DPUCVDX8G for Versal ACAPs

Product Guide

PG389 (v1.0) July 22, 2021

Table of Contents

Introduction

The Xilinx® Versal™ Deep Learning Processing Unit (DPUCVDX8G) is a configurable computation engine optimized for convolution neural networks in Versal ACAP devices with AI Engines. DPUCVDX8G is suitable for embedded applications based on the VCK190 evaluation board. The degree of parallelism used 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 DPUCVDX8G has the following features:

- One AXI4-Lite slave interface for accessing configuration and status registers.
- One AXI4 master interface for accessing instructions.
- Supports batch sizes 1-6.
- Supports a configurable number of AXI4 interfaces for loading/saving the features.
- Supports optional interrupt request generation.

The following are some highlights of DPUCVDX8G functionality:

- Convolution and deconvolution
- Depthwise convolution
- Max pooling
- Average pooling
- ReLU, ReLU6, Leaky ReLU, Hard Sigmoid, and Hard Swish
- Concat
- Elementwise-Sum and Elementwise-Multiply
- Dilation
- Reorg
- Fully connected layer

- Batch Normalization
- Split

IP Facts

DPUCVDX8G IP Facts Table	
Core Specifics	
Supported Device Family	Versal™ AI Core Series
Supported User Interfaces	AXI4-Lite, AXI4-Stream
Resources	See Resource Utilization
Provided with Core	
Design Files	Encrypted RTL
Example Design	Verilog
Test Bench	Not Provided
Constraints File	Xilinx Constraints File
Simulation Model	Not Provided
Supported S/W Driver	Included in PetaLinux
Tested Design Flows[1]	
Design Entry	Vitis™ Unified Software Platform
Simulation	N/A
Synthesis	Vivado® Synthesis
Support	
Xilinx Support web page	

Notes:

1. For the supported versions of third-party tools, see the *Vitis Unified Software Platform Documentation: Application Acceleration Development* (UG1393).

Chapter 2

Overview

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All Versal™ ACAP design process Design Hubs can be found on the Xilinx.com website. This document covers the following design processes:

- **System and Solution Planning:** Identifying the components, performance, I/O, and data transfer requirements at a system level. Includes application mapping for the solution to PS, PL, and AI Engine. Topics in this document that apply to this design process include
  - Chapter 4: Designing with the Core
- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Chapter 3: Product Specification
  - Clocking
- **System Integration and Validation:** Integrating and validating the system functional performance, including timing, resource use, and power closure. Topics in this document that apply to this design process include:
  - Chapter 5: Example Design

Core Overview

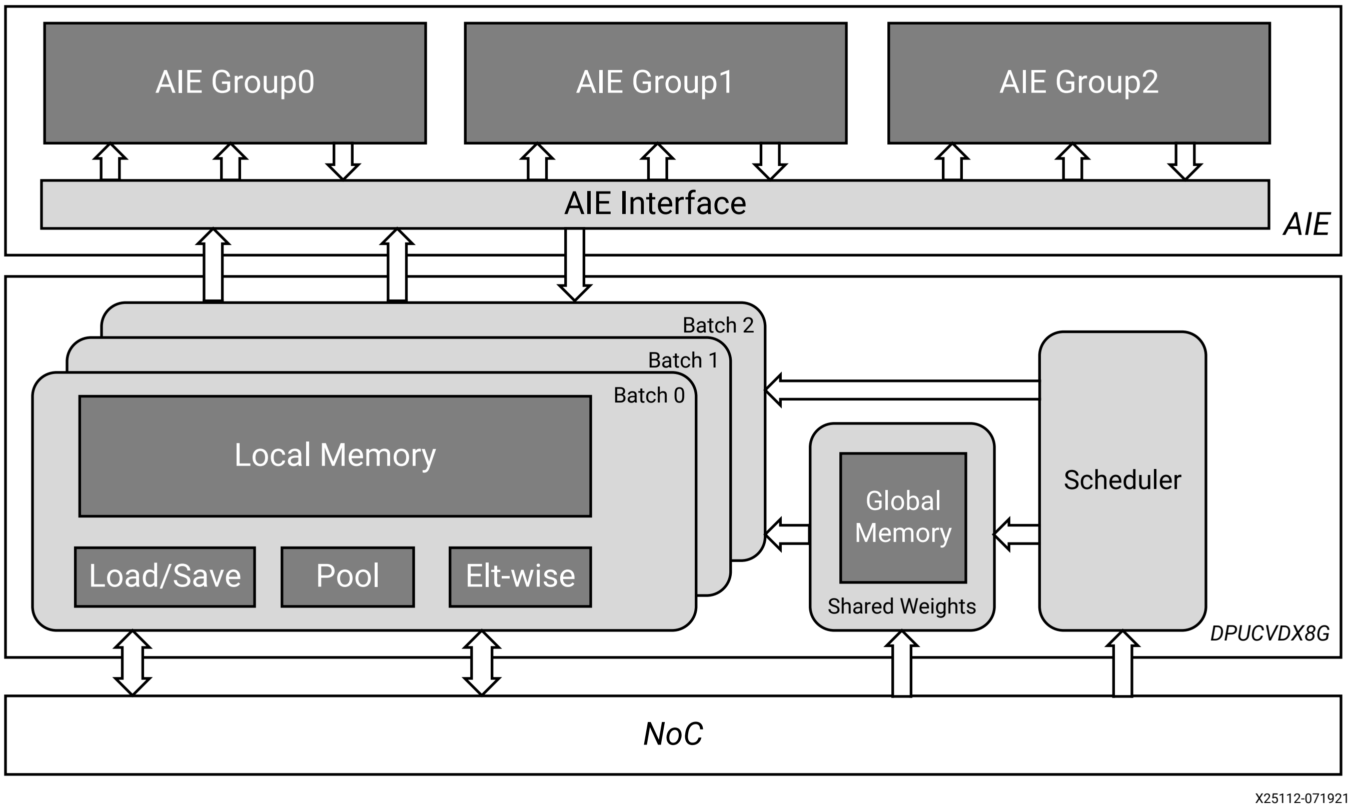
The DPUCVDX8G is a high-performance general CNN processing engine optimized for Versal ACAP devices. This IP is user-configurable and exposes several parameters which can be specified to configure a number of used AI Engine cores and PL resources or customize features. The DPUCVDX8G is composed of both AI Engines and PL.

The AI Engines in the DPUCVDX8G perform convolution. The AI Engine interface tiles transfer data between the AI Engine and the PL. For high-performance calculation in some Versal devices, the AI Engine groups are composed of multiple adjacent AI Engines. For a multi-batch DPUCVDX8G architecture, each batch handler has a private AI Engine group.

The PL component includes a high-level scheduler module, a global memory for shared weights, and batch handlers for Load, Save, Pool, Elt-wise. The scheduler and weights buffer are shared logic between all DPUCVDX8G batch handlers. The Load and Save Module, Pooling and Elt-wise Module, and local feature map storage are private for each batch handlers.

The top-level block diagram of DPUCVDX8G is shown in the following figure.

