addr7_l	0x35C	32	R/W	The lower 32 bits of base_address7 of DPUCZDX8G core1.
reg_dpu1_base_addr7_h	0x360	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address7 of DPUCZDX8G core1.
reg_dpu2_base_addr1_l	0x42C	32	R/W	The lower 32 bits of base_address1 of DPUCZDX8G core2.
reg_dpu2_base_addr1_h	0x430	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address1 of DPUCZDX8G core2.
reg_dpu2_base_addr2_l	0x434	32	R/W	The lower 32 bits of base_address2 of DPUCZDX8G core2.
reg_dpu2_base_addr2_h	0x438	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address2 of DPUCZDX8G core2.
reg_dpu2_base_addr3_l	0x43C	32	R/W	The lower 32 bits of base_address3 of DPUCZDX8G core2.
reg_dpu2_base_addr3_h	0x440	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address3 of DPUCZDX8G core2.
reg_dpu2_base_addr4_l	0x444	32	R/W	The lower 32 bits of base_address4 of DPUCZDX8G core2.
reg_dpu2_base_addr4_h	0x448	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address4 of DPUCZDX8G core2.
reg_dpu2_base_addr5_l	0x44C	32	R/W	The lower 32 bits of base_address5 of DPUCZDX8G core2.
reg_dpu2_base_addr5_h	0x450	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address5 of DPUCZDX8G core2.
reg_dpu2_base_addr6_l	0x454	32	R/W	The lower 32 bits of base_address6 of DPUCZDX8G core2.
reg_dpu2_base_addr6_h	0x458	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address6 of DPUCZDX8G core2.
reg_dpu2_base_addr7_l	0x45C	32	R/W	The lower 32 bits of base_address7 of DPUCZDX8G core2.
reg_dpu2_base_addr7_h	0x460	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address7 of DPUCZDX8G core2.
reg_dpu3_base_addr0_l	0x524	32	R/W	The lower 32 bits of base_address0 of DPUCZDX8G core3.
reg_dpu3_base_addr0_h	0x528	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address0 of DPUCZDX8G core3.
reg_dpu3_base_addr1_l	0x52C	32	R/W	The lower 32 bits of base_address1 of DPUCZDX8G core3.

Table 6: **reg_dpu_base_addr** *(cont'd)*

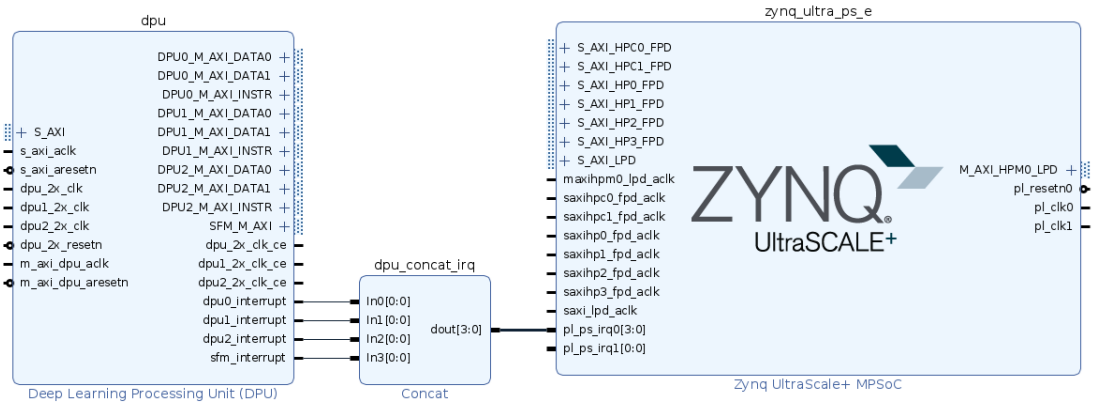
Register	Address Offset	Width	Type	Description
reg_dpu3_base_addr1_h	0x530	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address1 of DPUCZDX8G core3.
reg_dpu3_base_addr2_l	0x534	32	R/W	The lower 32 bits of base_address2 of DPUCZDX8G core3.
reg_dpu3_base_addr2_h	0x538	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address2 of DPUCZDX8G core3.
reg_dpu3_base_addr3_l	0x53C	32	R/W	The lower 32 bits of base_address3 of DPUCZDX8G core3.
reg_dpu3_base_addr3_h	0x540	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address3 of DPUCZDX8G core3.
reg_dpu3_base_addr4_l	0x544	32	R/W	The lower 32 bits of base_address4 of DPUCZDX8G core3.
reg_dpu3_base_addr4_h	0x548	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address4 of DPUCZDX8G core3.
reg_dpu3_base_addr5_l	0x54C	32	R/W	The lower 32 bits of base_address5 of DPUCZDX8G core3.
reg_dpu3_base_addr5_h	0x550	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address5 of DPUCZDX8G core3.
reg_dpu3_base_addr6_l	0x554	32	R/W	The lower 32 bits of base_address6 of DPUCZDX8G core3
reg_dpu3_base_addr6_h	0x558	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address6 of DPUCZDX8G core3.
reg_dpu3_base_addr7_l	0x55C	32	R/W	The lower 32 bits of base_address7 of DPUCZDX8G core3.
reg_dpu3_base_addr7_h	0x560	32	R/W	The lower 8 bits in the register represent the upper 8 bits of base_address7 of DPUCZDX8G core3.

Interrupts

The DPUCZDX8G generates an interrupt to signal the completion of a task. A high state on reg_dpu0_start signals the start of a DPUCZDX8G task for DPUCZDX8G core0. At the end of the task, the DPUCZDX8G generates an interrupt and bit0 in reg_dpu_isr is set to 1. The position of the active bit in the reg_dpu_isr depends on the number of DPUCZDX8G cores. For example, when DPUCZDX8G core1 finishes a task while DPUCZDX8G core0 is still working, reg_dpu_isr would maintain `2'b10`.

There will be DPU_NUM of `dpux_interrupt` signals for each of DPUCZDX8G cores. Theses signals shall be routed through a concat block, and then be connected to the PS. The reference connection is shown here.

Figure 8: **Reference Connection for DPUCZDX8G Interrupt**



***Note*:**

1. If the softmax option is enabled, then the softmax interrupt should be correctly connected to the PS according to the device tree description.
2. irq7~irq0 corresponds to pl_ps_irq0[7:0].
3. irq15~irq8 corresponds to pl_ps_irq1[7:0].

DPU Configuration

Introduction

The DPUCZDX8G IP provides some user-configurable parameters to optimize resource usage and customize different features. Different configurations can be selected for DSP slices, LUT, block RAM, and UltraRAM usage based on the amount of available programmable logic resources. There are also options for additional functions, such as channel augmentation, average pooling, depthwise convolution, and softmax. Furthermore, there is an option to determine the number of DPUCZDX8G cores that will be instantiated in a single DPUCZDX8G IP.

The deep neural network features and the associated parameters supported by the DPUCZDX8G are shown in the following table.

A configuration file named `arch.json` is generated during the Vivado or Vitis flow. The `arch.json` file is used by the Vitis AI Compiler for model compilation. For more information of Vitis AI Compiler, see refer to the *Vitis AI User Guide* (UG1414).

In the Vivado flow, the `arch.json` file is located at `$TRD_HOME/prj/Vivado/srcs/top/ip/top_dpu_0/arch.json`.

In the Vitis flow, the `arch.json` file is located at `$TRD_HOME/prj/Vitis/binary_container_1/link/vivado/vpl/prj/prj.gen/sources_1/bd/zcu102_base/ip/zcu102_base_DPUCZDX8G_1_0/arch.json`.

Note: `$TRD_home = Vitis-AI/dsa/DPU-TRD/`

Table 7: **Deep Neural Network Features and Parameters Supported by the DPUCZDX8G**

Features	Description	
Convolution	Kernel Sizes	w, h: [1, 16]
	Strides	w, h: [1, 8]
	Padding	w: [0, kernel_w-1] h: [0, kernel_h-1]
	Input Size	Arbitrary
	Input Channel	1~256 * channel_parallel
	Output Channel	1~256 * channel_parallel
	Activation	ReLU, ReLU6 and LeakyReLU
	Dilation	dilation * input_channel ≤ 256 * channel_parallel && stride_w == 1 && stride_h == 1
	Constraint*	kernel_w * kernel_h * (ceil(input_channel / channel_parallel)) <= bank_depth
Depthwise Convolution	Kernel Sizes	w, h: [1, 16]
	Strides	w, h: [1, 8]
	Padding	w: [0, kernel_w-1] h: [0, kernel_h-1]
	Input Size	Arbitrary
	Input Channel	1~256 * channel_parallel
	Output Channel	1~256 * channel_parallel
	Activation	ReLU, ReLU6
	Dilation	dilation * input_channel ≤ 256 * channel_parallel && stride_w == 1 && stride_h == 1
	Constraint*	kernel_w * kernel_h * (ceil(input_channel / channel_parallel)) <= bank_depth
Transposed Convolution	Kernel Sizes	kernel_w/stride_w: [1, 16] kernel_h/stride_h: [1, 16]
	Strides	
	Padding	w: [0, kernel_w-1] h: [0, kernel_h-1]
	Input Size	Arbitrary
	Input Channel	1~256 * channel_parallel
	Output Channel	1~256 * channel_parallel
	Activation	ReLU, ReLU6 and LeakyReLU
Depthwise Transposed Convolution	Kernel Sizes	kernel_w/stride_w: [1, 16] kernel_h/stride_h: [1, 16]
	Strides	
	Padding	w: [0, kernel_w-1] h: [0, kernel_h-1]
	Input Size	Arbitrary
	Input Channel	1~256 * channel_parallel
	Output Channel	1~256 * channel_parallel
	Activation	ReLU, ReLU6

Table 7: **Deep Neural Network Features and Parameters Supported by the DPUCZDX8G** *(cont'd)*

Features	Description	
Max Pooling	Kernel Sizes	w, h: [1, 8]
	Strides	w, h: [1, 8]
	Padding	w: [0, kernel_w-1] h: [0, kernel_h-1]
Average Pooling	Kernel Sizes	w, h: [2, 8] w==h
	Strides	w, h: [1, 8]
	Padding	w: [0, kernel_w-1] h: [0, kernel_h-1]
Max Reduce (max pooling for large size)	Kernel Sizes	w, h: [1, 256]
	Strides	Equals to kernel size
	Padding	Not supported
Elementwise-Sum	Input channel	1~256 * channel_parallel
	Input size	Arbitrary
	Feature Map Number	1~4
Elementwise-Multiply	Input channel	1~256 * channel_parallel
	Input size	Arbitrary
	Feature Map Number	2
Concat	Output channel	1~256 * channel_parallel
Reorg	Strides	stride * stride * input_channel ≤ 256 * channel_parallel
Fully Connected	Input_channel	Input_channel ≤ 2048 * channel_parallel
	Output_channel	Arbitrary

Notes:

1. The parameter channel_parallel is determined by the DPUCZDX8G configuration. For example, channel_parallel for the B1152 is 12, and channel_parallel for B4096 is 16 (see Parallelism for Different Convolution Architectures table in Configuration Options section).
2. In some neural networks, the FC layer is connected with a Flatten layer. The Vitis AI compiler will automatically combine the Flatten+FC to a global CONV2D layer, and the CONV2D kernel size is directly equal to the input feature map size of Flatten layer. For this case, the input feature map size cannot exceed the limitation of the kernel size of CONV, otherwise an error will be generated during compilation.

   This limitation occurs only in the Flatten+FC situation. This will be optimized in future releases.
3. The bank_depth refers to the on-chip weight buffer depth. In all DPU architectures, the bank_depth of the feature map and weights is 2048.
4. Max Reduce is similar to Max Pooling but it is able to handle larger kernel sizes. The constraint is kernel_w * kernel_h * input_channel <= PP * CP * bank_depth.
5. If the Batch Normalization is quantized and can be transformed to a depthwise-conv2d equivalently, it would be transformed to depthwise-conv2d and the compiler would search for compilation opportunities to map the Batch Normalization into DPU implementations. Otherwise, the batch_norm would be executed by CPU.

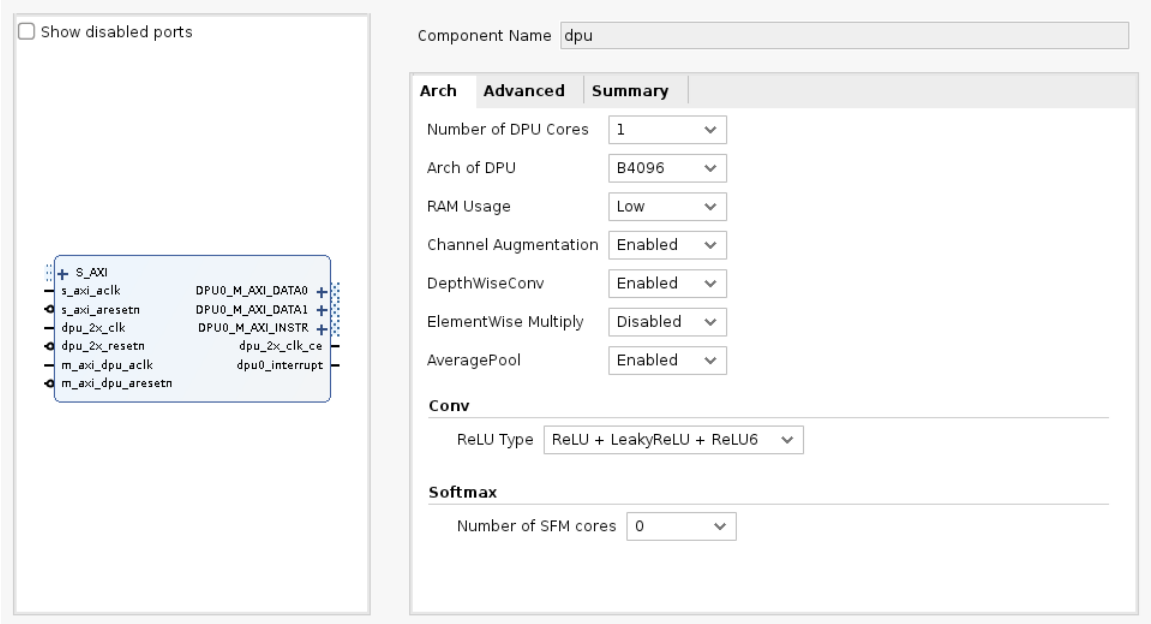
Related Information

Configuration Options

Configuration Options

The DPUCZDX8G can be configured with some predefined options, which includes the number of DPUCZDX8G cores, the convolution architecture, DSP cascade, DSP usage, and UltraRAM usage. These options allow you to set the DSP slice, LUT, block RAM, and UltraRAM usage. The following figure shows the configuration page of the DPUCZDX8G.

Figure 9: **DPUCZDX8G Configuration – Arch Tab**



The following sections describe the configuration options.

Number of DPUCZDX8G Cores

A maximum of four cores can be selected in one DPUCZDX8G IP. Multiple DPUCZDX8G cores can be used to achieve higher performance. Consequently, it consumes more programmable logic resources.

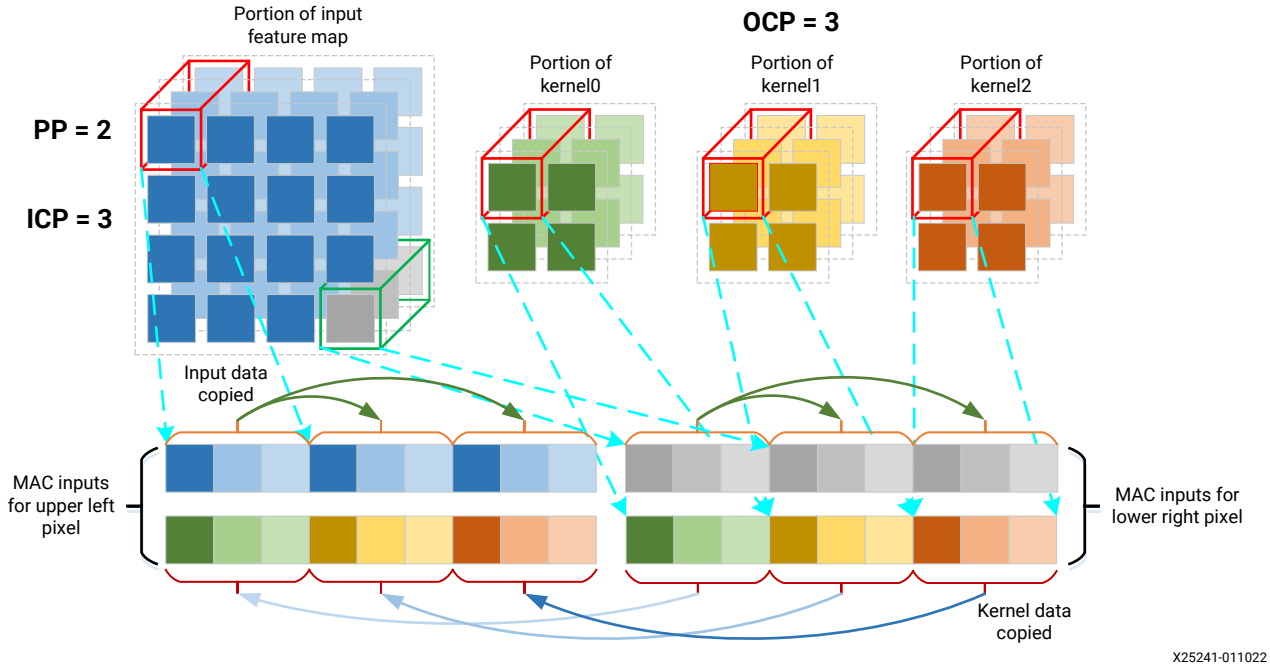
Contact your local Xilinx sales representative if you require more than four cores.

Architecture of the DPUCZDX8G

The DPUCZDX8G IP can be configured with various convolution architectures which are related to the parallelism of the convolution unit. The architectures for the DPUCZDX8G IP include B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096.

There are three dimensions of parallelism in the DPUCZDX8G convolution architecture: pixel parallelism, input channel parallelism, and output channel parallelism. The input channel parallelism is always equal to the output channel parallelism (this is equivalent to channel_parallel in the Table 8).

Figure 10: **Parallelism**



Visualizing pixel/input channel/output channel parallelism. In the figure, input channel parallelism (ICP) = 3; output channel parallelism (OCP) = 3; and pixel parallelism (PP) = 2. OCP is equivalent to the number of kernels used during a convolution computation. The pixels used in the figure are arbitrary to maintain clarity.

***Note*:** The elements used in the computation use 1 pixel from each channel (the red cuboids in the figure). With ICP = OCP = 3 and PP = 2, the number of convolution MACs per cycle is 3 * 3 * 2 = 18.

The different architectures require different programmable logic resources. The larger architectures can achieve higher performance with more resources. The parallelism for the different architectures is listed in the following table.

Table 8: **Parallelism for Different Convolution Architectures**

DPUCZDX8G 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

Table 8: **Parallelism for Different Convolution Architectures** *(cont'd)*

DPUCZDX8G Architecture	Pixel Parallelism (PP)	Input Channel Parallelism (ICP)	Output Channel Parallelism (OCP)	Peak Ops (operations/per clock)
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

Notes:

1. 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*ICP*OCP*2.

Resource Use

The resource utilization of a sample DPUCZDX8G single core project is as follows. The data is based on the ZCU102 platform with low RAM usage, depthwise convolution, average pooling, channel augmentation, average pool, leaky ReLU + ReLU6 features, and low DSP usage.

In the following tables, the triplet (PPxICPxOCP) after the architecture refers to the pixel parallelism, input channel parallelism, and output channel parallelism.

Table 9: **Resources of Different DPUCZDX8G Architectures**

DPUCZDX8G Architecture	LUT	Register	Block RAM	DSP
B512 (4x8x8)	27893	35435	73.5	78
B800 (4x10x10)	30468	42773	91.5	117
B1024 (8x8x8)	34471	50763	105.5	154
B1152 (4x12x12)	33238	49040	123	164
B1600 (8x10x10)	38716	63033	127.5	232
B2304 (8x12x12)	42842	73326	167	326
B3136 (8x14x14)	47667	85778	210	436
B4096 (8x16x16)	53540	105008	257	562

Another example of a DPUCZDX8G single core project is based on the ZCU104 platform. In this project, the image and weights buffer utilize UltraRAM. The project is configured with low RAM usage, depthwise convolution, average pooling, channel augmentation, average pool, leaky ReLU + ReLU6 features, and low DSP usage. The resource utilization of this project is as follows.

Table 10: **Resources of DPUCZDX8G using UltraRAM**

DPUCZDX8G Architecture	LUT	Register	Block RAM	UltraRAM	DSP
B512 (4x8x8)	27396	35251	1.5	18	78
B800 (4x10x10)	30356	42463	1.5	40	117
B1024 (8x8x8)	34134	50820	1.5	26	154
B1152 (4x12x12)	33103	49502	2	44	164
B1600 (8x10x10)	38526	63294	1.5	56	232
B2304 (8x12x12)	42538	74000	2	60	326
B3136 (8x14x14)	47270	85782	2	64	436
B4096 (8x16x16)	52681	104562	2	68	562

RAM Usage

The weights, bias, and intermediate features are buffered in the on-chip memory. The on-chip memory consists of RAM which can be instantiated as block RAM and UltraRAM. The RAM Usage option determines the total amount of on-chip memory used in different DPUCZDX8G architectures, and the setting is for all the DPUCZDX8G cores in the DPUCZDX8G IP. High RAM Usage means that the on-chip memory block will be larger, allowing the DPUCZDX8G more flexibility in handling the intermediate data. High RAM Usage implies higher performance in each DPUCZDX8G core. The number of BRAM36K blocks used in different architectures for low and high RAM Usage is illustrated in the following table.

***Note*:** The DPUCZDX8G instruction set for different options of RAM Usage is different. When the RAM Usage option is modified, the DPUCZDX8G instructions file should be regenerated by recompiling the neural network. The following results are based on a DPUCZDX8G with depthwise convolution.

Table 11: **Number of BRAM36K Blocks in Different Architectures for Each DPUCZDX8G Core**

DPUCZDX8G 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

Channel Augmentation

Channel augmentation is an optional feature for improving the efficiency of the DPUCZDX8G when the number of input channels is much lower than the available channel parallelism. For example, the input channel of the first layer in most CNNs is three, which does not fully use all the available hardware channels. If the number of input channels is larger than the channel parallelism, channel augmentation can still be utilized.

Thus, channel augmentation can improve the total efficiency for most CNNs, but it will cost extra logic resources. The following table illustrates the extra LUT resources used with channel augmentation and the statistics are for reference.

Table 12: **Extra LUTs of DPUCZDX8G with Channel Augmentation**

DPUCZDX8G Architecture	Extra LUTs with Channel Augmentation
B512(4x8x8)	3121
B800(4x10x10)	2624
B1024(8x8x8)	3133
B1152(4x12x12)	1744
B1600(8x10x10)	2476
B2304(8x12x12)	1710
B3136(8x14x14)	1946
B4096(8x16x16)	1701

DepthwiseConv

In standard convolution, each input channel needs to perform the operation with one specific kernel, and then the result is obtained by combining the results of all channels together.

In depthwise separable convolution, the operation is performed in two steps: depthwise convolution and pointwise convolution. Depthwise convolution is performed for each feature map separately as shown on the left side of the following figure. The next step is to perform pointwise convolution, which is the same as standard convolution with kernel size 1x1. The parallelism of depthwise convolution is half that of the pixel parallelism.

Figure 11: **Depthwise Convolution and Pointwise Convolution**

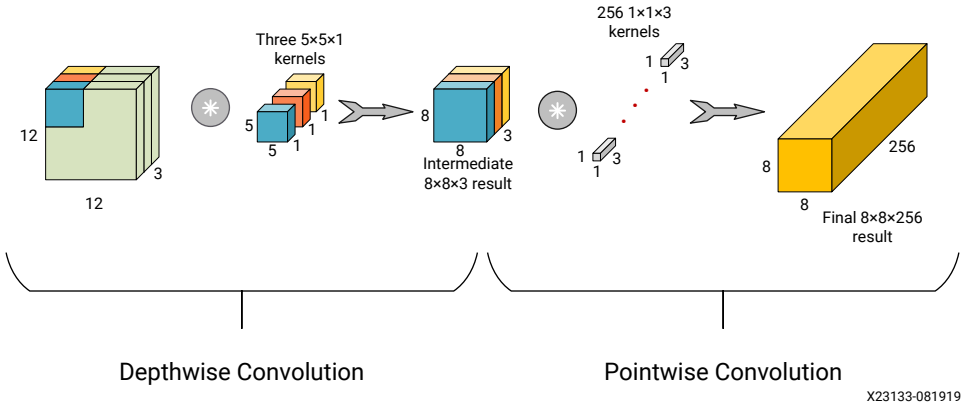


Table 13: **Extra resources of DPUCZDX8G with Depthwise Convolution**

DPUCZDX8G Architecture	Extra LUTs	Extra Block RAMs	Extra DSPs
B512(4x8x8)	1734	4	12
B800(4x10x10)	2293	4.5	15
B1024(8x8x8)	2744	4	24
B1152(4x12x12)	2365	5.5	18
B1600(8x10x10)	3392	4.5	30
B2304(8x12x12)	3943	5.5	36
B3136(8x14x14)	4269	6.5	42
B4096(8x16x16)	4930	7.5	48

ElementWise Multiply

The ElementWise Multiply calculates the Hadamard product of two input feature maps. The input channel of ElementWise Multiply ranges from 1 to 256 * channel_parallel.

The extra resources with ElementWise Multiply is listed in the following table.

Table 14: **Extra Resources of DPUCZDX8G with ElementWise Multiply**

DPUCZDX8G Architecture	Extra LUTs	Extra FFs[1]	Extra DSPs
B512(4x8x8)	159	-113	8
B800(4x10x10)	295	-93	10
B1024(8x8x8)	211	-65	8
B1152(4x12x12)	364	-274	12
B1600(8x10x10)	111	292	10
B2304(8x12x12)	210	-158	12
B3136(8x14x14)	329	-267	14

Table 14: **Extra Resources of DPUCZDX8G with ElementWise Multiply** *(cont'd)*

DPUCZDX8G Architecture	Extra LUTs	Extra FFs[1]	Extra DSPs
B4096(8x16x16)	287	78	16

Notes:

1. Negative numbers imply a relative decrease.

AveragePool

The AveragePool option determines whether the average pooling operation will be performed on the DPUCZDX8G or not. The supported sizes range from 2x2, 3x3, ..., to 8x8, with only square sizes supported.

The extra resources with Average Pool is listed in the following table.

Table 15: **Extra LUTs of DPUCZDX8G with Average Pool**

DPUCZDX8G Architecture	Extra LUTs
B512(4x8x8)	1507
B800(4x10x10)	2016
B1024(8x8x8)	1564
B1152(4x12x12)	2352
B1600(8x10x10)	1862
B2304(8x12x12)	2338
B3136(8x14x14)	2574
B4096(8x16x16)	3081

ReLU Type

The ReLU Type option determines which kind of ReLU function can be used in the DPUCZDX8G. ReLU and ReLU6 are supported by default. The option “ReLU + LeakyReLU + ReLU6“ means that LeakyReLU becomes available as an activation function.

***Note*:** LeakyReLU coefficient is fixed to 0.1.

Table 16: **Extra LUTs with ReLU + LeakyReLU + ReLU6 compared to ReLU+ReLU6**

DPU Architecture	Extra LUTs
B512(4x8x8)	347
B800(4x10x10)	725
B1024(8x8x8)	451
B1152(4x12x12)	780
B1600(8x10x10)	467
B2304(8x12x12)	706

Table 16: **Extra LUTs with ReLU + LeakyReLU + ReLU6 compared to ReLU+ReLU6** *(cont'd)*

DPU Architecture	Extra LUTs
B3136(8x14x14)	831
B4096(8x16x16)	925

Softmax

This option allows the softmax function to be implemented in hardware. It is a separate accelerator which has its own interface and runtime with int8 input and floating-point output data format. This softmax HW option is just packaged into DPU wrapper. The hardware implementation of softmax can be 160 times faster than a software implementation on MPSoC ARM in some application cases. Enabling this option depends on the customer chose algorithm model and desired throughput.

***Note*:** The maximum categories number of hardware softmax is 1023. If the categories number is greater than 1023, it is recommend to use the software softmax. For more information, refer to the *Vitis AI Library User Guide* (UG1354) .

When softmax is enabled, an AXI master interface named SFM_M_AXI and an interrupt port named `sfm_interrupt` will appear in the DPUCZDX8G IP. The softmax module uses `m_axi_dpu_aclk` as the AXI clock for SFM_M_AXI as well as for computation.

The extra resources with Softmax enabled are listed in the following table.

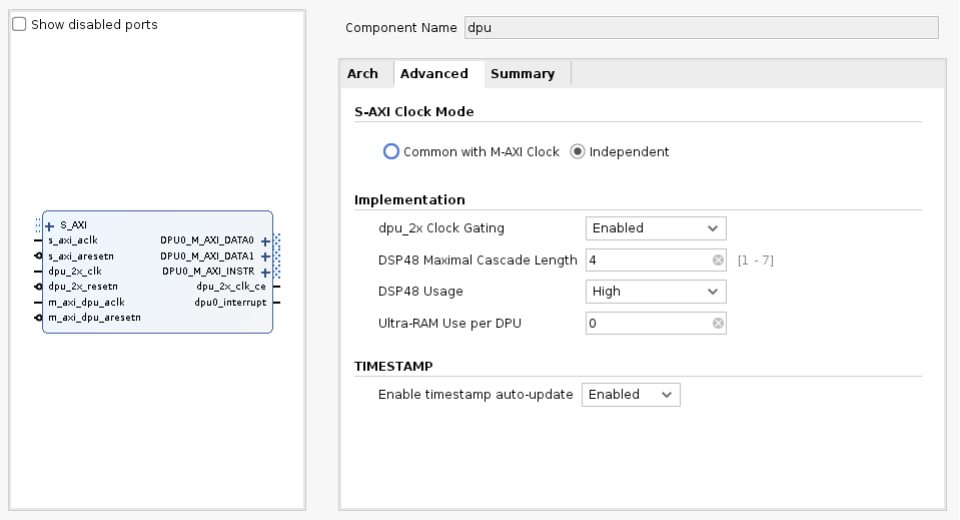
Table 17: **Extra Resources with Softmax**

IP Name	Extra LUTs	Extra FFs	Extra BRAMs	Extra DSPs
Softmax	9580	8019	4	14

Advanced Tab

The following figure shows the Advanced tab of the DPUCZDX8G configuration.

Figure 12: **DPUCZDX8G Configuration – Advanced Tab**



- **S-AXI Clock Mode:** `s_axi_aclk` is the S-AXI interface clock. When **Common with M-AXI Clock** is selected, `s_axi_aclk` shares the same clock as `m_axi_aclk` and the `s_axi_aclk` port is hidden. When **Independent** is selected, a clock different from `m_axi_aclk` must be provided.
- **dpu_2x Clock Gating:** `dpu_2x` clock gating is an option for reducing the power consumption of the DPUCZDX8G. When the option is enabled, a port named `dpu_2x_clk_ce` appears for each DPUCZDX8G core. The `dpu_2x_clk_ce` port should be connected to the `clk_dsp_ce` port in the dpu_clk_wiz IP. The `dpu_2x_clk_ce` signal can shut down the `dpu_2x_clk` when the computing engine in the DPUCZDX8G is idle. To generate the `clk_dsp_ce` port in the `dpu_clk_wiz` IP, the clocking wizard IP should be configured with specific options. For more information, see the Reference Clock Generation section.
- **DSP Cascade:** The maximum length of the DSP48E slice cascade chain can be set. Longer cascade lengths typically use fewer logic resources but might have worse timing. Shorter cascade lengths might not be suitable for small devices as they require more hardware resources. Xilinx recommends selecting the mid-value, which is four, in the first iteration and adjust the value if the timing is not met.
- **DSP Usage:** This allows you to select whether DSP48E slices will be used for accumulation in the DPUCZDX8G convolution module. When low DSP usage is selected, the DPUCZDX8G 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. The extra logic usage compared of high and low DSP usage is shown in the following table.

Table 18: **Extra Resources of Low DSP Usage Compared with High DSP Usage**

DPUCZDX8G Architecture	Extra LUTs	Extra Registers	Extra DSPs[1]
B512	1418	1903	-32

Table 18: **Extra Resources of Low DSP Usage Compared with High DSP Usage** *(cont'd)*

DPUCZDX8G Architecture	Extra LUTs	Extra Registers	Extra DSPs[1]
B800	1445	2550	-40
B1024	1978	3457	-64
B1152	1661	2525	-48
B1600	2515	4652	-80
B2304	3069	4762	-96
B3136	3520	6219	-112
B4096	3900	7359	-128

Notes:

1. Negative numbers imply a relative decrease.

- **UltraRAM:** There are two kinds of on-chip memory resources in Zynq® UltraScale+™ devices: block RAM and UltraRAM. The available amount of each memory type is device-dependent. Each block RAM consists of two 18K slices which can be configured as 9b*4096, 18b*2048, or 36b*1024. UltraRAM has a fixed-configuration of 72b*4096. A memory unit in the DPUCZDX8G has a width of ICP*8 bits and a depth of 2048. For the B1024 architecture, the ICP is 8, and the width of a memory unit is 8*8 bit. Each memory unit can then be instantiated with one UltraRAM block. When the ICP is greater than eight, each memory unit in the DPUCZDX8G needs at least two UltraRAM blocks.

  The DPUCZDX8G uses block RAM as the memory unit by default. For a target device with both block RAM and UltraRAM, configure the number of UltraRAM to determine how many UltraRAMs are used to replace some block RAMs. The number of UltraRAM should be set as a multiple of the number of UltraRAM required for a memory unit in the DPUCZDX8G. An example of block RAM and UltraRAM usage is shown in the Summary tab section.

- **Timestamp:** When enabled, the DPUCZDX8G records the time that the DPUCZDX8G project was synthesized. When disabled, the timestamp keeps the value at the moment of the last IP update.

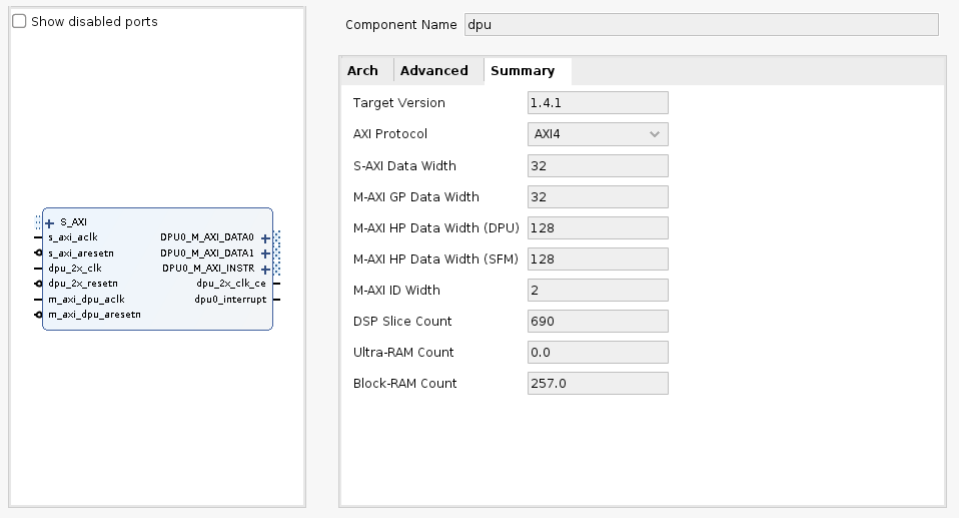
Related Information

Reference Clock Generation
Summary Tab

Summary Tab

A summary of the configuration settings is displayed in the Summary tab. The target version shows the DPUCZDX8G instruction set version number.

Figure 13: **Summary Tab of DPUCZDX8G Configuration**



DPUCZDX8G Performance on Different Devices

The following table shows the peak theoretical performance of the DPUCZDX8G on different devices.

Table 19: **DPUCZDX8G Performance GOPs per second (GOPS) on Different Devices**

Device	DPUCZDX8G Configuration	Frequency (MHz)	Peak Theoretical Performance (GOPS)
ZU2	B1152x1	370	426
ZU3	B2304x1	370	852
ZU5	B4096x1	350	1400
ZU7EV	B4096x2	330	2700
ZU9	B4096x3	333	4100

Performance of Different Models

In this section, the performance of several models is given for reference. The results shown in the following table were measured on a Xilinx® ZCU102 board with three B4096 cores with 16 threads running at 287 MHz.

Table 20: **Performance of Different Models**

Network Model	Workload (GOPs per image)	Input Image Resolution	Accuracy (DPUCZDX8G)[2]	Frame per second (FPS)
Inception-v1	3.2	224*224	Top-1: 0.6954	452.4
ResNet50	7.7	224*224	Top-1: 0.7338	163.4
MobileNet_v2	0.6	299*299	Top-1: 0.6352	587.2
SSD_ADAS_VEHICLE[1]	6.3	480*360	mAP: 0.4190	306.2
SSD_ADAS_PEDESTRIAN[1]	5.9	640*360	mAP: 0.5850	279.2
SSD_MobileNet_v2	6.6	480*360	mAP: 0.2940	124.7
YOLO-V3-VOC	65.4	416*416	mAP: 0.8153	43.6
YOLO-V3_ADAS[1]	5.5	512*256	mAP: 0.5301	239.7

Notes:

1. These models were pruned by the Vitis AI Optimizer.
2. Accuracy values are obtained using 8-bit quantization.

Unsupported Models

Some models in a specific DPUCZDX8G architecture may not be supported due to a large feature map size. Following is a list of unsupported models in our Model Zoo under different architectures.

Table 21: **Unsupported Models in Different DPUCZDX8G Architectures**

DPUCZDX8G Architecture	Unsupported models
B512	inception_resnet_v2_tf
	vgg_16_tf
	vgg_19_tf
	facerec_resnet20
	facerec_resnet64
	facerec-resnet20_mixed_pt
B800	vgg_16_tf
	vgg_19_tf
	facerec_resnet20
	facerec_resnet64
	facerec-resnet20_mixed_pt
	fadnet_pruned

Table 21: **Unsupported Models in Different DPUCZDX8G Architectures** *(cont'd)*

DPUCZDX8G Architecture	Unsupported models
B1024	inception_resnet_v2_tf
	vgg_16_tf
	vgg_19_tf
	facerec_resnet20
	facerec_resnet64
	facerec-resnet20_mixed_pt
B1152	vgg_16_tf
	vgg_19_tf
B1600	vgg_16_tf
	vgg_19_tf
	facerec_resnet20
	facerec_resnet64
	facerec-resnet20_mixed_pt
	fadnet_pruned
B2304	vgg_16_tf
	vgg_19_tf
B3136	vgg_16_tf
	vgg_19_tf

I/O Bandwidth Requirements

When different neural networks run on the DPUCZDX8G, the I/O bandwidth requirement will vary depending on which neural network is currently being executed. Even the I/O bandwidth requirement of different layers in one neural network will be different. The I/O bandwidth requirements for some neural networks, averaged by layer, have been tested with one DPUCZDX8G core running at full speed. The peak and average I/O bandwidth requirements of three different neural networks are shown in the table below. The table only shows the number of two commonly used DPUCZDX8G architectures (B1152 and B4096).

***Note*:** When multiple DPUCZDX8G cores run in parallel, each core might not be able to run at full speed due to the I/O bandwidth limitations.

Table 22: **I/O Bandwidth Requirements for B1152 and B4096**

Network Model	B1152		B4096	
	Peak (MB/s)	Average (MB/s)	Peak (MB/s)	Average (MB/s)
Inception-v1	1704	890	4626	2474
ResNet50	2052	1017	5298	3132

Table 22: **I/O Bandwidth Requirements for B1152 and B4096** *(cont'd)*

Network Model	B1152		B4096	
	Peak (MB/s)	Average (MB/s)	Peak (MB/s)	Average (MB/s)
SSD ADAS VEHICLE	1516	684	5724	2049
YOLO-V3-VOC	2076	986	6453	3290

If one DPUCZDX8G core needs to run at full speed, the peak I/O bandwidth requirement shall be met. The I/O bandwidth is mainly used for accessing data though the AXI master interfaces (DPU0_M_AXI_DATA0 and DPU0_M_AXI_DATA1).

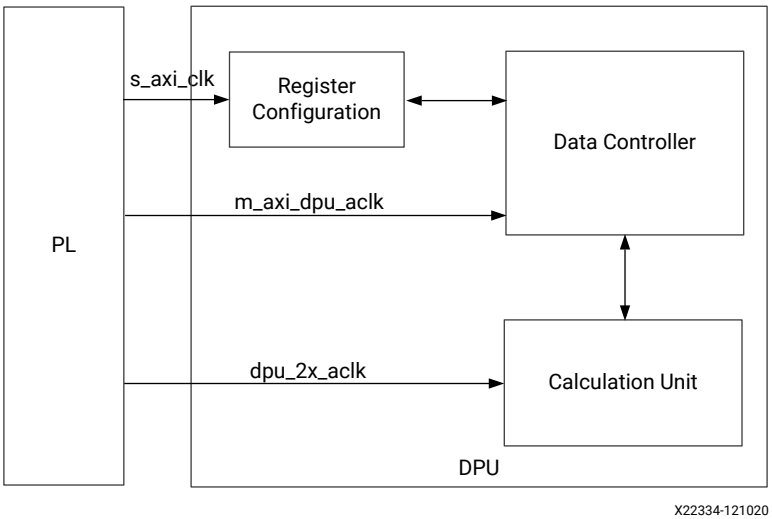
Clocking

There are three clock domains in the DPUCZDX8G IP: the register configuration, the data controller, and the computation unit. The three input clocks can be configured depending on the requirements. Therefore, the corresponding reset for the three input clocks must be configured correctly.

Clock Domain

The following figure shows the three clock domains.

Figure 14: **Clock Domains in the DPUCZDX8G**



Register Clock

s_axi_aclk is used for the register configuration module. This module receives the DPUCZDX8G configuration though the S_AXI interface. The S_AXI clock can be configured as common with the M-AXI clock or as an independent clock. The DPUCZDX8G configuration registers are updated at a very low frequency and most of those registers are set at the start of a task. The M-AXI is used as a high-frequency clock, Xilinx recommends setting the S-AXI clock as an independent clock with a frequency of 100 MHz.

In the Vitis flow, the platform may provide only two clocks for the DPUCZDX8G IP. In this case, the S_AXI clock must be configured as common with the M-AXI clock.

Data Controller Clock

The primary function of the data controller module is to schedule the data flow in the DPUCZDX8G IP. The data controller module works with m_axi_dpu_aclk. The data transfer between the DPUCZDX8G and external memory happens in the data controller clock domain, so m_axi_dpu_aclk is also the AXI clock for the AXI_MM master interface in the DPUCZDX8G IP. m_axi_dpu_aclk should be connected to the AXI_MM master clock.

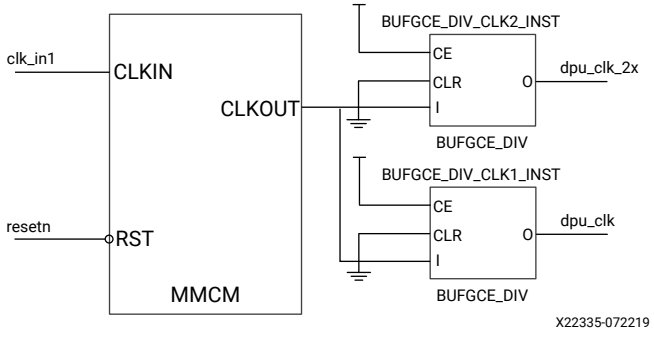
Computation Clock

The DSP slices in the computation unit module are in the dpu_2x_clk domain, which runs at twice the clock frequency of the data controller module. The two related clocks must be edge-aligned.

Reference Clock Generation

There are three input clocks for the DPUCZDX8G and the frequency of dpu_2x_clk should be twice that of m_axi_dpu_aclk. m_axi_dpu_aclk and dpu_2x_clk must be synchronous. The recommended circuit design is shown here.

Figure 15: **Reference Circuit**



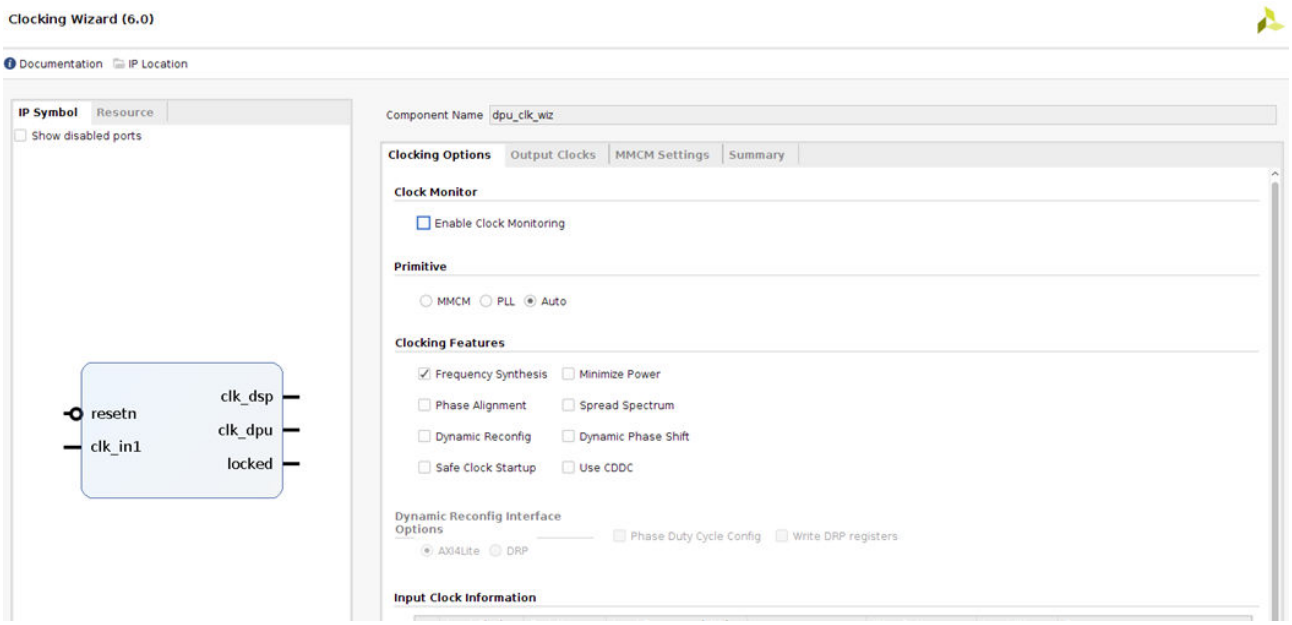
An MMCM and two BUFGCE_DIV blocks can be instantiated to design this circuit. The frequency of `clk_in1` is arbitrary and the frequency of output clock CLKOUT in the MMCM should be the frequency of `dpu_clk_2x`. BUFGCE_DIV_CLK1_INST divides the frequency of CLKOUT by two. `dpu_clk` and `dpu_clk_2x` are derived from the same clock, so they are synchronous. The two BUFGCE_DIVs reduce the skew between the two clocks, which helps with timing closure.

Configuring Clock Wizard

Instantiating the Xilinx clock wizard IP can implement the above circuit. In this reference design, the frequency of `s_axi_aclk` is set to 100 MHz and `m_axi_dpu_aclk` is set to 325 MHz. Therefore, the frequency of the `dpu_2x_clk` should be set to 650 MHz accordingly. The recommended configuration of the Clocking Options tab is shown in the following figure.

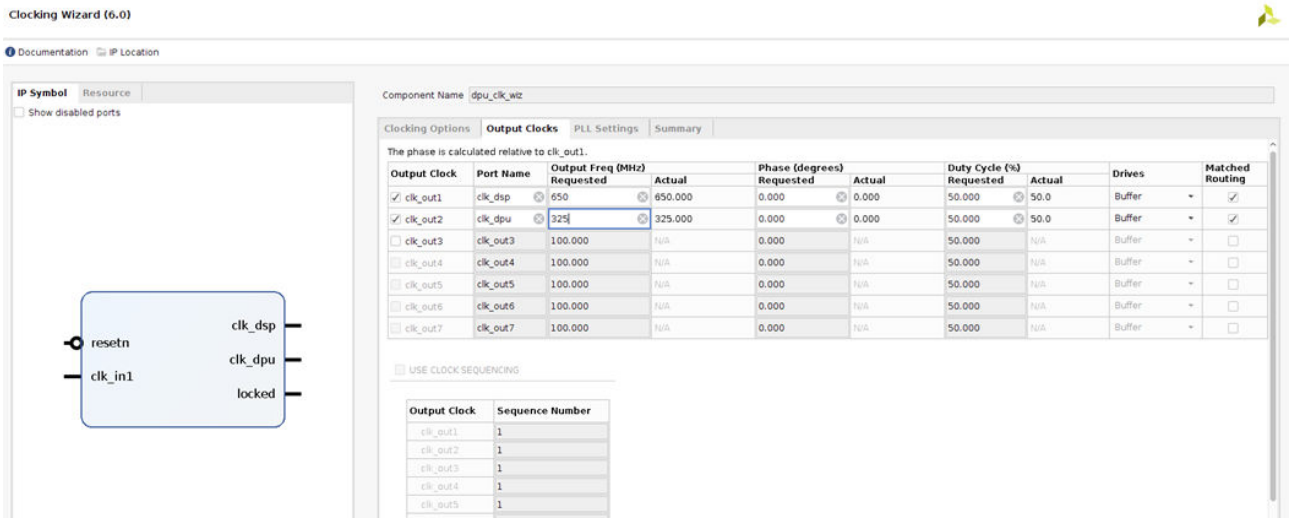
***Note*:** The parameter of the Primitive must be set to Auto.

Figure 16: **Recommended Clocking Options of Clock Wizard**



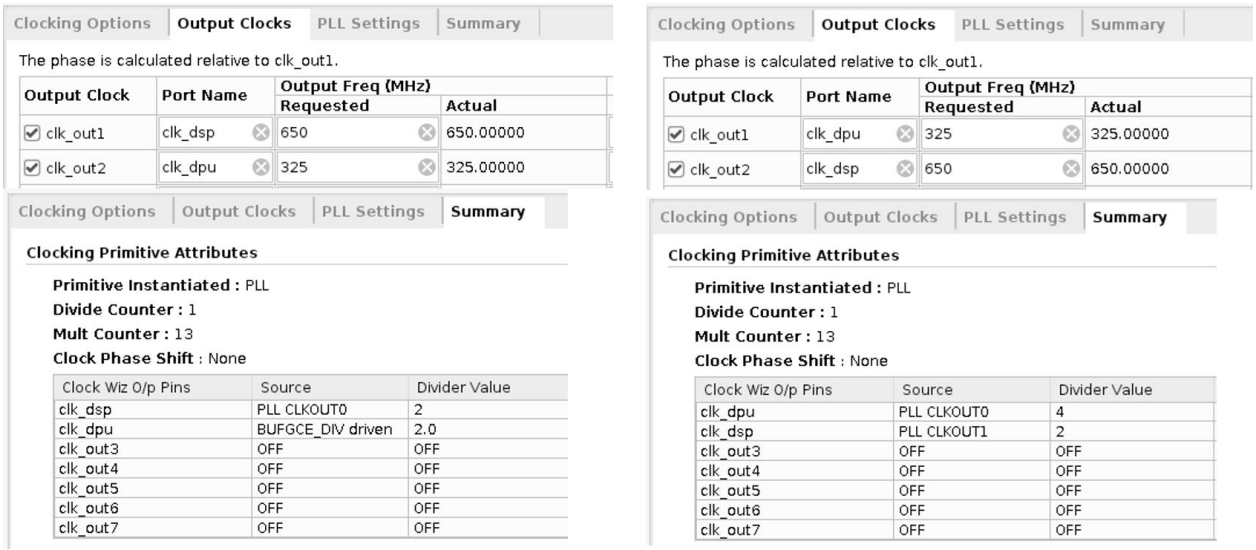
In addition, Matched Routing must be selected for `m_axi_dpu_aclk` and `dpu_2x_clk` in the Output Clocks tab of the Clock Wizard IP. Matched Routing significantly reduces the skew between clocks generated through BUFGCE_DIV blocks. The related configuration is shown in the following figure.

Figure 17: **Matched Routing in Clock Wizard**



***Note*:** Set the frequencies of the clkout from High to Low. Figure (a) shows the correct sequence. The settings in figure (a) achieved the dedicated clock design in the Summary page while the figure (b) did not. For more details, refer to the *Clocking Wizard LogiCORE IP Product Guide* (PG065).

Figure 18: **Comparison of clkout Frequency Sequence**

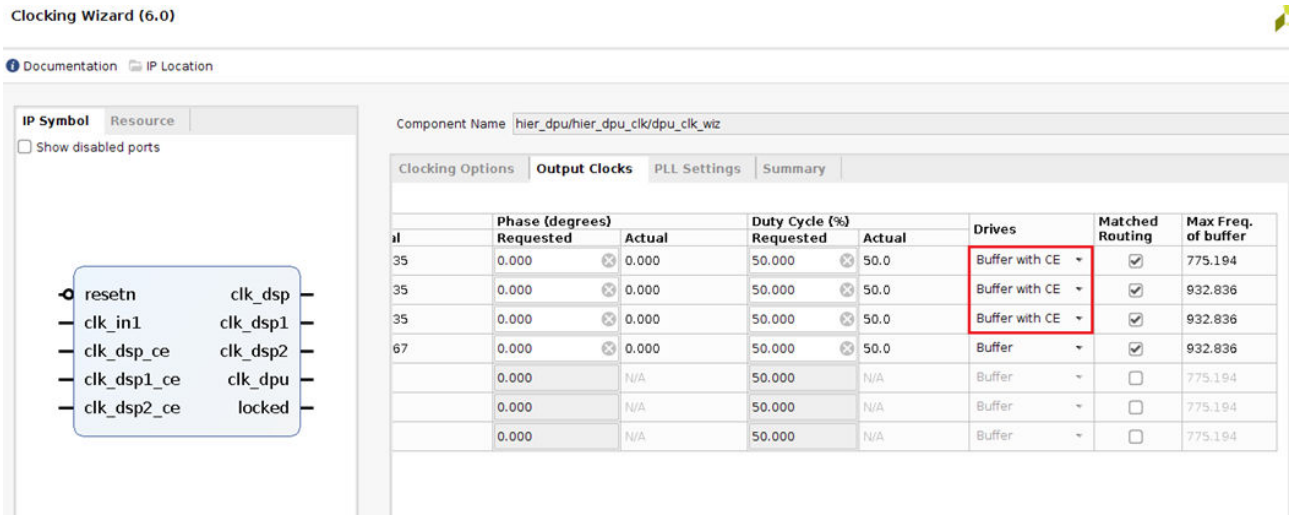


(a) (b)

Adding CE for dpu_2x_clk

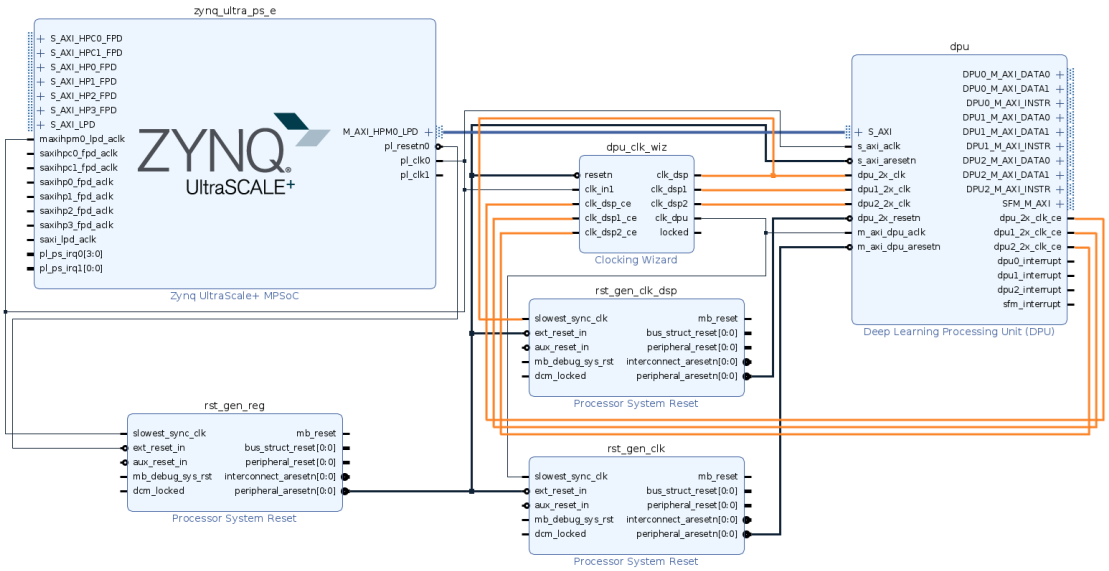
The `dpu_2x` clock gating option can reduce the power consumption of the DPUCZDX8G. When the option is enabled, the number of generated `clk_dsp` should be equal to the number of DPUCZDX8G cores. Each `clk_dsp` should be set as a buffer with CE in the clock wizard IP. As shown in the following figure, three `clk_dsp_ce` appear when the output clock is configured with the CE. To enable the `dpu_2x` clock gating function, each `clk_dsp_ce` port should be connected to the corresponding `dpu_2x_clk_ce` port in the DPUCZDX8G.

Figure 19: **Configure Clock Wizard with Buffer CE**



After configuring the clock wizard, the `clock_dsp_ce` should be connected to the corresponding port in the DPUCZDX8G. The connections are shown in the following figure.

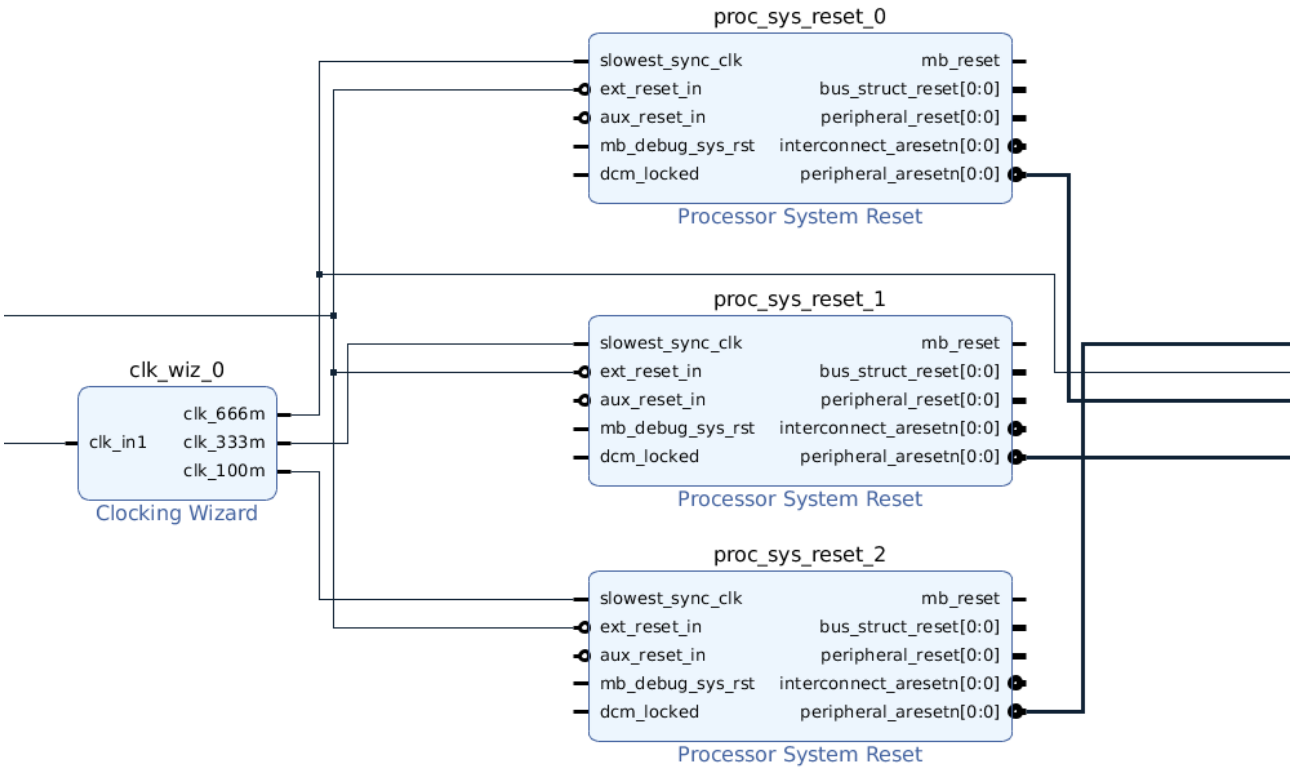
Figure 20: **Clock CE and DPUCZDX8G Connections**



Reset

There are three input clocks for the DPUCZDX8G IP and each clock has a corresponding reset. Each reset must be synchronous to its corresponding clock. If the related clocks and resets are not synchronized, the DPUCZDX8G might not work properly. A Processor System Reset IP block is recommended to generate a synchronized reset signal. The reference design is shown here.

Figure 21: **Reference Design for Resets**



Development Flow

Customizing and Generating the Core in the Zynq UltraScale+ MPSoC

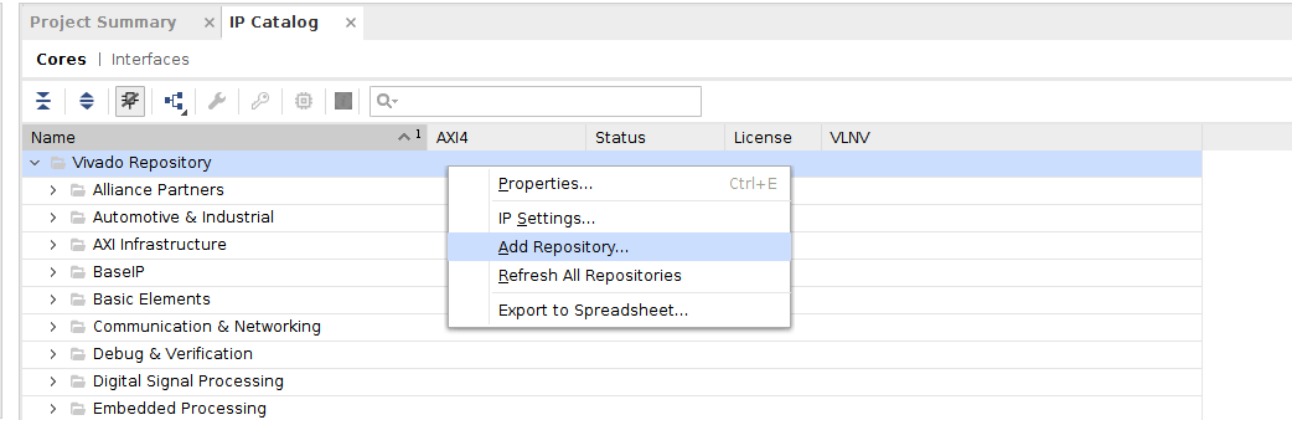
The following sections describe the development flow on how to use the DPUCZDX8G IP with the Vivado® Design Suite:

- Add the DPUCZDX8G into Repository or Upgrade the DPUCZDX8G from a Previous Version
- Add the DPUCZDX8G into the Block Design
- Configure DPUCZDX8G Parameters
- Connecting the DPUCZDX8G to the Processing System in the Zynq UltraScale+ MPSoC
- Assign Register Addresses
- Generate Bitstream
- Generate BOOT.BIN
- Device Tree

Add the DPUCZDX8G into Repository or Upgrade the DPUCZDX8G from a Previous Version

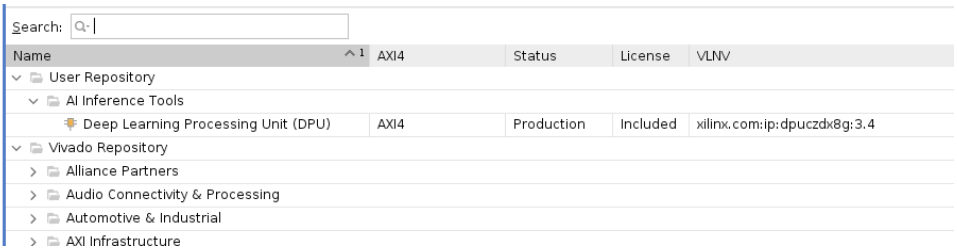
In the Vivado Integrated Design Environment (IDE), click **Project Manager→IP Catalog**. In the IP catalog tab, right-click and select **Add Repository** (see figure below), then select the location of the DPUCZDX8G IP.

Figure 22: **Add Repository**



The DPUCZDX8G IP will appear in the IP catalog page.

Figure 23: **DPUCZDX8G IP in Repository**



If there is an existing hardware project with an old version of the DPUCZDX8G, upgrading to the latest version is required, follow these steps:

1. Delete the old DPUCZDX8G IP in the block design and IP repository.
2. Add the new DPUCZDX8G IP into the IP repository.
3. Add the new DPUCZDX8G IP into the block design.

Add the DPUCZDX8G into the Block Design

Search for DPUCZDX8G in the block design interface and add the DPUCZDX8G IP into the block design. The procedure is shown in the following figures.

Figure 24: **Search DPUCZDX8G IP**

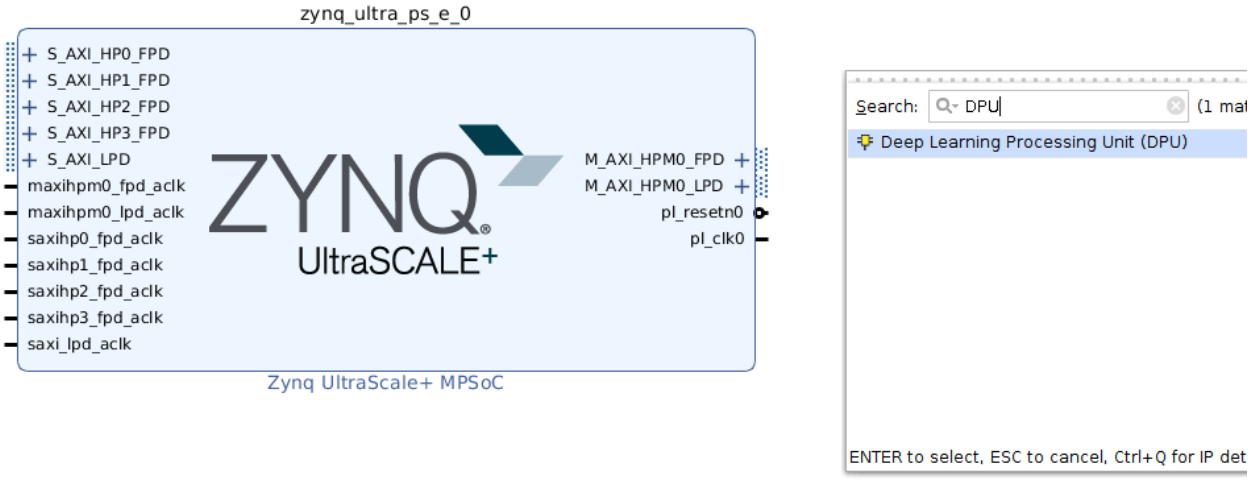
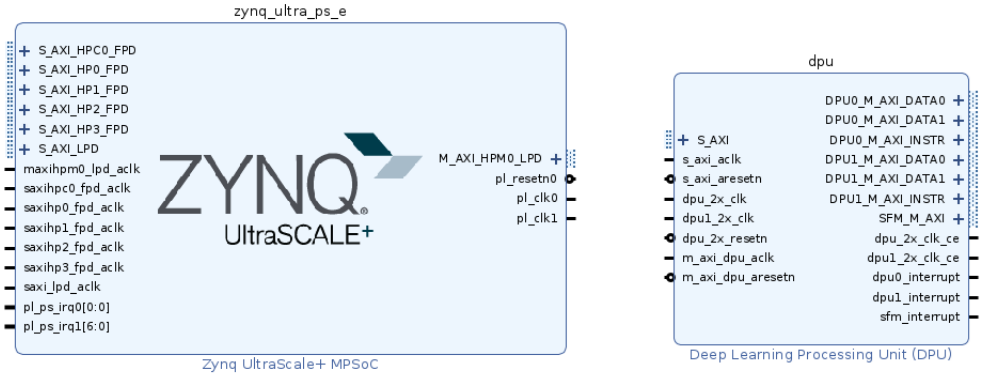


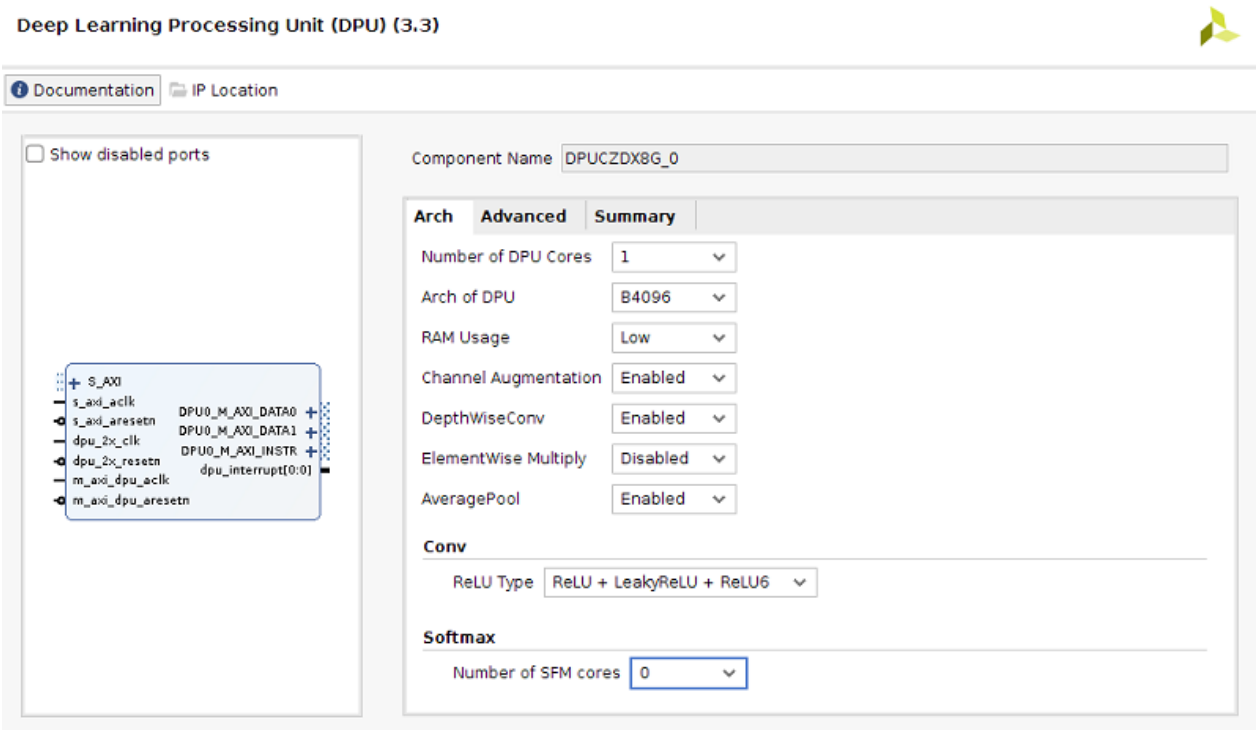
Figure 25: **DPUCZDX8G IP into Block Design**



Configure DPUCZDX8G Parameters

You can configure the DPUCZDX8G IP as shown in the following figure. Details about these parameters can be found in the Chapter 4: DPU Configuration section.

Figure 26: **Configure DPUCZDX8G**



Connecting the DPUCZDX8G to the Processing System in the Zynq UltraScale+ MPSoC

The DPUCZDX8G IP contains only one slave interface. The number of DPUCZDX8G cores depends on the parameter DPU_NUM. Each DPUCZDX8G core has three master interfaces, one for instruction fetch, and the other two for data access.

The DPUCZDX8G IP can be connected to the processing system (PS) with an AXI Interconnection IP as long as the DPUCZDX8G can correctly access the DDR memory space. Generally, when data is transferred through an Interconnect IP, the data transaction delay will increase. The delay incurred by the Interconnect will reduce the DPUCZDX8G performance. Therefore, Xilinx recommends that each master interface in the DPUCZDX8G is connected to the PS through a direct connection rather than through an AXI Interconnect IP when there are sufficient AXI slave ports available on the PS.

When the AXI slave ports of the PS are insufficient for the DPUCZDX8G, an AXI interconnect for connection is unavoidable. The two AXI master ports for data fetching are high bandwidth ports and the AXI master port for instruction fetching is a low bandwidth port. Typically, it is recommended that all the master ports for instruction fetching connect to the S_AXI_LPD of PS through one interconnect. The rest of the master ports for data fetching should be directly connected to the PS as much as possible. Xilinx recommends that the master ports of the DPUCZDX8G core with higher priority (smaller number, like DPU0) be directly connected to the slave ports of the PS with higher priority (smaller number, like S_AXI_HP0_FPD).

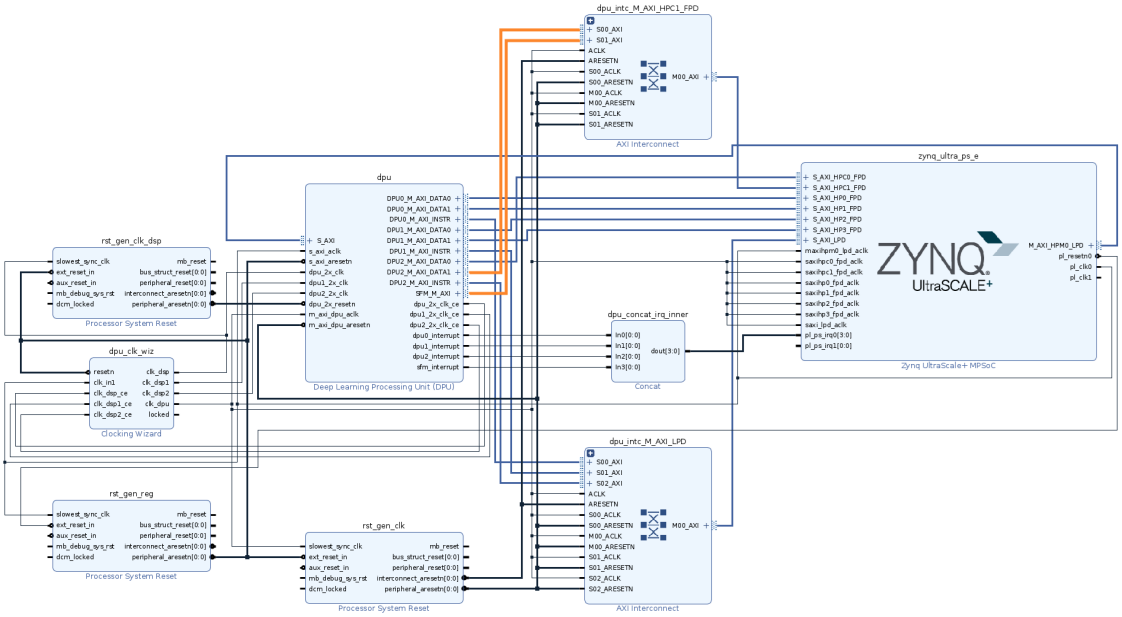
For example, if there are three DPUCZDX8G cores and one SFM core, there will be seven master ports, and four slave ports: S_AXI_HP1~3 and S_AXI_HPC0. A possible connection setup would be:

- DPU0_DATA0 to HP1
- DPU0_DATA1 to HP2
- DPU1_DATA0 and DPU1_DATA1 to HP3
- DPU2_DATA0, DPU2_DATA1, and SFM to HPC0

It is recommended that the slave port of DPUCZDX8G be connected to M_AXI_HPM0_LPD of the PS.

A reference connection between the DPUCZDX8G and PS in theZynq UltraScale+ MPSoC is shown here. The number of DPUCZDX8G core is set to three, and the Softmax function is enabled.

Figure 27: **DPUCZDX8G and PS Connections for Zynq UltraScale+ MPSoC**



Assign Register Addresses

When the DPUCZDX8G connection is complete, the next step is to assign the register address of the AXI slave interface. The minimum space needed for the DPUCZDX8G is 16 MB. The DPUCZDX8G slave interface can be assigned to any starting address accessible by the host CPU.

***Note*:** The DPUCZDX8G base address must be set with a range of 16 MB. The addresses in the device driver and device tree file must match those assigned in Vivado.

The reference address assignments of the DPUCZDX8G are shown here.

Figure 28: **DPUCZDX8G Address Assignment**

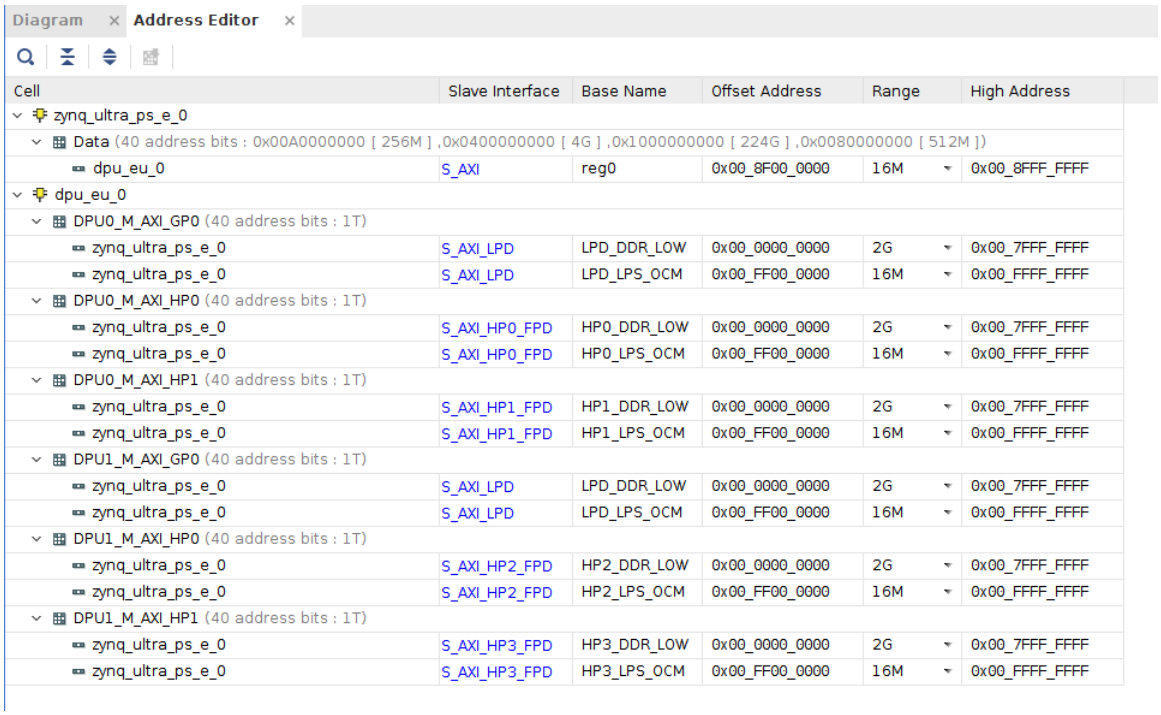


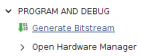
Diagram | Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
zynq_ultra_ps_e_0					
Data (40 address bits : 0x00A0000000 [256M] ,0x0400000000 [4G] ,0x1000000000 [224G] ,0x0080000000 [512M])					
dpu_eu_0	S_AXI	reg0	0x00_8F00_0000	16M	0x00_8FFF_FFFF
dpu_eu_0					
DPU0_M_AXI_GP0 (40 address bits : 1T)					
zynq_ultra_ps_e_0	S_AXI_LPD	LPD_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
zynq_ultra_ps_e_0	S_AXI_LPD	LPD_LPS_OCM	0x00_FF00_0000	16M	0x00_FFFF_FFFF
DPU0_M_AXI_HP0 (40 address bits : 1T)					
zynq_ultra_ps_e_0	S_AXI_HP0_FPD	HP0_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HP0_FPD	HP0_LPS_OCM	0x00_FF00_0000	16M	0x00_FFFF_FFFF
DPU0_M_AXI_HP1 (40 address bits : 1T)					
zynq_ultra_ps_e_0	S_AXI_HP1_FPD	HP1_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HP1_FPD	HP1_LPS_OCM	0x00_FF00_0000	16M	0x00_FFFF_FFFF
DPU1_M_AXI_GP0 (40 address bits : 1T)					
zynq_ultra_ps_e_0	S_AXI_LPD	LPD_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
zynq_ultra_ps_e_0	S_AXI_LPD	LPD_LPS_OCM	0x00_FF00_0000	16M	0x00_FFFF_FFFF
DPU1_M_AXI_HP0 (40 address bits : 1T)					
zynq_ultra_ps_e_0	S_AXI_HP2_FPD	HP2_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HP2_FPD	HP2_LPS_OCM	0x00_FF00_0000	16M	0x00_FFFF_FFFF
DPU1_M_AXI_HP1 (40 address bits : 1T)					
zynq_ultra_ps_e_0	S_AXI_HP3_FPD	HP3_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HP3_FPD	HP3_LPS_OCM	0x00_FF00_0000	16M	0x00_FFFF_FFFF

Generate Bitstream

Click **Generate Bitstream** in Vivado as shown below.

Figure 29: **Generate Bitstream**



Generate BOOT.BIN

To generate the `BOOT.BIN` file, use the Vivado® Design Suite or PetaLinux. For create the boot image using the Vivado Design Suite, refer to the *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209). For PetaLinux, refer to the *PetaLinux Tools Documentation: Reference Guide* (UG1144).

Device Tree

The device tree information is generated differently for the Vivado flow and Vitis flow.

In the Vivado flow, the DPUCZDX8G and Softmax IP are configured automatically by the PetaLinux.

In the Vitis flow, the device tree is included in the platform configuration. For more information about generating device tree in Vitis flow, see Update the Device Tree.

A device tree configuration sample for a Zynq UltraScale+ MPSoC is shown below:

```
&amba {
            ...
             dpu {
                 compatible = "xilinx,dpu";
                 base-addr = <0x8f000000>;//CHANGE THIS ACCORDING TO YOUR
DESIGN
                 dpucore {
                     compatible = "xilinx,dpucore";
                     interrupt-parent = <&intc>;
                     interrupts = <0x0 106 0x1 0x0 107 0x1>;
                     core-num = <0x2>;
                     };
                 };
                 softmax {
                     compatible = "xilinx, smfc";
                     interrupt-parent = <&intc>;
                     interrupts = <0x0 110 0x1>;
                     core-num = <0x1>;
                 ....
}
```

Customizing and Generating the Core in the Vitis IDE

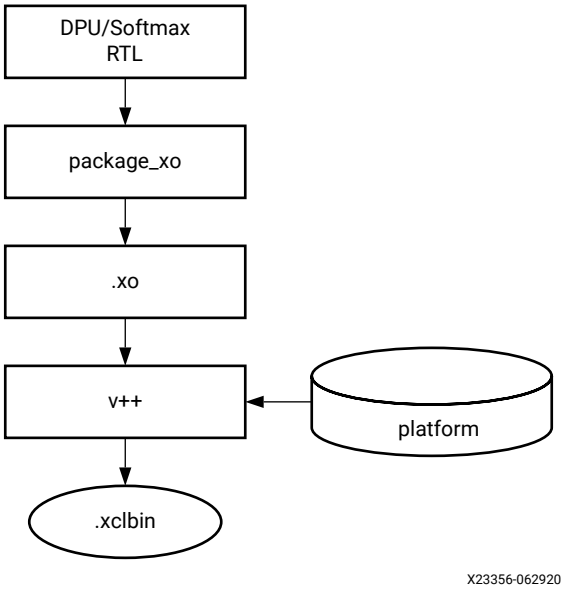
The Vitis™ embedded software development flow has the following two methods to generate the core:

1. GUI Flow
2. Command Flow

The command flow is recommended for its flexibility and ease of use. If you want to use the GUI flow, refer to here.

The build process for the DPUCZDX8G is shown in the following figure:

Figure 30: **Build Process for the DPUCZDX8G**



The following definitions describe the command flow and the process of using the DPUCZDX8G IP with the Vitis libraries:

- **Makefile and other scripts:** The DPUCZDX8G kernel is independently compiled to a Xilinx object (`.xo`) file. It is compiled using the `package_xo` utility. The RTL kernel wizard in the Vitis IDE can be used to simplify this process. The `.xo` file is linked with the hardware platform (shell) to create the FPGA binary (.xclbin). The v++ compiler automatically uses the Vivado® Design Suite tools to build the kernels to run on the FPGA platform.

  The Makefile and the other scripts are present in the Vitis DPU TRD.

- **Configure DPUCZDX8G Parameters:** You can modify the `Vitis-AI/DPU-TRD/prj/Vitis/dpu_conf.vh` file to configure the DPU parameters. See Chapter 3: Product Specification for more details on the DPU parameters.

- **Architecture:** Select the hardware architecture from the following:: B512, B800, B1024, B1600, B2304, B3136, and B4096. For the B4096, the definition is as follows:

  ```
  `define B4096
  ```

- **UltraRAM Number:** Modify the `dpu_config.vh` file to set the numbers. Enable `` `define URAM_ENABLE `` and `` `define URAM_DISABLE ``.

  When UltraRAM is enabled, set the following parameters:

  - `'define def_UBANK_IMG_N 5`
  - `'define def_UBANK_WGT_N 17`
  - `'define def_UBANK_BIAS 1`

The following table lists the recommended UltraRAM numbers for different architectures. You can also adjust the numbers according to the resource usage of the entire project.

Table 23: **Recommended UltraRAM Numbers**

	B512	B800	B1024	B1152	B1600	B2304	B3136	B4096
U_BANK_IMG	2	2	4	2	4	4	4	5
U_BANK_WGT	9	11	9	13	11	13	15	17
U_BANK_BIAS	1	1	1	1	1	1	1	1

- **RAM Usage:**

  RAM usage high - `` `define RAM_USAGE_HIGH ``

  RAM usage low - `` `define RAM_USAGE_LOW ``

- **Channel Augmentation:**

  Enable - `` `define CHANNEL_AUGMENTATION_ENABLE ``

  Disable - `` `define CHANNEL_AUGMENTATION_DISABLE ``

- **DepthwiseConv:**

  Enable - `` `define DWCV_ENABLE ``

  Disable - `` `define DWCV_DSIABLE ``

- **AveragePool:**

  Enable - `` `define POOL_AVG_ENABLE ``

  Disable - `` `define POOL_AVG_DISABLE ``

- **Elementwise Multiply:**

  Enable - `` `define ELEW_MULT_ENABLE ``

  Disable - `` `define ELEW_MULT_DISABLE ``

- **RELU Type:** There are two options of RELU type, they are:
  - RELU_RELU6
  - RELU_LEAKRELU_RELU6

  If you want to use the RELU, Leaky ReLU, and ReLU6, define as shown below:

  ```
  `define RELU_LEAKYRELU_RELU6
  ```

- **DSP Usage:**

High - `` `define DSP48_USAGE_HIGH ``

Low - `` `define DSP48_USAGE_LOW ``

- **Low Power Mode:**

  Enable - `` `define LOWPOWER_ENABLE ``

  Disable - `` `define LOWPOWER_DISABLE ``

- **Device Configuration:**

  Support Zynq UltraScale+ MPSoC - `` `define MPSOC ``.

- **Set the DPUCZDX8G Number:** The number of cores is set to one by default. Add the [connectivity] property to configure the DPU number as follows:

```
[connectivity]
```

```
nk=dpu_xrt_top:2
```

  The project will integrate two DPUs.

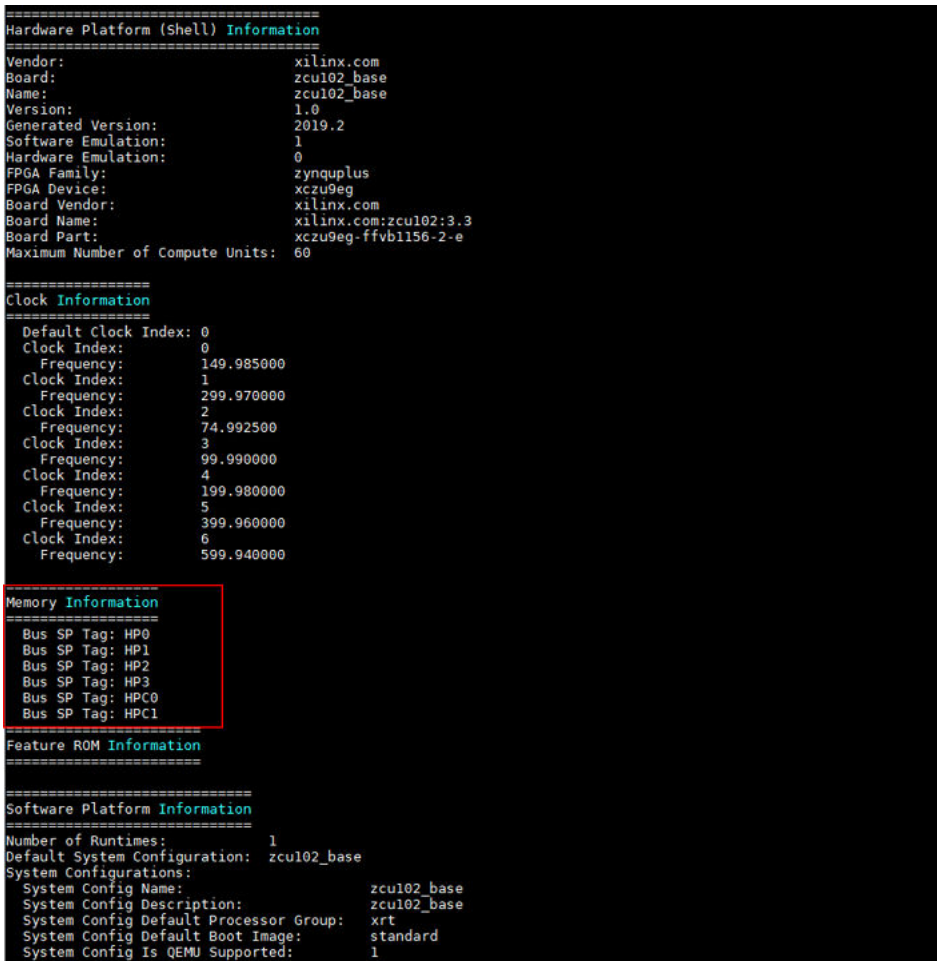
- **Softmax:** The Softmax IP is an independent kernel. The user can choose whether to integrate Softmax IP according to their own applications. Please refer to Vitis DPU TRD Flow for the detail method.

- **Specify Connectivity for the Ports:** Specify the connectivity to the various ports in the system for the DPU. Add the [connectivity] property to configure the DPUCZDX8G ports.

  Use the following command to check the ports of platform:

```
% platforminfo -p zcu102_base/zcu102_base.xpfm
```

Figure 31: **Platform Information**



If the platform does not have enough ports to connect to all the ports of the DPUCZDX8G, then the ports can be shared.

Add the [connectivity] property to specify the DPUCZDX8G ports as follows:

```
[connectivity]
sp=dpu_xrt_top_1.M_AXI_GP0:HP0
sp=dpu_xrt_top_1.M_AXI_HP0:HP1
sp=dpu_xrt_top_1.M_AXI_HP2:HP2
"
```

The project may have timing issues. You can add the `[vivado] property` to configure the Vivado implementation strategy.

```
[vivado]
prop=run.impl_1.strategy=Performance_Explore
```

The Vivado implementation step uses the Performance_Explore strategy.

XILINX

Chapter 6

Example Design

Introduction

The Xilinx® DPUCZDX8G targeted reference design (TRD) provides instructions on how to use the DPUCZDX8G with a Xilinx SoC platform to build and run deep neural network applications. The TRD includes two parts, the Vivado DPU TRD and the Vitis™ DPU TRD. The TRD uses the Vivado IP integrator flow for building the hardware design and the Xilinx Yocto PetaLinux flow for software design. The Zynq® UltraScale+™ MPSoC platform is used to create this TRD. The TRD can be accessed through this link: https://www.xilinx.com/products/intellectual-property/dpu.html#overview.

This chapter describes the architecture of the reference design and provides a functional description of its components. It is organized as follows:

- Vivado TRD Overview provides a high-level overview of the Zynq UltraScale+ MPSoC architecture, the reference design architecture, and a summary of key features.
- Hardware Design gives an overview of how to use the Xilinx Vivado Design Suite to generate the reference hardware design.
- Software Design describes the design flow of project creation in the PetaLinux environment.
- Demo execution describes how to run an application created by the TRD.

The architecture of the Vitis DPU TRD reference design is similar to the Vivado TRD. However, the Vitis DPU TRD is friendly and flexible to the software designer.

Vivado DPU TRD Flow

For the Vivado DPU TRD Flow, refer to the https://github.com/Xilinx/Vitis-AI/blob/master/dsa/DPU-TRD/prj/Vivado/README.md.

This tutorial contains information about:

- Setting up the ZCU102 evaluation board and run the TRD.
- Changing the configuration of DPU.

Vitis DPU TRD Flow

For the Vitis™ DPU TRD Flow, refer to https://github.com/Xilinx/Vitis-AI/blob/master/dsa/DPU-TRD/prj/Vitis/README.md.

This tutorial contains information about:

- Setting up the ZCU102 evaluation board and running the TRD.
- Changing the DPU configuration.
- Integrating the DPU in a custom platform in the Vitis environment.
- Run the DPU_TRD in Vitis GUI flow.

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

***Note*:** For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. *Vitis AI User Guide* (UG1414)
2. *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209)
3. *PetaLinux Tools Documentation: Reference Guide* (UG1144)
4. *ZCU102 Evaluation Board User Guide* (UG1182)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
01/20/2022 Version 3.4	
Entire document	• Updated the interrupt signal of DPU image. • Deleted the device tree paragraph of Vivado flow. • Updated unsupported models table.
07/22/2021 Version 3.3	
Chapter 4: DPU Configuration	• Added Unsupported Models. • Added a figure to illustrate the parallelism of DPU. • Added DPU resource table with UltraRAM enabled. • Support for Zynq-7000 devices has been discontinued. Contact your local sales representative.
02/03/2021 Version 3.3	
Chapter 4: DPU Configuration	Updated Configuring Clock Wizard.
12/17/2020 Version 3.3	
Entire document	• Added Elementwise-Multiply function. • Added Max Reduce function. • Added constrain of Convolution and Deconvolution in Table 7. • Updated figures.
07/07/2020 Version 3.2	
Entire document	• Added Average Pool and BatchNormal in Table 7. • Added resources increment table of Average Pool, LeakyReLU, Depthwise conv, and Softmax . • Deleted the detailed description of Vivado flow and Vitis flow. For more details see the DPU_TRD GitHub.

Section	Revision Summary
03/23/2020 Version 3.2	
Entire document	• Updated the Vivado flow and Vitis flow for Target Version of 1.4.1. • Replaced the description of DNNDK and DNNC to Vitis AI and Vitis AI Compiler. • Updated the maximum of DPU core number from three to four and modified the descriptions accordingly.
12/02/2019 Version 3.1	
Entire document	Updated the flow for Vitis™ device support.
08/13/2019 Version 3.0	
Vitis AI Development Kit	Updated description.
Configuration Options	Added description in RAM Usage, Channel Augmentation, and updated numbers in Softmax section.
Advanced Tab	Added note in DSP Cascade and updated LUT numbers for High DSP in Resources for Different DSP Usage table.
Build the PetaLinux Project	Updated code.
07/31/2019 Version 3.0	
Chapter 2: Overview	Updated whole chapter.
Chapter 3: Product Specification	Updated whole chapter.
Table 1: DPU Signal Description	Added dpu_2x_clk_ce description.
DPU Configuration	Updated whole chapter.
Introduction	Updated description.
Table 7: Deep Neural Network Features and Parameters Supported by DPU	Updated Depthwise Convolution and Max Pooling descriptions.
Configuration Options	• Updated figures. • Added Channel Augmentation and dpu_2x Clock Gating sections and updated all description sections. • Updated Clocking and Reset.
Adding CE for dpu_2x_clk	Added section.
Chapter 5: Development Flow	Updated whole chapter.
Add the DPUCZDX8G into Repository or Upgrade the DPUCZDX8G from a Previous Version	Updated section.
Customizing and Generating the Core in Zynq-7000 Devices	Updated figure.
Chapter 6: Example Design	Updated whole chapter.
DPU Configuration	Updated section.
06/07/2019 Version 2.0	
Vitis AI Development Kit	Added description.
Table 1: DPU Signal Description	Added softmax descriptions.
Interrupts	Updated notes.
Table 7: Deep Neural Network Features and Parameters Supported by DPU	Added Depthwise Convolution.

Section	Revision Summary
Configuration Options	Added some new features: depthwise convolution, average pooling, ReLU type, softmax. Updated some figures of DPU GUI. Added description about s-axi clock mode.
Table 12: Performance of Different Models	Updated table.
Table 13: I/O Bandwidth Requirements for DPU-B1152 and DPU-B4096	Updated table.
Register Clock	Fixed the recommended frequency for DPU clock.
Configuring Clock Wizard	Updated description and figure.
Adding CE for dpu_2x_clk	Updated description and figure.
Configure DPUCZDX8G Parameters	Updated figure.
Connecting the DPUCZDX8G to the Processing System in the Zynq UltraScale+ MPSoC	Updated section.
Assign Register Addresses	Updated note.
Device Tree	Added section.
Customizing and Generating the Core in Zynq-7000 Devices	Added section.
Design Files	Updated figure.
DPU Configuration	Updated figure.
Software Design	Updated section.
03/26/2019 Version 1.2	
Build the PetaLinux Project	Updated description.
Build the Demo	Updated figure.
Demo Execution	Updated code.
03/08/2019 Version 1.1	
Table 6: reg_dpu_base_addr	Updated description.
Figure 10: DPU Configuration	Updated figure.
Build the PetaLinux Project	Updated code.
Build the Demo	Updated description.
03/05/2019 Version 1.1	
Chapter 6: Example Design	Added chapter regarding the DPU targeted reference design.
02/28/2019 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright