DPUCAHX8L for Convolutional Neural Networks

PG366 (v1.0) July 22, 2021

Table of Contents

Chapter 1

Introduction

The Xilinx® Deep Learning Processor Unit (DPU) is a series of soft IP for convolutional neural networks acceleration. The DPUCAHX8L is a low latency CNN inference IP for Alveo™ cards with high bandwidth memory (HBM). It runs with a set of efficiently optimized instructions and it can support most convolutional neural networks, such as VGG, ResNet, GoogLeNet, YOLO, SSD, MobileNet, FPN, etc.

Features

- Supports one AXI slave interface for accessing configuration and status registers.
- Supports one AXI master interface for code fetch.
- Supports two AXI master read interface for input feature-map and model parameters loading.
- Supports one AXI master write interface for feature map output.
- Supports eight AXI master interfaces for DPUCAHX8L operation with virtual data banks in the HBM.
- DPU functionality includes the following:
    - Configurable depthwise convolution engines.
    - Convolution and deconvolution
    - Depthwise convolution
    - Max pooling
    - Average pooling
    - ReLU and ReLU6
    - Concat
    - Elementwise-sum
    - Dilation
    - Reorg
    - Fully connected layer

- Batch normalization
- Split

IP Facts

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family	Alveo™ U280 and U50(LV) Data Center accelerator cards
Supported User Interfaces	AXI4-Lite CSR Interface
Resources	Chapter 4: DPU Configuration
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[1]	Xilinx® Runtime (XRT)
Tested Design Flows[2]	
Design Entry	Vitis™ unified software platform
Simulation	N/A
Synthesis	Vivado® Synthesis
Support	
Xilinx Support web page	

Notes:

1. Vitis™ AI development flow.
2. For the supported versions of the tools, see the Linux OS and driver support information are available from the *Vitis Unified Software Platform Documentation: Application Acceleration Development* (UG1393).

Chapter 2

Overview

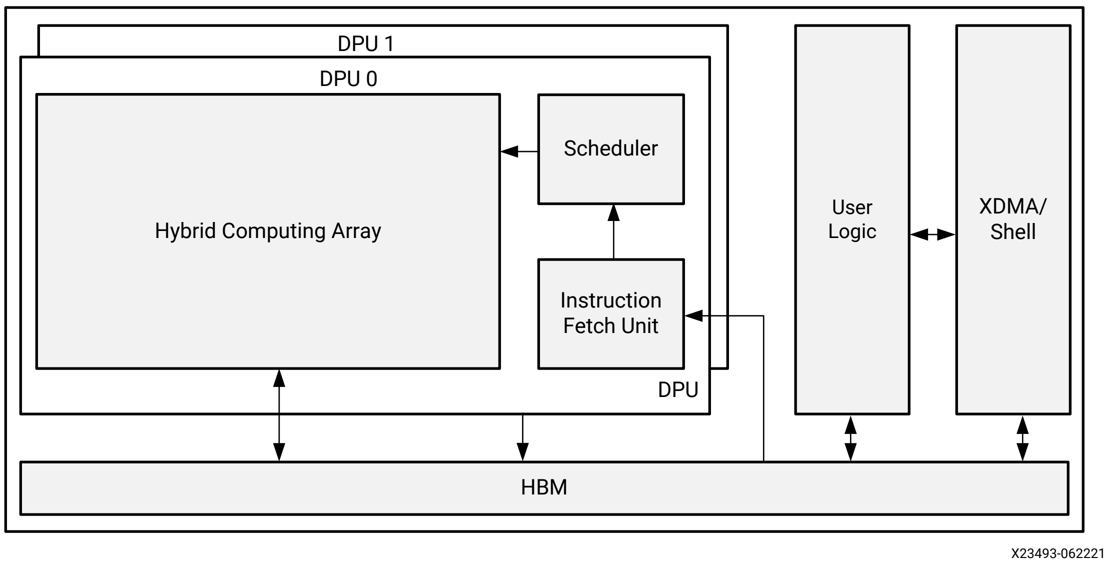
Core Overview

The Xilinx® DPUCAHX8L is a programmable DPU core optimized for convolutional neural networks, mainly for low latency applications. The engine includes a high-performance scheduler module, a hybrid computing array module, and an instruction fetch unit module. It 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, FPN, and many others.

The DPUCAHX8L is implemented in the programmable logic (PL) of the Alveo™ U280 and U50/U50LV Data Center accelerator cards.

The following figure shows the top-level block diagram of the DPUCAHX8L:

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

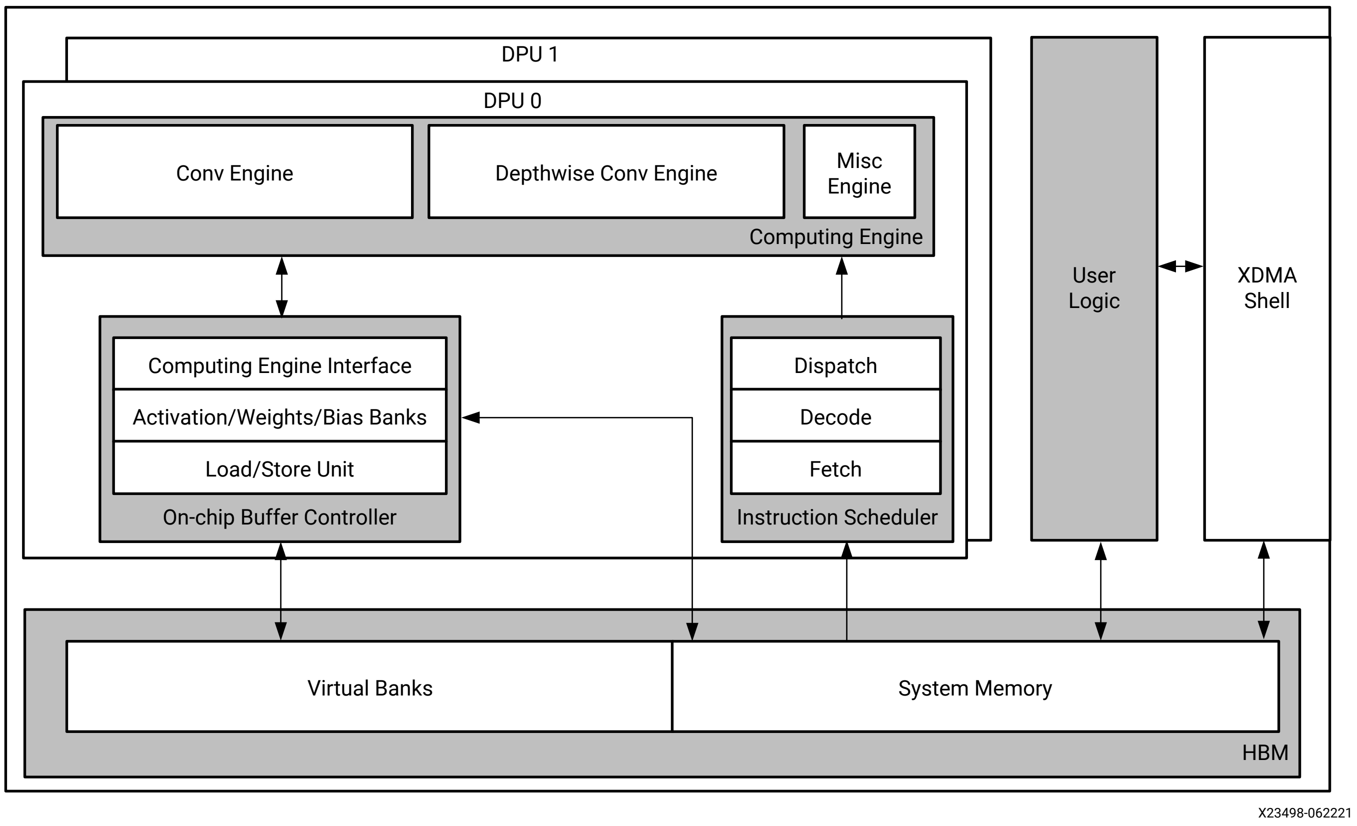


Hardware Architecture

The detailed hardware architecture of the DPUCAHX8L is shown in the following figure. The HBM memory space is divided into virtual banks and system memory. The virtual banks are used to store temporary data and the system memory is used to store instructions, input images, output results, and user data. After starting up, the DPU fetches model instructions from system memory to control the operation of the computing engine. The model instructions are generated by the Vitis AI compiler (running on the host server) which performs substantial optimizations.

On-chip memory is used to buffer weights, bias, intermediate data, and output data to achieve high throughput and efficiency. A deeply pipelined design is used for the computing engine. PEs which include the conv engine, depthwise conv engine, and misc logic take full advantage of the fine-grained building blocks such as multipliers, adders, and accumulators in Xilinx® devices.

Figure 2: **DPU Hardware Architecture**



Development Tools

The Vitis™ Integrated Design Environment (IDE) version 2020.2 is required to integrate the DPU into your projects. Contact your local sales representative if the project requires an older version of the Vitis™ software platform.

Note: For timing closure issues, use Vivado® Design Suite 2020.2 instead of 2021.1.

Device Resources

The DPUCAHX8L can only be deployed on Alveo cards with HBM devices or on the Alveo U50, U50LV, and U280 cards.

For more information on resource utilization, see Chapter 4: DPU Configuration.

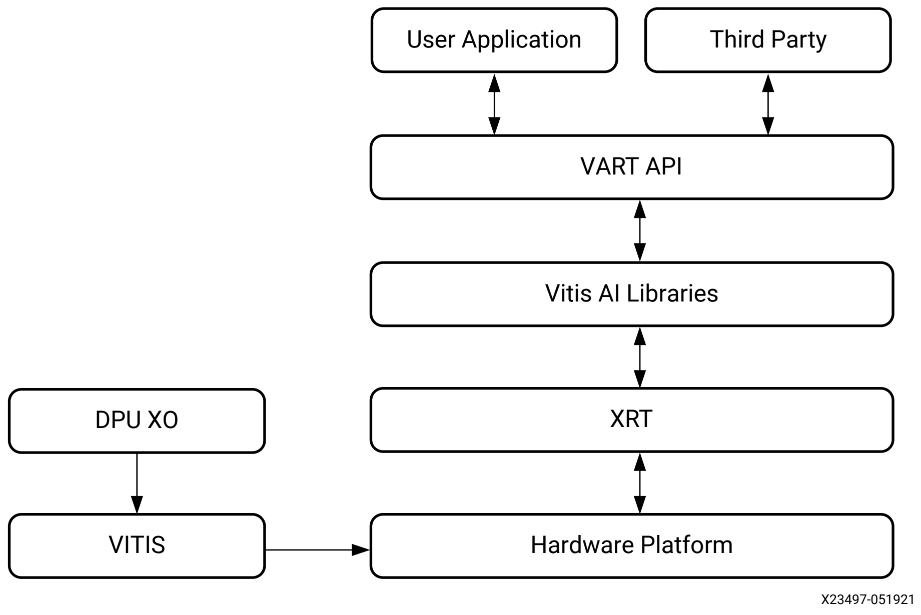
DPU Development Flow

The DPU is available with the software development stack of Vitis AI development kit. Free developer resources can be obtained from the Xilinx website.

The *Vitis AI User Guide* (UG1414) describes how to use the DPU for deploying machine learning applications with the Vitis AI tools. The development flow for DPU applications is summarized in the following steps and shown in the following figure.

1. Use the Vitis tool to generate the bitstream.
2. Download the bitstream to the target board. For instructions on how to set up a running environment for DPU applications, refer to the *Vitis AI User Guide* (UG1414).

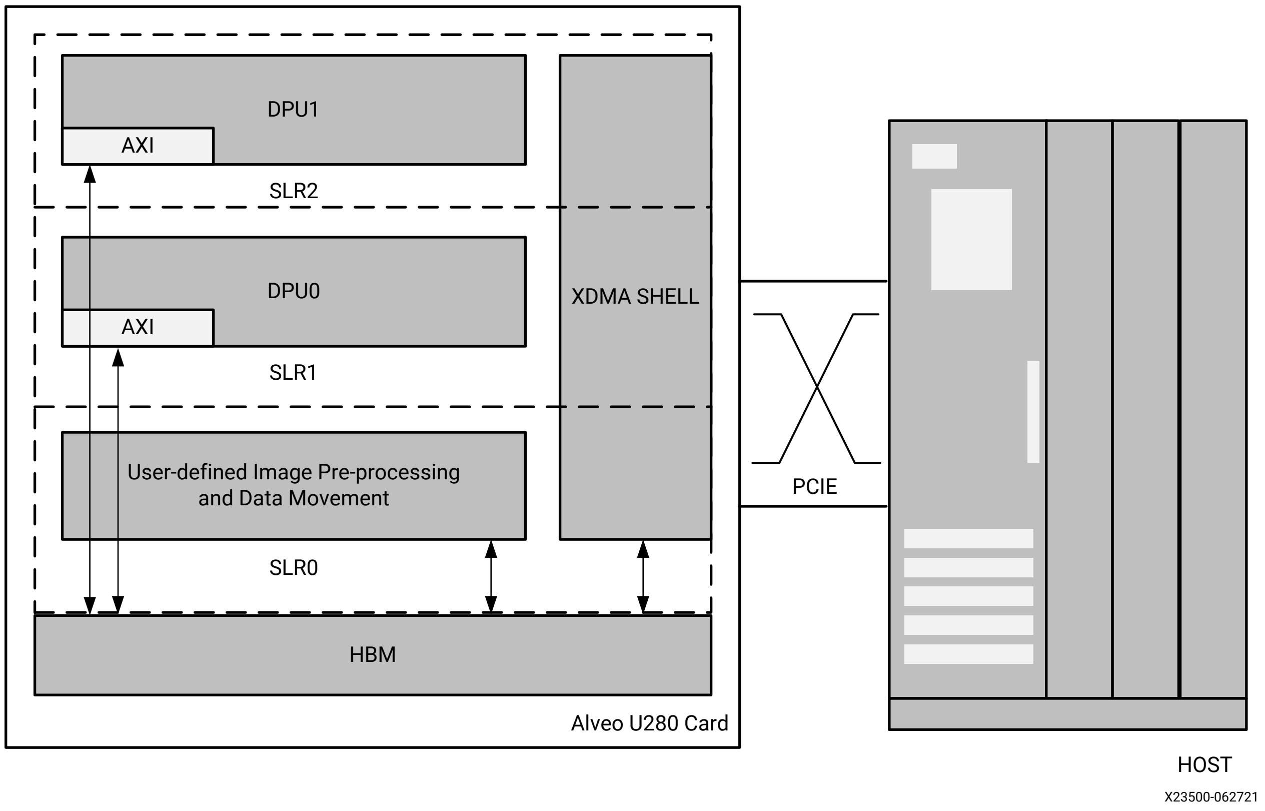
Figure 3: **DPU Development Flow**



Example System with DPUCAHX8L

The following figure shows the example system block diagram with the includes an UltraScale+ XCU280 FPGA and a PCIe® interface. Each implementation has one to two DPU cores and each DPU requires 2 GB HBM memory space. The DPU cores are integrated into the system through the AXI which connects to the HBM, and the whole system is integrated into the server through the PCIe interconnect. It could be used to perform deep learning inference tasks such as image classification, object detection, and semantic segmentation.

Figure 4: **Example System with Integrated DPU**

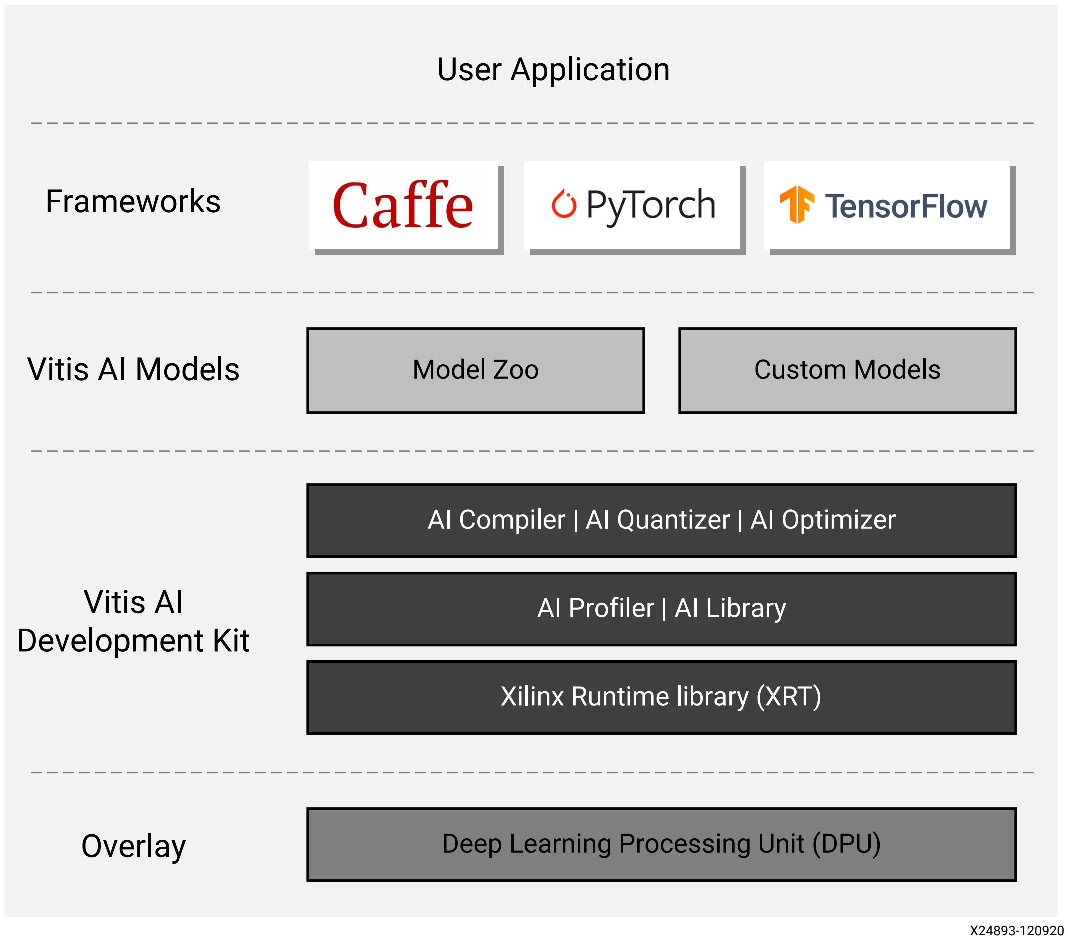


Vitis AI Development Kit

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 5: **Vitis AI Stack**



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

The Vitis AI development kit can be freely downloaded from here.

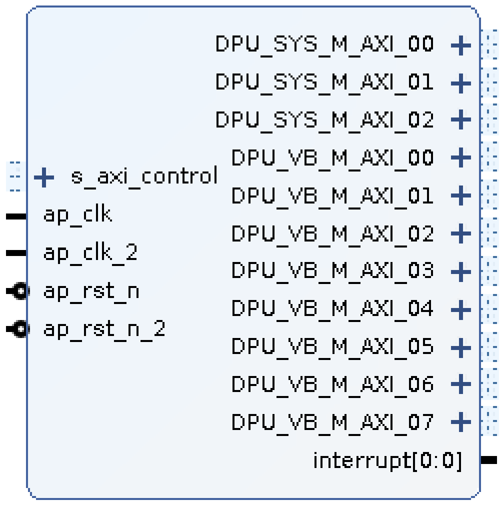
Chapter 3

Product Specification

Port Descriptions

The top-level interfaces of the DPU with five processing engines is shown in the following figure:

Figure 6: **DPU IP Port**



DPU Signals

The following table lists the 5-engine DPU I/O signals and their function descriptions.

Table 1: **DPU Signals**

Signal Name	Interface Type	Width	I/O	Description
s_axi_control	Memory mapped AXI slave interface	32	I/O	32-bit memory mapped AXI interface for registers.

Table 1: **DPU Signals** *(cont'd)*

Signal Name	Interface Type	Width	I/O	Description
ap_clk	Clock	1	I	Kernel clock. The frequency of the clock should match the clock of the DPU core. The supported frequencies are 300 MHz, 275 MHz, and 250 MHz.
ap_clk_2	Clock	1	I	Reference clock for mmcm in the DPU. It is set to 100 MHz.
ap_rst_n	Reset	1	I	Active-Low reset for DPU general logic.
ap_rst_n_2	Reset	1	I	Unused in the core.
DPU_SYS_M_AXI_00	Memory mapped AXI master interface	256	I/O	256-bit memory mapped AXI interface for system data.
DPU_SYS_M_AXI_01	Memory mapped AXI master interface	256	I/O	256-bit memory mapped AXI interface for system data.
DPU_SYS_M_AXI_02	Memory mapped AXI master interface	256	I/O	256-bit memory mapped AXI interface for system data.
DPU_VB_M_AXI_00	Memory mapped AXI master interface	256	I/O	256-bit memory mapped AXI interface for virtual bank data.
DPU_VB_M_AXI_01	Memory mapped AXI master interface	256	I/O	256-bit memory mapped AXI interface for virtual bank data.
DPU_VB_M_AXI_02	Memory mapped AXI master interface	256	I/O	256-bit memory mapped AXI interface for virtual bank data.
DPU_VB_M_AXI_03	Memory mapped AXI master interface	256	I/O	256-bit memory mapped AXI interface for virtual bank data.
DPU_VB_M_AXI_04	Memory mapped AXI master interface	256	I/O	256-bit memory mapped AXI interface for virtual bank data.
DPU_VB_M_AXI_05	Memory mapped AXI master interface	256	I/O	256-bit memory mapped AXI interface for virtual bank data.
DPU_VB_M_AXI_06	Memory mapped AXI master interface	256	I/O	256-bit memory mapped AXI interface for virtual bank data.
DPU_VB_M_AXI_07	Memory mapped AXI master interface	256	I/O	256-bit memory mapped AXI interface for virtual bank data.
`interrupt`	Interrupt	1	O	Active-High interrupt output from DPU.

Notes:

1. The DPU_AXI_1~ DPU_AXI_4 options are shown depending on the number of DPU engines.

Register Space

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

DPU Control Registers

The DPU control registers are used to start a DPU core, waiting for task finish and then clear DPU status. The details of control registers are shown in the following table.

Table 2: **DPU Control Registers**

Register	Address Offset	Width	Type	Description
reg_ap_control	0x000	32	r/w	Bit 0: ap_start (read/write/clear on handshake) Bit 1: ap_done (read/clear on read) Bit 2: ap_idle (read) Others: reserved
Global interrupt enable register (GIER)	0x004	32	r/w	Bit 0: global interrupt enable Others: reserved
IP interrupt enable register (IPIER)	0x008	32	r/w	Bit 0: channel 0 (ap_done) Others: reserved
IP interrupt status register (IPISR)	0x00c	32	r/w	Bit 0: channel 0 (ap_done) (read/ toggle on write) Others: reserved
reg_dpu_start	0x010	32	r/w	Bit [0]: enable DPU to start
reg_finish_clr	0x018	32	r/w	Bit [0]: clear reg_finish_sts
reg_finish_sts	0x080	32	r	Bit [0]: indicate DPU has finished. The DPU finish signal is also output as DPU interrupt to trigger XDMA or custom logic. The DPU finish is a level and asynchronous signal.

DPU Configuration Registers

The DPU configuration registers are used to indicate instruction address, common address and mean value settings.

The reg_instr_addr register is used to indicate the instruction address of the DPU core.

The reg_base_addr register is used to indicate the address of input image and parameters for the DPU in external memory. The width of a DPU base address is 33 bits so it can support an address space up to 8 GB. All registers are 32-bit wide, so two registers are required to represent a 33-bit wide base address. The reg_base_addr0_l register represents the lower 32 bits of base_address0, and reg_base_addr0_h represents the upper 1 bit of base_address0. There are eight groups of DPU base address.

The details of configuration registers are shown in the following figure.

Table 3: **DPU Configuration Registers**

Register	Address Offset	Width	Type	Description
reg_base_addr_0_l	0x100	32	r/w	The lower 32 bits of base address0 of DPU. 4 KB aligned.
reg_base_addr_0_h	0x104	32	r/w	The lower 1-bit in the register represent the upper 1-bit of base address0 of DPU.
reg_base_addr_1_l	0x108	32	r/w	The lower 32 bits of base address1 of DPU. 4 KB aligned.
reg_base_addr_1_h	0x10c	32	r/w	The lower 1-bit in the register represent the upper 1-bit of base address1 of DPU.
reg_base_addr_2_l	0x110	32	r/w	The lower 32 bits of base address2 of DPU. 4 KB aligned.
reg_base_addr_2_h	0x114	32	r/w	The lower 1-bit in the register represent the upper 1-bit of base address2 of DPU.
reg_base_addr_3_l	0x118	32	r/w	The lower 32 bits of base address3 of DPU. 4 KB aligned.
reg_base_addr_3_h	0x11c	32	r/w	The lower 1-bit in the register represent the upper 1-bit of base address3 of DPU.
reg_base_addr_4_l	0x120	32	r/w	The lower 32 bits of base address4 of DPU. 4 KB aligned.
reg_base_addr_4_h	0x124	32	r/w	The lower 1-bit in the register represent the upper 1-bit of base address4 of DPU.
reg_base_addr_5_l	0x128	32	r/w	The lower 32 bits of base address5 of DPU. 4 KB aligned.
reg_base_addr_5_h	0x12c	32	r/w	The lower 1-bit in the register represent the upper 1-bit of base address5 of DPU.
reg_base_addr_6_l	0x130	32	r/w	The lower 32 bits of base address6 of DPU. 4 KB aligned.
reg_base_addr_6_h	0x134	32	r/w	The lower 1-bit in the register represent the upper 1-bit of base address6 of DPU.
reg_base_addr_7_l	0x138	32	r/w	The lower 32 bits of base address7 of DPU. 4 KB aligned.
reg_base_addr_7_h	0x13c	32	r/w	The lower 1-bit in the register represent the upper 1-bit of base address7 of DPU.
reg_instr_addr_l	0x140	32	r/w	The lower 32 bits of instruction address of DPU. 4 KB aligned.
reg_instr_addr_h	0x144	32	r/w	The lower 1-bit in the register represent the upper 1-bit of instruction address of DPU. 4 KB aligned.

DPU Debug Registers

The DPU debug registers are used to indicate the processing cycles for task.

The details of debug registers are shown in the following table.

Table 4: **DPU Debug Registers**

Register	Address Offset	Width	Type	Description
reg_prof_value	0x0a8	32	r	Indicates the cycle counter of DPU processing time. Saturation counting.

Interrupts

The DPU generates an interrupt to signal the completion of a task. A high state on `reg_dpu_start` or `ap_start` signals the start of a DPU task. At the end of the task, the DPU generates an interrupt and bit0 in IPISR and `reg_finish_sts` is set to 1.

To support DPU interrupt, the DPU implements the following registers:

- **Global Interrupt Enable Register (GIER):** Provides the master enable/disable for the interrupt output to the processor or Interrupt Controller. See Global Interrupt Enable Register (GIER) in Table 2 for more details.
- **IP Interrupt Enable Register (IPIER):** Implements the independent interrupt enable bit for each channel. See IP Interrupt Enable (IPIER) and IP Status Registers (IPISR) in Table 2 for more details.
- **IP Interrupt Status Register (IPISR):** Implements the independent interrupt status bit for each channel. The IPISR provides Read and Toggle-On-Write access. The Toggle-On-Write mechanism allows interrupt service routines to clear one or more ISR bits using a single write transaction. The IPISR can also be manually set to generate an interrupt for testing purposes. See IP Interrupt Enable (IPIER) and IP Status Registers (IPISR) in Table 1 for additional details.

DPU Configuration

The DPU core 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 is an option to determine the frequency of DPU cores that will be implemented on the board. The deep neural network features and the associated parameters supported by the DPU are shown in the following table.

Table 5: **Deep Neural Network Features and Parameters Supported by DPU**

Features	Description (channel_parallel=32)	
conv2d	Kernel Sizes	kernel_w: [1, 16] kernel_h: [1, 16]
	Strides	stride_w: [1, 4] stride_h: [1, 4]
	Pad_left/Pad_right	[0, (kernel_w - 1) * dilation_w + 1]
	Pad_top/Pad_bottom	[0, (kernel_h - 1) * dilation_h + 1]
	In Size	kernel_w * kernel_h * ceil(input_channel / channel_parallel) <= 4096
	Out Size	output_channel <= 256 * channel_parallel
	Activation	ReLU, ReLU6
	Dilation	dilation * input_channel <= 256 * channel_parallel
depthwise-conv2d	Kernel Sizes	kernel_w: [3] kernel_h: [3]
	Strides	stride_w: [1, 2] stride_h: [1, 2]
	Pad_left/Pad_right	[0, (kernel_w - 1) * dilation_w + 1]
	Pad_top/Pad_bottom	[0, (kernel_h - 1) * dilation_h + 1]
	In Size	kernel_w * kernel_h * ceil(input_channel / channel_parallel) <= 4096
	Out Size	output_channel <= 256 * channel_parallel
	Activation	ReLU, ReLU6
	Dilation	dilation * input_channel <= 256 * channel_parallel

Table 5: **Deep Neural Network Features and Parameters Supported by DPU** *(cont'd)*

Features	Description (channel_parallel=32)	
transposed-conv2d	Kernel Sizes	kernel_w: [1, 16] kernel_h: [1, 16]
	Strides	kernel_w: [1, 16] kernel_h: [1, 16]
	Pad_left/Pad_right	[1, kernel_w-1]
	Pad_top/Pad_bottom	[1, kernel_h-1]
	Out Size	output_channel <= 256 * channel_parallel
	Activation	ReLU, ReLU6
depthwise-transposed-conv2d	Kernel Sizes	kernel_w: [3] kernel_h: [3]
	Strides	kernel_w: [3] kernel_h: [3]
	Pad_left/Pad_right	[1, kernel_w-1]
	Pad_top/Pad_bottom	[1, kernel_h-1]
	Out Size	output_channel <= 256 * channel_parallel
	Activation	ReLU, ReLU6
max-pooling	Strides	kernel_w: [1, 8] kernel_h: [1, 8]
	Kernel Sizes	W, H: {2, 3, 5, 7, 8}
	Pad_left/Pad_right	[1, kernel_w-1]
	Pad_top/Pad_bottom	[1, kernel_h-1]
average-pooling	Kernel Sizes	kernel_w, kernel_h: {2, 3, 5, 7, 8} kernel_w==kernel_h
	Strides	kernel_w: [1, 8] kernel_h: [1, 8]
	Pad_left/Pad_right	[1, kernel_w-1]
	Pad_top/Pad_bottom	[1, kernel_h-1]
elementwise-sum	Input channel	input_channel <= 256 * channel_parallel
	Activation	ReLU
Fully Connected	Input Channel	input channel <= 16*16*32
Concat	Network-specific limitation related to the size of feature maps, quantization results, and compiler optimizations.	

Configuration Options

The DPUCAHX8L can be configured with some predefined options which include the number of processing engines, frequency, and card type. These options are used to choose how many engines to implement and the frequency of the IP.

Resource Utilization

The resource utilization of one DPUCAHX8L on the Alveo U280 card is shown in the following table.

Table 6: **DPUCAHX8L Utilization**

Architecture	LUT	Register	Block RAM	UltraRAM	DSP
DPUCAHX8L	212860	299342	459	312	2452

Performance

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

Table 7: **DPU Performance on Different Device**

Device	DPU Configuration	Frequency (MHz)	Peak Performance
Alveo U280 card	1 core	300	4.8 Tops
Alveo U280 card	2 core	250	8.0 Tops

Chapter 5

Development Flow

Customizing and Generating the Core in Shell Mode with Vitis Flow

The following sections describe the development flow on how to use the DPU IP with the Vitis™ IDE.

Generating .xo Files

The .xo file is a format of IP that can be used by the shell in the Vitis flow. DPU IP files are released as .xo files.

1. Download the DPUCAHX8L_xo_gen_flow.tar package to generate the DPU xo files. The package includes DPU related encrypted RTL and timing constrain files.
2. cd to the path: DPUCAHX8L_xo_gen_flow.
3. Refer Makefile to generate the DPU .xo files with required options.

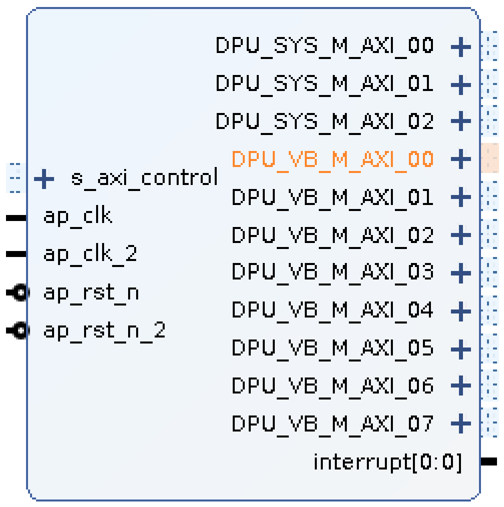
```
make help:
==========================
  Xilinx DPUCAHX8L IP/XO RELEASE HELP
  ==========================
  release_DPUCAHX8L_A_xo : DPUCAHX8L_A xo file release
  release_DPUCAHX8L_B_xo : DPUCAHX8L_B xo file release
```

The differences between DPUCAHX8L_A and DPUCAHX8L_B are as follows:

- They use different memory banks of HBM as their virtual bank. DPUCAH8XL_A used 0-2 GB and DPUCAH8XL_B uses 2-4 GB.
- regslices insert is different because the location of SLRs is different in the DPUs.

The interface of the DPU xo is as following figure:

Figure 7: **DPU xo Block**



Adding the DPU xo Files to the Implementation Flow

To add the DPU xo files to the implementation flow, follow these steps:

1. Add following constraints into `cons.ini` file:

```
[connectivity]
nk=DPUCAHX8L_A:1:DPUCAHX8L_A
nk=DPUCAHX8L_B:1:DPUCAHX8L_B
```

2. Run the v++ command with the `--config "cons.ini"` option.

Configuring the HBM

The Alveo™ U50, U50LV, and U280 cards support HBM. This section describes how to customize the HBM IP to meet DPU requirements and improve performance.

Although each AXI port connected to the HBM controller (HMSS) can access all the DDR memory banks within the HBM, the efficiency to access the DDR memory bank directly connected or within one switch is much higher than to access the DDR memory bank which is far from the AXI port. To get better access performance, each feature map AXI port must be constrained at a DDR memory bank to avoid latency caused by inefficient horizontal access.

For the DPUCAHX8L_A, 0~2 GB of space should not be used by your kernel. For the DPUCAHX8L_B, it is 2~4 GB.

To configure the HBM, follow these steps:

1. Add the sp constraints to the `cons.ini` file. In this example, the connectivity of each AXI port covers full HBM-range access.

```
[connectivity]
nk=DPUCAHX8L_A:1:DPUCAHX8L_A
sp=DPUCAHX8L_A.DPU_VB_M_AXI_00:HBM[00:31]
sp=DPUCAHX8L_A.DPU_VB_M_AXI_01:HBM[00:31]
sp=DPUCAHX8L_A.DPU_VB_M_AXI_02:HBM[00:31]
sp=DPUCAHX8L_A.DPU_VB_M_AXI_03:HBM[00:31]
sp=DPUCAHX8L_A.DPU_VB_M_AXI_04:HBM[00:31]
sp=DPUCAHX8L_A.DPU_VB_M_AXI_05:HBM[00:31]
sp=DPUCAHX8L_A.DPU_VB_M_AXI_06:HBM[00:31]
sp=DPUCAHX8L_A.DPU_VB_M_AXI_07:HBM[00:31]
sp=DPUCAHX8L_A.DPU_SYS_M_AXI_00:HBM[00:31]
sp=DPUCAHX8L_A.DPU_SYS_M_AXI_01:HBM[00:31]
sp=DPUCAHX8L_A.DPU_SYS_M_AXI_02:HBM[00:31]
nk=DPUCAHX8L_B:1:DPUCAHX8L_B
sp=DPUCAHX8L_B.DPU_VB_M_AXI_00:HBM[00:31]
sp=DPUCAHX8L_B.DPU_VB_M_AXI_01:HBM[00:31]
sp=DPUCAHX8L_B.DPU_VB_M_AXI_02:HBM[00:31]
sp=DPUCAHX8L_B.DPU_VB_M_AXI_03:HBM[00:31]
sp=DPUCAHX8L_B.DPU_VB_M_AXI_04:HBM[00:31]
sp=DPUCAHX8L_B.DPU_VB_M_AXI_05:HBM[00:31]
sp=DPUCAHX8L_B.DPU_VB_M_AXI_06:HBM[00:31]
sp=DPUCAHX8L_B.DPU_VB_M_AXI_07:HBM[00:31]
sp=DPUCAHX8L_B.DPU_SYS_M_AXI_00:HBM[00:31]
sp=DPUCAHX8L_B.DPU_SYS_M_AXI_01:HBM[00:31]
sp=DPUCAHX8L_B.DPU_SYS_M_AXI_02:HBM[00:31]
```

2. Use the `--config "cons.ini"` command to the v++ command.

3. Set the corresponding connections to HBM Memory Sub-System (HMSS) and set the `host_port` address range (S00_MEM). Include all the memory ranges that are used in the `sys_link_post.tcl`.

4. Put the following code at `sys_link_post.tcl` for one core:

```
hbm_memory_subsystem::force_host_port 28 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_00] 0 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_01] 1 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_02] 2 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_03] 3 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_04] 4 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_05] 5 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_06] 6 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_07] 7 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_SYS_M_AXI_00] 29 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_SYS_M_AXI_01] 24 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_SYS_M_AXI_02] 25 1 [get_bd_cells hmss_0]

set ap [get_property CONFIG.ADVANCED_PROPERTIES [get_bd_cells /hmss_0]]
```

```
dict set ap minimal_initial_configuration true
set_property CONFIG.ADVANCED_PROPERTIES $ap [get_bd_cells /hmss_0]

set_param bd.hooks.addr.debug_scoped_use_ms_name true
assign_bd_address [get_bd_addr_segs {DPUCAHX8L_A/s_axi_control/reg0 }]
```

For two cores, use the following code:

```
hbm_memory_subsystem::force_host_port 28 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_00]  0 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_01]  2 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_02]  1 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_03]  3 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_04]  4 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_05]  7 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_06]  5 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_VB_M_AXI_07]  6 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_SYS_M_AXI_00]  26 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_SYS_M_AXI_01]  24 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_A/
DPU_SYS_M_AXI_02]  25 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_B/
DPU_VB_M_AXI_00]  8 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_B/
DPU_VB_M_AXI_01]  10 1 [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_B/
DPU_VB_M_AXI_02]  9 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_B/
DPU_VB_M_AXI_03]  11 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_B/
DPU_VB_M_AXI_04]  12 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_B/
DPU_VB_M_AXI_05]  15 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_B/
DPU_VB_M_AXI_06]  13 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_B/
DPU_VB_M_AXI_07]  14 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_B/
DPU_SYS_M_AXI_00]  27 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_B/
DPU_SYS_M_AXI_01]  22 1  [get_bd_cells hmss_0]
hbm_memory_subsystem::force_kernel_port [get_bd_intf_pins /DPUCAHX8L_B/
DPU_SYS_M_AXI_02]  23 1  [get_bd_cells hmss_0]

set ap [get_property CONFIG.ADVANCED_PROPERTIES [get_bd_cells /hmss_0]]
dict set ap minimal_initial_configuration
true

                                             set_property
CONFIG.ADVANCED_PROPERTIES $ap [get_bd_cells /hmss_0]

set_param bd.hooks.addr.debug_scoped_use_ms_name true
```

```
assign_bd_address [get_bd_addr_segs {DPUCAHX8L_A/s_axi_control/reg0 }]
assign_bd_address [get_bd_addr_segs {DPUCAHX8L_B/s_axi_control/reg0 }]
validate_bd_design -force

validate_bd_design -force
```

5. Add the following lines to the `cons.ini` file.

```
[advanced]
param=compiler.userPostSysLinkTcl=*/sys_link_post.tcl
```

Making HBM Connections

DPUCAHX8L can be deployed with a single core on the Alveo U50 card and with one or two cores on the Alveo U280 card. The cores are named DPUCAHX8L_A and DPUCAHX8L_B. You can locate DPUCAHX8L_A on SLR1 and DPUCAHX8L_B on SLR2 .To get the best performance of HBM, DPU AXI ports should be connected in reference of `sys_link_post.tcl`.

Generating the Bitstream

Run the following command to generate the bitstream:

```
v++ -t hw --platform $platform_name --temp_dir $direction_name -l --config
"<vitis configuration file - *.ini>" -o dpu.xclbin DPUCAHX8L_A.xo
```

If two xo are used, add the xo file names to the command as shown in the following example for implementing on the Alveo U280 card:

```
v++ -t hw --platform xilinx_u280_xdma_201920_3 --save-temps -temp_dir haha -
l --config "<vitis configuration file - *.ini>" -o dpu.xclbin
DPUCAHX8L_A.xo DPUCAHX8L_B.xo
```

Chapter 6

Upgrading

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

Chapter 7

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)

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

Copyright