Figure 1: **DPUCVDX8G Block Diagram**



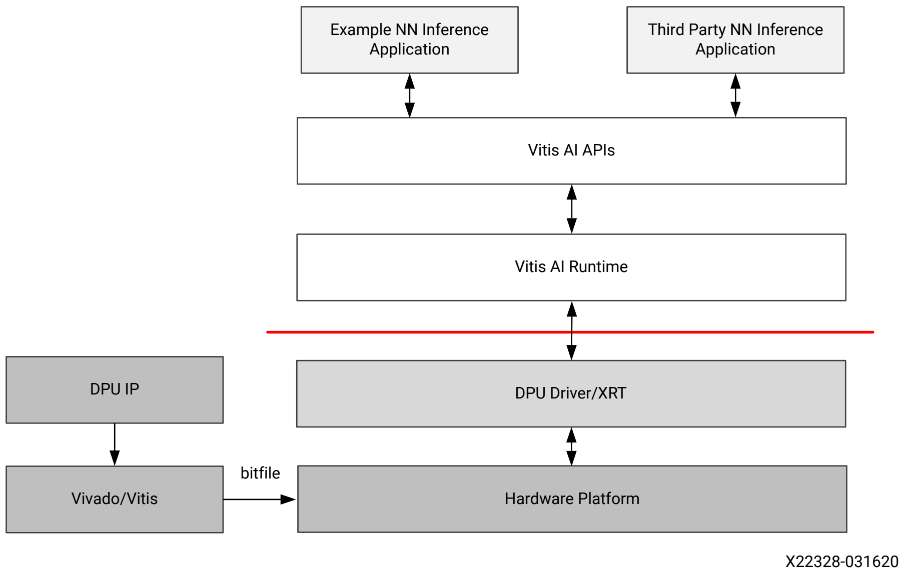
Development Tools

Currently, the DPUCVDX8G only supports the Vitis™ flow to integrate the IP into hardware platforms. The Vitis unified software platform 2020.2 or later is required for the Vitis flow.

DPUCVDX8G Development Flow

The DPUCVDX8G requires a device driver which is included in the Vitis AI development kit. Free developer resources can be obtained from the Vitis AI website. The *Vitis AI User Guide* (UG1414) describes how to use the DPUCVDX8G with the Vitis AI tools. The basic development flow is shown in the following figure. First, use the Vitis IDE 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).

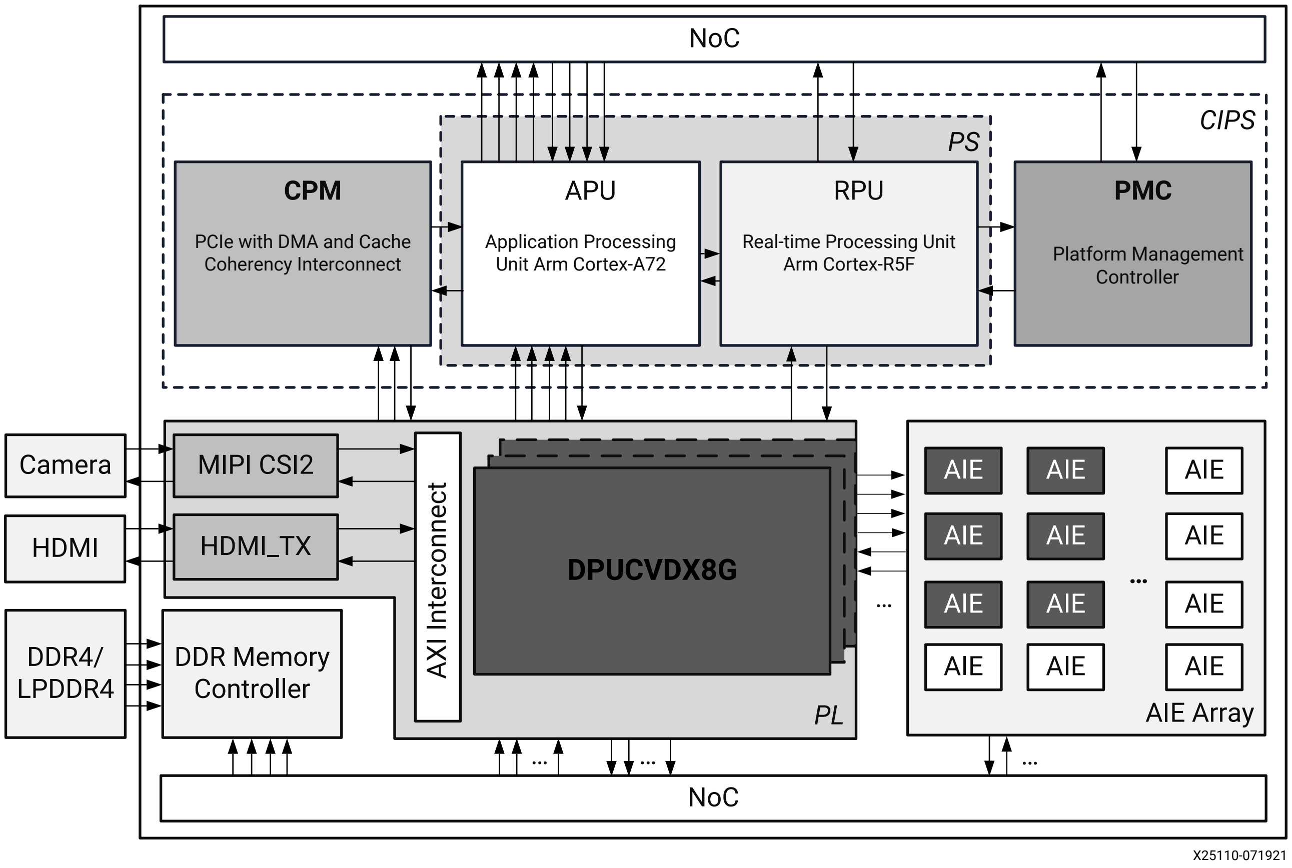
Figure 2: **Hardware/Software Stack**



Example System with DPUCVDX8G

The following figure shows an example system block diagram with the Versal devices using a camera input. The DPUCVDX8G is integrated into the system through NoC and SmartConnect to perform deep learning inference tasks such as image classification, object detection, and semantic segmentation.

Figure 3: **Example System Block Diagram**


NoC
CIPS
PS
CPM
PCIe with DMA and Cache Coherency Interconnect
APU
Application Processing Unit Arm Cortex-A72
RPU
Real-time Processing Unit Arm Cortex-R5F
PMC
Platform Management Controller
Camera
MIPI CSI2
HDMI
HDMI_TX
AXI Interconnect
DPUCVDX8G
DDR4/ LPDDR4
DDR Memory Controller
PL
AIE
AIE Array
NoC
X25110-071921

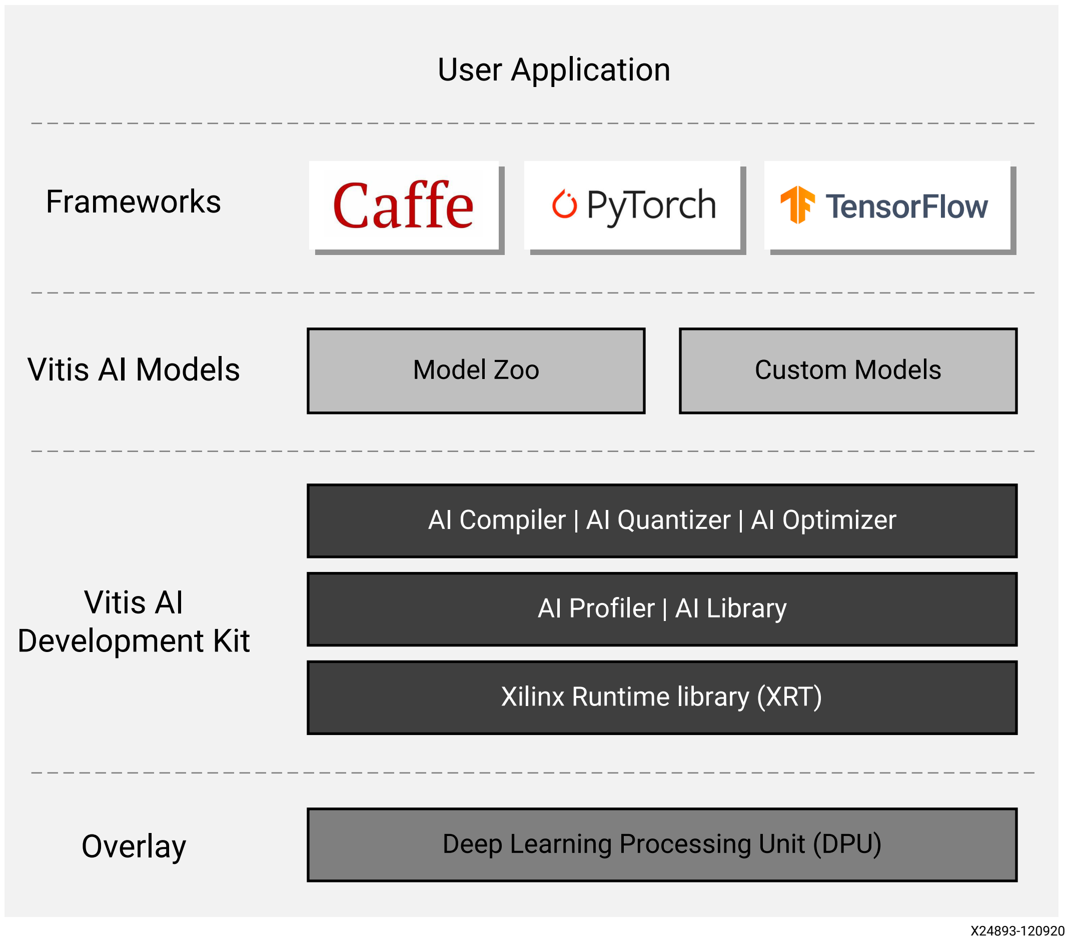
For information on Versal ACAP architecture, see *Versal ACAP Technical Reference Manual* (AM011).

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 4: **Vitis AI Stack**



For more information of the Vitis AI development kit, see the *Vitis AI User Guide* in the *Vitis AI User Documentation* (UG1431).

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

Product Specification

Resource Utilization

The resource utilization of several DPUCVDX8G architectures is shown in the following table. The C32 means that the CPB_N (number of AI Engine cores per batch handler) equals to 32, B3 means that there are three batch handlers, and L2S2 means that the LOAD_PARALLEL_IMG=2 and the SAVE_PARALLEL_IMG=2.

Table 1: **Referenced Resources Utilization of Different DPUCVDX8G Architecture**

Architecture	AIE Cores	LUT	FF	Block RAM	UltraRAM	DSP	PL NMU
C32B1L2S2	32	87858	116386	0	204	263	8
C32B2L2S2	64	157369	200910	0	268	521	10
C32B3L2S2	96	227420	285434	0	332	779	12
C32B4L2S2	128	312561	372098	0	396	1037	14
C32B5L2S2	160	387691	459083	0	460	1295	16
C32B6L2S2	192	441004	540421	644	411	1169	18

Performance (Theoretical)

The following table shows the peak theoretical performance of the DPUCVDX8G with C32B1L2S2 architecture.

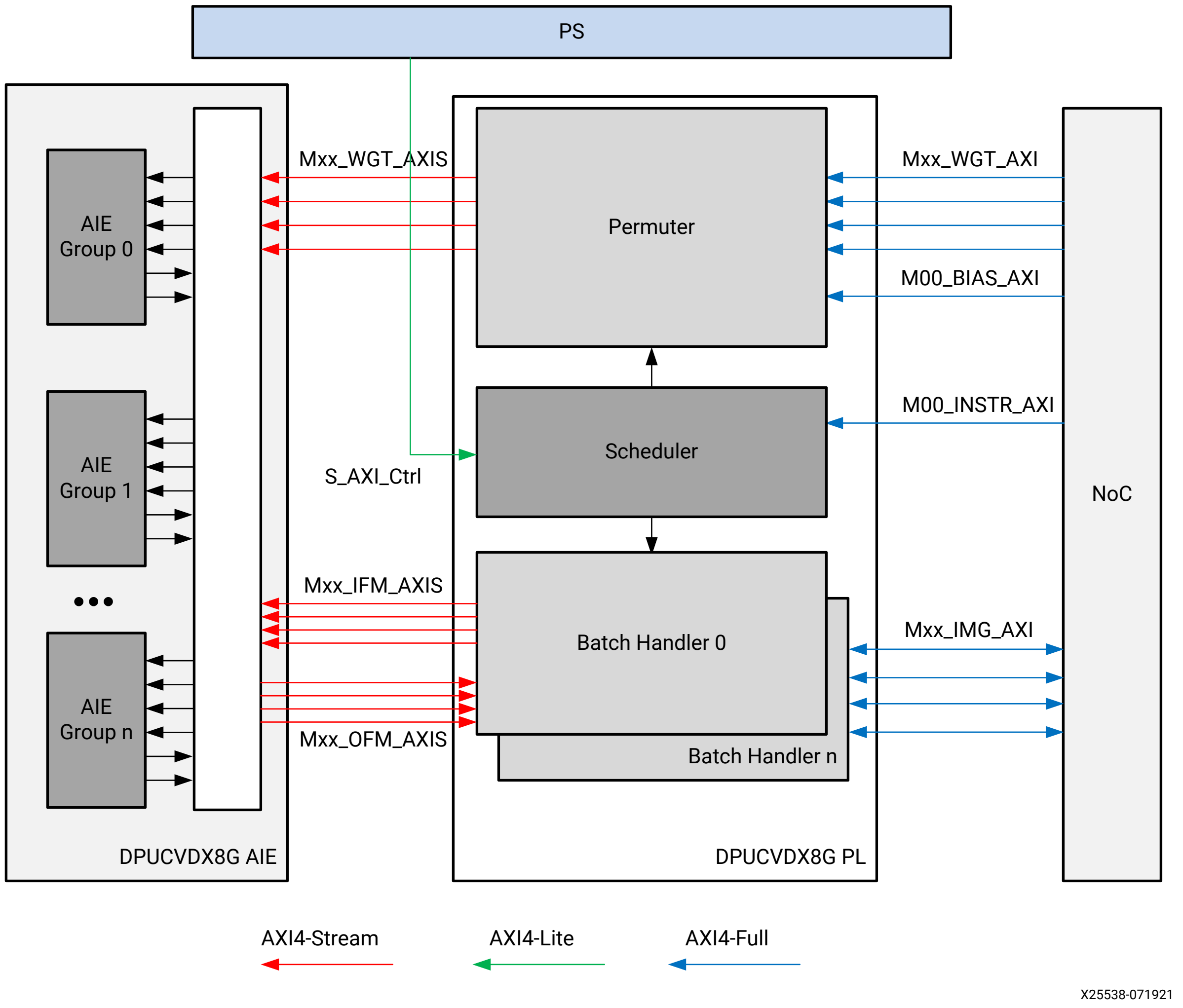
Table 2: **Peak Theoretical Performance of the DPUCVDX8G**

Architecture	PL Frequency (MHz)	AIE Frequency (GHz)	Peak Theoretical Performance (TOPS)
C32B1L2S2	333	1.333	10.9

DPUCVDX8G Port Description

Ports are available from the DPUCVDX8G to the NoC and the PS. The interface from the PS to the DPUCVDX8G PL component is used for register configuration. The interfaces from the DPUCVDX8G PL component to the NoC are for image and weight transfers. The interfaces from the PL component to the AI Engine component are for intermediate data exchange. The DPUCVDX8G top-level interfaces are shown in the following figure.

Figure 5: **Connections from DPUCVDX8G to PS and NoC**



There are parameters to configure the DPUCVDX8G architecture. The data width and number of different DPUCVDX8G interfaces differs for different architecture. A screen capture of DPUCVDX8G IP catalog with C32B1 (CPB_N=32, Batch_N=1) architecture is shown as follows:

Figure 6: **DPUCVDX8G Ports with C32B1 Arch**

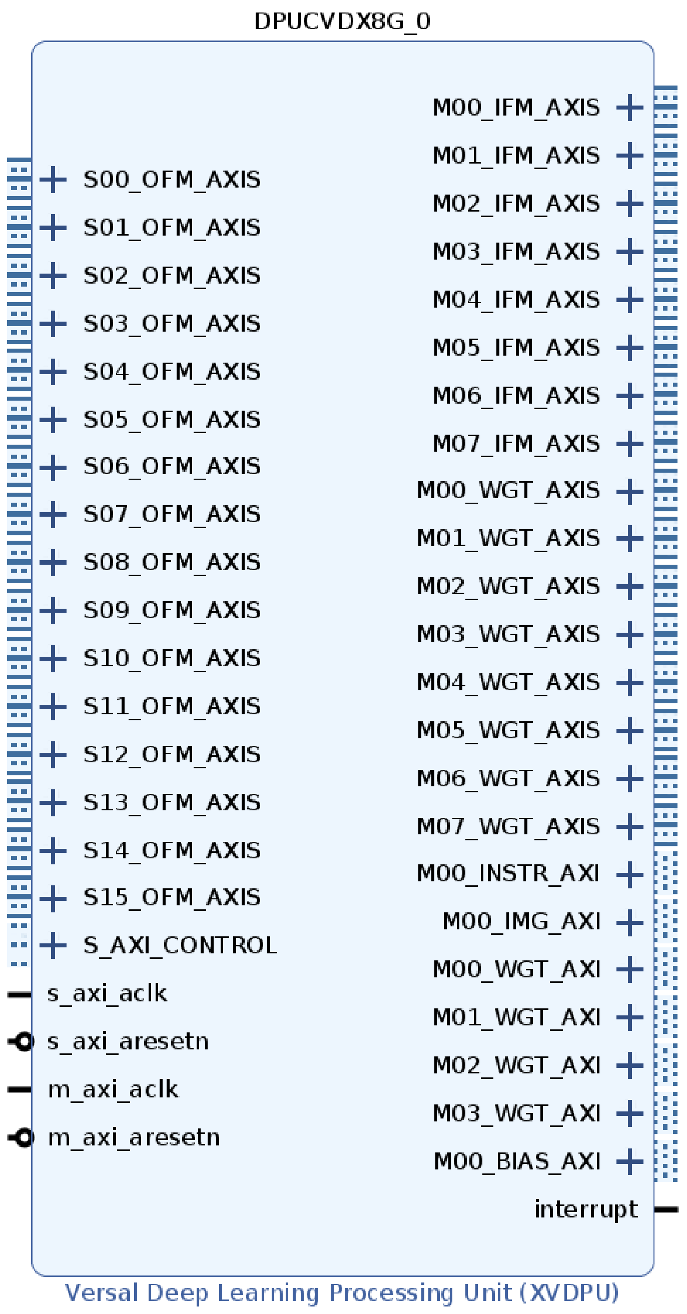


Table 3: **DPUCVDX8G Port Description**

Port Name	Interface Type	Data Width	I/O	Description
m_axi_aclk	Clock	1	I	Input clock used for DPUCVDX8G general logic.

Table 3: **DPUCVDX8G Port Description** *(cont'd)*

Port Name	Interface Type	Data Width	I/O	Description
m_axi_aresetn	Reset	1	I	Active-Low reset for DPUCVDX8G general logic.
s_axi_aclk	Clock	1	I	AXI clock input for S_AXI_CONTROL.
s_axi_aresetn	Reset	1	I	Active-Low reset for S_AXI_CONTROL.
interrupt	Interrupt	1	O	Active-High interrupt output from the DPUCVDX8G.
S_AXI_CONTROL	AXI4-Lite	32	I/O	32-bit AXI4-Lite interface for the DPUCVDX8G registers.
Sxx_OFM_AXIS	AXI4-Stream	64	I	Output feature map from the AI Engine side to the PL side. The port number depends on the DPUCVDX8G architecture and the batch number.
Mxx_IFM_AXIS	AXI4-Stream	64 or 128	O	Input feature map from the PL side to the AI Engine side. The data width depends on the DPUCVDX8G architecture. The port number depends on the DPUCVDX8G architecture and batch number.
Mxx_WGT_AXIS	AXI4-Stream	128	O	Weights data from the PL side to the AI Engine side. The port number depends on the DPUCVDX8G architecture and batch number.
M00_INSTR_AXI	AXI4	32	I/O	32-bit memory mapped AXI interface for DPU instructions.
M00_BIAS_AXI	AXI4	128	I/O	128-bit memory mapped AXI interface for loading bias data.
Mxx_IMG_AXI	AXI4	128	I/O	128-bit memory mapped AXI interface for loading image and uploading output. The port number depends on the DPUCVDX8G architecture and batch number.
Mxx_WGT_AXI	AXI4	512	I/O	512-bit memory mapped AXI interface for loading shared weights. The port number is fixed at 4.

DPUCVDX8G Registers

The DPUCVDX8G implements registers in programmable logic. These registers are accessible from the APU through the `S_AXI_CONTROL` interface. The following tables show the DPUCVDX8G registers.

AP Registers

The AP registers are general registers for the Vitis flow. The description of the AP registers are shown in the following table.

Table 4: **AP Registers**

Name	Offset Address	Bits	Filed Name	Type	Description
Control	0x00	[31:7]	Reserved		Reserved
		[6]	AP_RESET_DONE	r	The completion flag of soft reset. Active-High.
		[5]	AP_RESET	r/w	The soft reset for DPUCVDX8G. Active-High.
		[4]	Reserved		Reserved
		[3]	AP_READY	r	Indicates when the DPUCVDX8G is ready for new input data. Cleared on AP_DONE asserted.
		[2]	AP_IDLE	r	Asserted when the DPUCVDX8G is idle.
		[1]	AP_DONE	r	Asserted when the DPUCVDX8G has completed operation. Cleared on read.
		[0]	AP_START	r/w	Asserted when kernel can start processing data. Cleared on handshake with `ap_done` being asserted.
Global Interrupt Enable	0x04	[31:1]	Reserved		Reserved
		[0]	Global Interrupt Enable	r/w	When asserted, along with the DPUCVDX8G Interrupt Enable bit, the interrupt is enabled.
Interrupt Enable Register	0x08	[31:1]	Reserved		Reserved
		[1]	IER_AP_REAYD	r/w	The ap_ready of IER.
		[0]	IER_AO_DONE	r/w	The ap_done of IER.
Interrupt Status Register	0x0C	[31:1]	Reserved		Reserved
		[1]	ISR_AP_READY	r/w	The ap_ready of ISR.
		[0]	ISR_AP_DONE	r/w	The ap_done of ISR.

- **Control (0x0000) Register:** This register controls the operation of the core.
  - Bit [0] of the Control register, `ap_start`, kicks off the IP from software. Writing 1 to this bit, starts the IP to working.

  - Bit [1] of the Control register, `ap_done`, indicates when the IP has completed all operations in the current transaction. A logic 1 on this signal indicates that the IP has completed all operations in this transaction.
  - Bit [2] of the Control register, `ap_idle`, signal indicates if the IP is operating or idle (no operation). The idle state is indicated by logic 1. This signal is asserted Low once the IP starts operating. This signal is asserted High when the IP completes operation and no further operations are performed.
  - Bit [3] of the Control register, `ap_ready`, signal indicates when the IP is ready for new inputs. It is set to logic 1 when the IP is ready to accept new inputs, indicating that all input reads for this transaction are completed. If the IP has no operations in the pipeline, new reads are not performed until the next transaction starts. This signal is used to make a decision on when to apply new values to the input ports and whether to start a new transaction.
  - Bit [5] of the Control register is the soft reset for the IP. When this is set, the IP is reset by the software.
  - Bit [6] of the Control register is the finished status of soft reset. This is asserted when the soft reset is done.

- **Global Interrupt Enable (0x0004) Register:** This register is the master control for all interrupts. Bit [0] can be used to enable/disable interrupts.
- **Interrupt Enable (0x0008) Register:** This register allows interrupts to be enabled selectively. Currently, two interrupt sources are available, `ap_done` and `ap_ready`. `ap_done` is triggered after the IP processing is complete, while `ap_ready` is triggered after the IP is ready to start processing the next task.
- **Interrupt Status (0x000C) Register:** This is a dual purpose register. When an interrupt occurs, the corresponding interrupt source bit is set in this register. In readback mode (Get status), the interrupting source can be determined. In writeback mode (Clear interrupt), the requested interrupt source bit is cleared.

DPUCVDX8G Configurable Registers

The DPUCVDX8G configurable registers are used for configuring Xilinx IPs.

The `reg_dpu_instr_addr` register is used to indicate the instruction address of the DPUCVDX8G. The instruction address is a 44-bit signal consist of the 32-bit of INSTR_ADDR_L and lower 12-bit of INSTR_ADDR_H. Actually the lower 12-bit of `reg_dpu_instr_addr` are set to be zero in the DPUCVDX8G logic. Hence, the available instruction address for the DPUCVDX8G ranges from 0x1000 to 0xFFF_FFFF_F000.

The `reg_dpu_base_addr` register is used to indicate the address of input image and parameters in external memory. The width of the DPUCVDX8G base address is 44 bits, so it can support address ranges from 0 to 16 TB. All registers are 32 bits wide, so two registers are required to compose the 44-bit wide base address. Reg `BATCH0_ADDR0_L` represents the lower 32 bits of base address0 of the DPUCVDX8G batch handler0 and `BATCH0_ADDR0_H` represents the upper 12 bits of base address0. For each DPUCVDX8G batch handler, there are eight base addresses. The DPUCVDX8G supports up to six batch handlers, so there are six groups of batch base addresses.

The description of those registers are shown in the following table.

Table 5: **DPUCVDX8G Configurable Registers**

Name	Offset Address	Bits	Type	Description
IRQ_CLR	0x40	[31:1]		Reserved
		[0]	r/w	When asserted, the DPUCVDX8G interrupt is cleared. The IRQ_CLR will be cleared after the interrupt is cleared.
INSTR_ADDR_L	0x50	[31:0]	r/w	The lower 32-bit of start address for fetching instructions from an external memory.
INSTR_ADDR_L	0x54	[31:12]		Reserved
		[11:0]	r/w	The higher 12-bit of start address for fetching instructions from an external memory.
BATCH0_ADDR0_L	0x200	[31:0]	r/w	The lower 32-bit base address0 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR0_H	0x204	[11:0]	r/w	The higher 12-bit base address0 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR1_L	0x208	[31:0]	r/w	The lower 32-bit base address1 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR1_H	0x20c	[11:0]	r/w	The higher 12-bit base address1 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR2_L	0x210	[31:0]	r/w	The lower 32-bit base address2 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR2_H	0x214	[11:0]	r/w	The higher 12-bit base address2 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR3_L	0x218	[31:0]	r/w	The lower 32-bit base address3 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR3_H	0x21c	[11:0]	r/w	The higher 12-bit base address3 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR4_L	0x220	[31:0]	r/w	The lower 32-bit base address4 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR4_H	0x224	[11:0]	r/w	The higher 12-bit base address4 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR5_L	0x228	[31:0]	r/w	The lower 32-bit base address5 of batch handler0 for loading and saving image and weights.

Table 5: **DPUCVDX8G Configurable Registers** *(cont'd)*

Name	Offset Address	Bits	Type	Description
BATCH0_ADDR5_H	0x22c	[11:0]	r/w	The higher 12-bit base address5 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR6_L	0x230	[31:0]	r/w	The lower 32-bit base address6 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR6_H	0x234	[11:0]	r/w	The higher 12-bit base address6 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR7_L	0x238	[31:0]	r/w	The lower 32-bit base address7 of batch handler0 for loading and saving image and weights.
BATCH0_ADDR7_H	0x23c	[11:0]	r/w	The higher 12-bit base address7 of batch handler0 for loading and saving image and weights.
BATCH1_ADDR0_L	0x240	[31:0]	r/w	The lower 32-bit base address0 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR0_H	0x244	[11:0]	r/w	The higher 12-bit base address0 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR1_L	0x248	[31:0]	r/w	The lower 32-bit base address1 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR1_H	0x24c	[11:0]	r/w	The higher 12-bit base address1 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR2_L	0x240	[31:0]	r/w	The lower 32-bit base address2 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR2_H	0x244	[11:0]	r/w	The higher 12-bit base address2 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR3_L	0x248	[31:0]	r/w	The lower 32-bit base address3 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR3_H	0x24c	[11:0]	r/w	The higher 12-bit base address3 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR4_L	0x250	[31:0]	r/w	The lower 32-bit base address4 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR4_H	0x254	[11:0]	r/w	The higher 12-bit base address4 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR5_L	0x268	[31:0]	r/w	The lower 32-bit base address5 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR5_H	0x26c	[11:0]	r/w	The higher 12-bit base address5 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR6_L	0x270	[31:0]	r/w	The lower 32-bit base address6 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR6_H	0x274	[11:0]	r/w	The higher 12-bit base address6 of batch handler1 for loading and saving image and weights.
BATCH1_ADDR7_L	0x278	[31:0]	r/w	The lower 32-bit base address7 of batch handler1 for loading and saving image and weights.

Table 5: **DPUCVDX8G Configurable Registers** *(cont'd)*

Name	Offset Address	Bits	Type	Description
BATCH1_ADDR7_H	0x27c	[11:0]	r/w	The higher 12-bit base address7 of batch handler1 for loading and saving image and weights.
BATCH2_ADDR0_L	0x280	[31:0]	r/w	The lower 32-bit base address0 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR0_H	0x284	[11:0]	r/w	The higher 12-bit base address0 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR1_L	0x288	[31:0]	r/w	The lower 32-bit base address1 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR1_H	0x28c	[11:0]	r/w	The higher 12-bit base address1 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR2_L	0x290	[31:0]	r/w	The lower 32-bit base address2 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR2_H	0x294	[11:0]	r/w	The higher 12-bit base address2 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR3_L	0x298	[31:0]	r/w	The lower 32-bit base address3 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR3_H	0x29c	[11:0]	r/w	The higher 12-bit base address3 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR4_L	0x2a0	[31:0]	r/w	The lower 32-bit base address4 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR4_H	0x2a4	[11:0]	r/w	The higher 12-bit base address4 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR5_L	0x2a8	[31:0]	r/w	The lower 32-bit base address5 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR5_H	0x2ac	[11:0]	r/w	The higher 12-bit base address5 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR6_L	0x2b0	[31:0]	r/w	The lower 32-bit base address6 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR6_H	0x2b4	[11:0]	r/w	The higher 12-bit base address6 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR7_L	0x2b8	[31:0]	r/w	The lower 32-bit base address7 of batch handler2 for loading and saving image and weights.
BATCH2_ADDR7_H	0x2bc	[11:0]	r/w	The higher 12-bit base address7 of batch handler2 for loading and saving image and weights.
BATCH3_ADDR0_L	0x2c0	[31:0]	r/w	The lower 32-bit base address0 of batch handler0 for loading and saving image and weights.
BATCH3_ADDR0_H	0x2c4	[11:0]	r/w	The higher 12-bit base address0 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR1_L	0x2c8	[31:0]	r/w	The lower 32-bit base address1 of batch handler3 for loading and saving image and weights.

Table 5: **DPUCVDX8G Configurable Registers** *(cont'd)*

Name	Offset Address	Bits	Type	Description
BATCH3_ADDR1_H	0x2cc	[11:0]	r/w	The higher 12-bit base address1 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR2_L	0x2d0	[31:0]	r/w	The lower 32-bit base address2 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR2_H	0x2d4	[11:0]	r/w	The higher 12-bit base address2 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR3_L	0x2d8	[31:0]	r/w	The lower 32-bit base address3 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR3_H	0x2dc	[11:0]	r/w	The higher 12-bit base address3 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR4_L	0x2e0	[31:0]	r/w	The lower 32-bit base address4 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR4_H	0x2e4	[11:0]	r/w	The higher 12-bit base address4 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR5_L	0x2e8	[31:0]	r/w	The lower 32-bit base address5 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR5_H	0x2ec	[11:0]	r/w	The higher 12-bit base address5 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR6_L	0x2f0	[31:0]	r/w	The lower 32-bit base address6 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR6_H	0x2f4	[11:0]	r/w	The higher 12-bit base address6 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR7_L	0x2f8	[31:0]	r/w	The lower 32-bit base address7 of batch handler3 for loading and saving image and weights.
BATCH3_ADDR7_H	0x2fc	[11:0]	r/w	The higher 12-bit base address7 of batch handler3 for loading and saving image and weights.
BATCH4_ADDR0_L	0x300	[31:0]	r/w	The lower 32-bit base address0 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR0_H	0x304	[11:0]	r/w	The higher 12-bit base address0 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR1_L	0x308	[31:0]	r/w	The lower 32-bit base address1 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR1_H	0x30c	[11:0]	r/w	The higher 12-bit base address1 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR2_L	0x310	[31:0]	r/w	The lower 32-bit base address2 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR2_H	0x314	[11:0]	r/w	The higher 12-bit base address2 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR3_L	0x318	[31:0]	r/w	The lower 32-bit base address3 of batch handler4 for loading and saving image and weights.

Table 5: **DPUCVDX8G Configurable Registers** *(cont'd)*

Name	Offset Address	Bits	Type	Description
BATCH4_ADDR3_H	0x31c	[11:0]	r/w	The higher 12-bit base address3 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR4_L	0x320	[31:0]	r/w	The lower 32-bit base address4 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR4_H	0x324	[11:0]	r/w	The higher 12-bit base address4 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR5_L	0x328	[31:0]	r/w	The lower 32-bit base address5 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR5_H	0x32c	[11:0]	r/w	The higher 12-bit base address5 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR6_L	0x330	[31:0]	r/w	The lower 32-bit base address6 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR6_H	0x334	[11:0]	r/w	The higher 12-bit base address6 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR7_L	0x338	[31:0]	r/w	The lower 32-bit base address7 of batch handler4 for loading and saving image and weights.
BATCH4_ADDR7_H	0x33c	[11:0]	r/w	The higher 12-bit base address7 of batch handler4 for loading and saving image and weights.
BATCH5_ADDR0_L	0x340	[31:0]	r/w	The lower 32-bit base address0 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR0_H	0x344	[11:0]	r/w	The higher 12-bit base address0 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR1_L	0x348	[31:0]	r/w	The lower 32-bit base address1 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR1_H	0x34c	[11:0]	r/w	The higher 12-bit base address1 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR2_L	0x350	[31:0]	r/w	The lower 32-bit base address2 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR2_H	0x354	[11:0]	r/w	The higher 12-bit base address2 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR3_L	0x358	[31:0]	r/w	The lower 32-bit base address3 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR3_H	0x35c	[11:0]	r/w	The higher 12-bit base address3 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR4_L	0x360	[31:0]	r/w	The lower 32-bit base address4 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR4_H	0x364	[11:0]	r/w	The higher 12-bit base address4 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR5_L	0x368	[31:0]	r/w	The lower 32-bit base address5 of batch handler5 for loading and saving image and weights.

Table 5: **DPUCVDX8G Configurable Registers** *(cont'd)*

Name	Offset Address	Bits	Type	Description
BATCH5_ADDR5_H	0x36c	[11:0]	r/w	The higher 12-bit base address5 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR6_L	0x370	[31:0]	r/w	The lower 32-bit base address6 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR6_H	0x374	[11:0]	r/w	The higher 12-bit base address6 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR7_L	0x378	[31:0]	r/w	The lower 32-bit base address7 of batch handler5 for loading and saving image and weights.
BATCH5_ADDR7_H	0x37c	[11:0]	r/w	The higher 12-bit base address7 of batch handler5 for loading and saving image and weights.

Chapter 4

Designing with the Core

This section includes guidelines and additional information to facilitate designing with the core.

Hardware Architecture

The DPUCVDX8G is composed of the PL and the AI Engine. The AI Engine is used for the convolution operation in neural networks. Data moving, instruction scheduler, pooling, element-wise sum, and depth-wise convolution are executed in the PL.

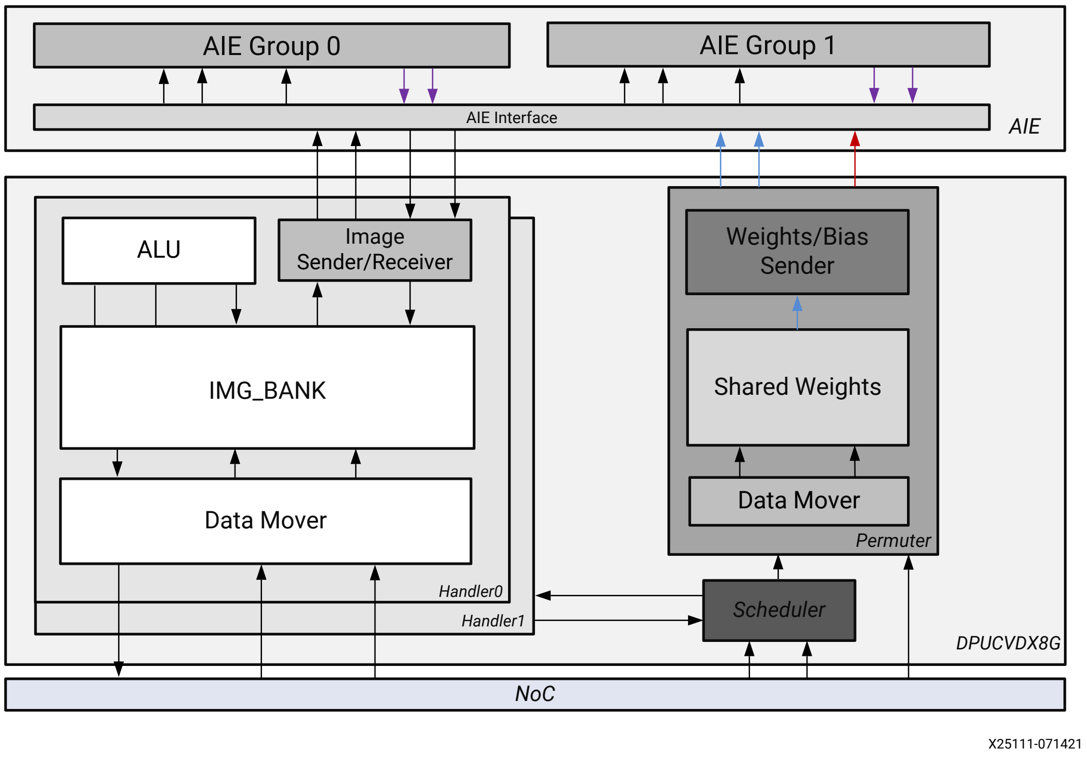
The DPUCVDX8G can be set up with multiple batch handlers. For each batch handler, there is a corresponding AI Engine array and related AI Engine interface tile resources. On the PL side, the DPUCVDX8G is split into two parts: batch handler and shared logic. The batch handler is mainly for the processing of the feature map, such as loading, saving, pooling, etc. The ALU module in the batch handler can process the pooling, element-wise, and depth-wise convolution operations for the feature maps. Feature maps are stored in the IMG BANK which is composed of the on-chip ram. The image sender and weights sender modules are used for preparing the data for the AI Engine array. The shared logic in the PL component includes the Permuter module and the Scheduler module. The scheduler fetches and dispatches instructions from the DPUCVDX8G and transfers them to the batch handler and the Permuter module. The Permuter module loads the weights and bias from the NoC and sends the specific weights data to the AI Engine array for each calculation iteration.

After starting up, the DPUCVDX8G fetches instructions from the NoC 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 to reduce the external memory bandwidth. A deeply pipe-lined design is used for the computing engine.

The detailed hardware architecture of the DPUCVDX8G is shown in the following figure.

Figure 7: **Hardware Architecture of the DPUCVDX8G**



DPUCVDX8G Feature Support

The DPUCVDX8G provides user-configurable parameters to optimize resource usage and customize features. Different configurations can be selected for DSP slices, LUT, block RAM, and Ultra RAM usage based on the amount of available programmable logic resources. There are also options for additional functions, such as channel augmentation, average pooling, and depthwise convolution. Furthermore, there is an option to configure the number of batch handlers of the DPUCVDX8G that is instantiated in a single DPUCVDX8G IP. The deep neural network features and the associated parameters supported by the DPUCVDX8G are shown in the following table.

A configuration file named `arch.json` is generated while integrating the DPUCVDX8G in using the Vitis™ accelerated flow. The `arch.json` file is used by the Vitis AI Compiler for model compilation. For more information on the Vitis AI Compiler, refer to the *Vitis AI User Guide* (UG1414). In the Vitis accelerated flow, the `arch.json` file is located at `$TRD_HOME/vitis_prj/package_out/sd_card/arch.json`.

Table 6: **Deep Neural Network Features and Parameters Supported by the DPUCVDX8G**

Features	Description	Range
Convolution	Kernel Sizes	kernel_w: 1~16 kernel_h: 1~16 kernel_h * kernel_w <= 64
	Strides	stride_w: 1~8 stride_h : 1~8
	Padding_w	0~(kernel_w-1)
	Padding_h	0~(kernel_h-1)
	Input Size	Arbitrary
	Input Channel	1~256 * channel_parallel
	Output Channel	1~256 * channel_parallel
	Activation	ReLU, ReLU6, LeakyReLU, PReLU, Hard Sigmoid and Hard Swish
	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/2
Depthwise Convolution	Kernel Sizes	kernel_w: 1~256 kernel_h: 1~256
	Strides	stride_w: 1~8 stride_h: 1~8
	Padding	pad_w: 0~255 pad_h: 0~255
	Input Size	Arbitrary
	Input Channel	1~256 * channel_parallel
	Output Channel	1~256 * channel_parallel
	Activation	ReLU, ReLU6, LeakyReLU, PReLU, Hard Sigmoid, and Hard Swish
	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/2
Deconvolution	Kernel Sizes	kernel_w: 1~256 kernel_h: 1~256
	Stride_w	(stride_w * output_channel) ≤ (256 * channel_parallel)
	Stride_h	Arbitrary
	Padding_w	0~(kernel_w-1)
	Padding_h	0~(kernel_h-1)
	Input Size	Arbitrary
	Input Channel	1~256 * channel_parallel
	Output Channel	1~256 * channel_parallel
	Activation	ReLU, ReLU6, LeakyReLU, PReLU, Hard Sigmoid, and Hard Swish

Table 6: **Deep Neural Network Features and Parameters Supported by the DPUCVDX8G** *(cont'd)*

Features	Description	Range
Max Pooling	Kernel Sizes	kernel_w: 1~256 kernel_h: 1~256
	Strides	stride_w: 1~8 stride_h: 1~8
	Padding	pad_w: 0~255 pad_h: 0~255
Average Pooling	Kernel Sizes	kernel_w: 1~256 kernel_h: 1~256
	Strides	stride_w: 1~8 stride_h: 1~8
	Padding	pad_w: 0~255 pad_h: 0~255
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
Batch Normalization	-	-
Fully Connected (FC)	Input_channel	Input_channel ≤ 2048 * channel_parallel
	Output_channel	Arbitrary

Notes:

1. The parameter, channel_parallel, is determined by the DPUCVDX8G configuration.
2. In some neural networks, the FC layer is connected with a Flatten layer. The Vitis AI compiler automatically combines 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 is generated during compilation.
   This limitation occurs only in the Flatten+FC situation.
3. The bank_depth is the on-chip weight buffer depth. In the DPUCVDX8G, the bank_depth is 16384.

Configuration Options

The DPU can be configured with some predefined options, which includes the DPUCVDX8G architecture, the batch number, and UltraRAM usage. These options allow you to set the DSP slice, LUT, block RAM, and UltraRAM usage.

BATCH_N

The BATCH_N parameter determines the number of batch handlers integrated in the DPUCVDX8G IP. This parameter supports a range of values from 1 to 6. A higher batch handler number denotes a better performance as well as more AI Engine cores and PL resources. You can balance the performance and resources according to your applications.

UBANK_IMG_N

There are two kinds of on-chip memory resources in Versal devices: block RAM and UltraRAM. Each block RAM has a capacity of 36 Kb and each UltraRAM has a capacity of 288 Kb. The number of available RAMs is device-dependent.

There are 16 IMG BANKs (128 KB per bank) in each DPUCVDX8G batch handler. Each IMG BANK can be composed of block RAM or UltraRAM. The parameter UBANK_IMG_N determines how many IMG BANKs are composed of UltraRAM. The remaining banks will be composed of block RAM. This parameter is designed to flexibly use the on-chip memory resources.

UBANK_WGT_N

There are 17 WGT BANKs (256 KB per bank) in the DPUCVDX8G irrespective of the number of batch handler. Each WGT BANK can be composed of block RAM or UltraRAM. The parameter UBANK_WGT_N determines how many WGT BANKs are composed of Ultra RAM. The remaining banks will be composed of block RAM. This parameter is designed to flexibly use the on-chip memory resources.

LOAD_PARALLEL_IMG

The LOAD_PARALLEL_IMG indicates the level of parallelism of loading images for each DPUCVDX8G batch handler. Each parallelism uses one AXI4 interface for data transmission. Hence, the number of M_IMG_AXI ports of the DPUCVDX8G depends on the LOAD_PARALLEL_IMG. In this release, the supported value for this parameter is limited to two. A higher parallelism means a larger throughput for loading an image and a larger bandwidth requirement, and therefore a higher PL resource usage.

SAVE_PARALLEL_IMG

The SAVE_PARALLEL_IMG indicates level of parallelism of saving images for each DPUCVDX8G batch handler. Each instance uses one AXI4 interface for data transmission. The save module uses the write channel of the AXI4 interface and the load module uses the read channel of the AXI4 interface.

In this release, the supported value for this parameter is limited to two. A higher parallelism means a larger throughput for loading an image and a larger bandwidth requirement, and therefore a higher PL resource usage.

Note: The SAVE_PARALLEL_IMG cannot be set larger than the LOAD_PARALLEL_IMG. Currently, the SAVE_PARALLEL_IMG value is limited to two.

Clocking

There are three clock domains in the DPUCVDX8G IP:

- `s_axi_aclk` for register configuration.
- `m_axi_aclk` for general logic control in the DPUCVDX8G PL component.
- `s00_aie_aclk` for the AI Engine component.

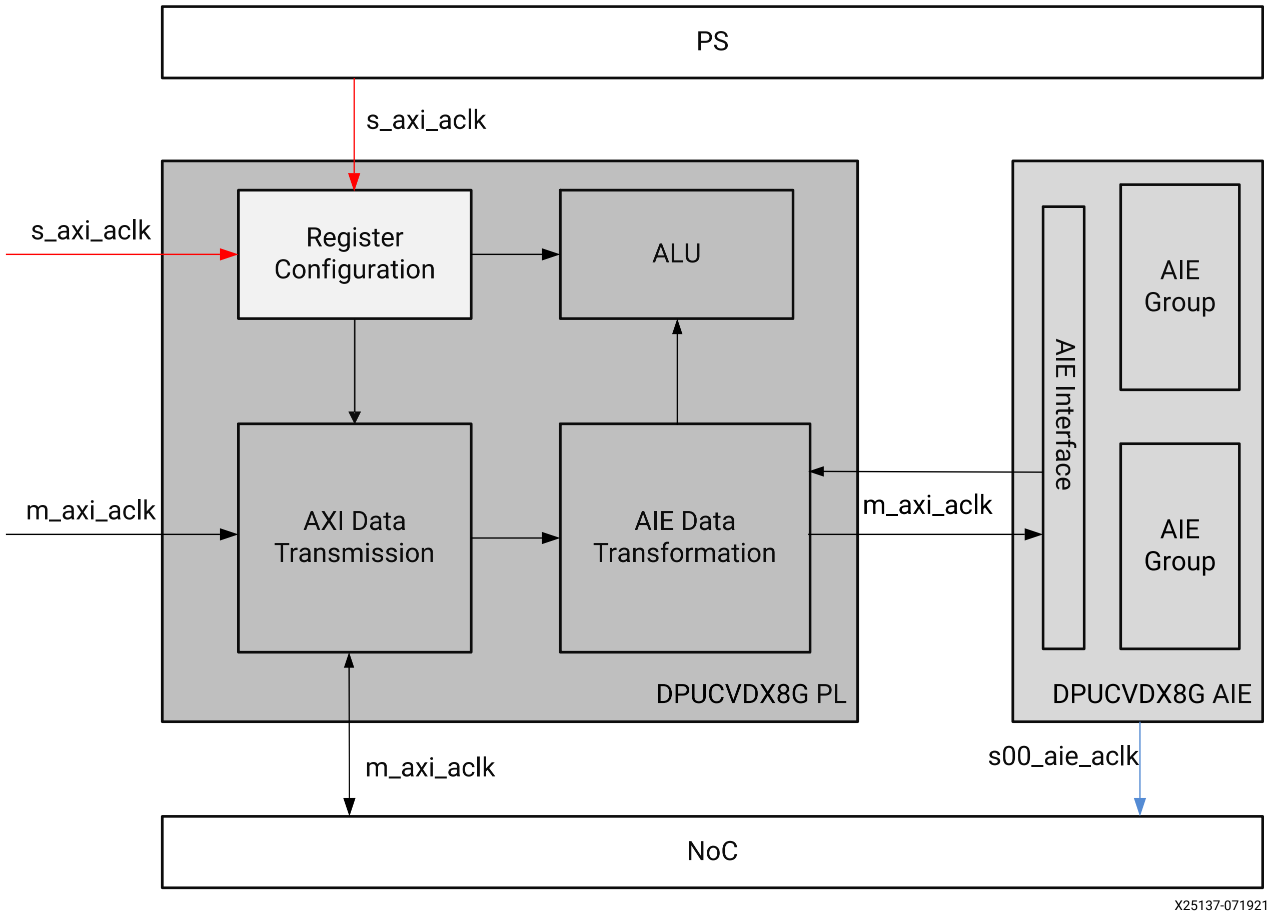
The two input clocks on the PL component can each be configured at different frequencies independently. Generally, the `s_axi_aclk` is set at a lower frequency to obtain a better timing closure. Therefore, the corresponding reset for the two input clocks must be configured correctly.

The `s00_aie_aclk` is the output clock of the AI Engine module. This clock is output to the NoC for data transmission between the AI Engine and the NoC. The frequency of `s00_aie_aclk` can be set in the Vitis flow.

Clock Domains

The following figure shows the three clock domains.

Figure 8: **Clock Domains in the DPUCVDX8G**



- **`s_axi_aclk`**: The `s_axi_aclk` is used for the register configuration module. This module receives the DPUCVDX8G configuration though the S_AXI_CONTROL interface. The DPU configuration registers are updated at a very low frequency and most of those registers are set at the start of a task. It is recommended to use a frequency of 100 MHz for the S-AXI clock.
- **`m_axi_aclk`**: The `m_axi_aclk` is used for most of the logic in the PL component except for the register configuration module. The `m_axi_aclk` is also used for the data transmission between the DPUCVDX8G PL and the NoC. The `m_axi_aclk` is the associated clock for all the AXI4 master interface and AXI4-Stream interface from the PL component. The recommended frequency for this clock is 333 MHz.
- **`s00_aie_aclk`**: The `s00_aie_aclk` is the working clock for the AI Engine interface and the AI Engine array. Generally, the frequency of `s00_aie_aclk` should set as four times of the frequency of `m_axi_aclk`. The frequency of `s00_aie_aclk` can be set in the `postlink.tcl` file in the Vitis flow.

Resets

There are two input clocks for the DPUCVDX8G PL, 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 DPUCVDX8G might not work properly.

The output clock on the AI Engine component does not have a corresponding reset signal.

Example Design

Vitis DPUCVDX8G TRD Flow

The DPUCVDX8G targeted reference design (TRD) provides instructions on how to integrate the DPUCVDX8G on the Versal™ ACAP platform to build and run deep neural network applications. The TRD uses the Vitis™ flow for building the hardware design and the Yocto PetaLinux flow for software design.

For the Vitis DPU TRD flow, refer to https://github.com/Xilinx/Vitis-AI/tree/master/dsa/XVDPU-TRD.

This tutorial contains the following information:

1. Setting up the VCK190 evaluation board.
2. Building and running the TRD with VCK190 platform on the Vitis unified software platform 2020.2.

Appendix A

Upgrading

This appendix is not applicable for the first release of the core.

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. *Versal ACAP Technical Reference Manual* (AM011)
2. *Versal ACAP AI Engine Architecture Manual* (AM009)
3. *VCK190 Evaluation Board User Guide* (UG1366)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
07/22/2021 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright