Zynq DPU v3.2

Product Guide

⚡ XILINX®

Table of Contents

Introduction

The Xilinx® Deep Learning Processing Unit (DPU) is a configurable computation engine dedicated to 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 DPU has the following features:

- One AXI slave interface for accessing configuration and status registers.

- One AXI master interface for accessing instructions.

- Supports configurable AXI master interface with 64 or 128 bits for accessing data depending on the target device.

- Supports individual configuration of each channel.

- Supports optional interrupt request generation.

- Some highlights of DPU functionality include:

  - Configurable hardware architecture includes: B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096

  - Maximum of four cores

  - Convolution and deconvolution

  - Depthwise convolution

  - Max pooling

  - Average pooling

  - ReLU, ReLU6, and Leaky ReLU

  - Concat

  - Elementwise-sum

- ◦ Dilation

- ◦ Reorg

- ◦ Fully connected layer

- ◦ Softmax

- ◦ Batch Normalization

- ◦ Split

IP Facts

DPU IP Facts Table	
Core Specifics	
Supported Device Family	Zynq®-7000 SoC and 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
Simulation	N/A
Synthesis	Vivado® Synthesis
Support	
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. Linux OS and driver support information are available from DPU TRD or Vitis™ AI development kit.

2. If the target device is Zynq-7000 SoC, see the notifications in Chapter 6: Development Flow.

3. For the supported tool versions, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973).

4. The DPU is driven by instructions generated by the Vitis AI compiler. When the target NN, DPU hardware architecture, or AXI data width is changed, the related .elf file which contains DPU instructions must be regenerated.

Overview

Introduction

The Xilinx® Deep Learning Processing Unit (DPU) is a programmable engine optimized for convolutional neural networks. The unit includes a high performance scheduler module, a hybrid computing array module, an instruction fetch unit module, and a global memory pool module. The DPU 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 DPU IP can be implemented in the programmable logic (PL) of the selected Zynq®-7000 SoC or Zynq® UltraScale+™ MPSoC device with direct connections to the processing system (PS). The DPU requires instructions to implement a neural network and accessible memory locations for input images as well as temporary and output data. A program running on the application processing unit (APU) is also required to service interrupts and coordinate data transfers.

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

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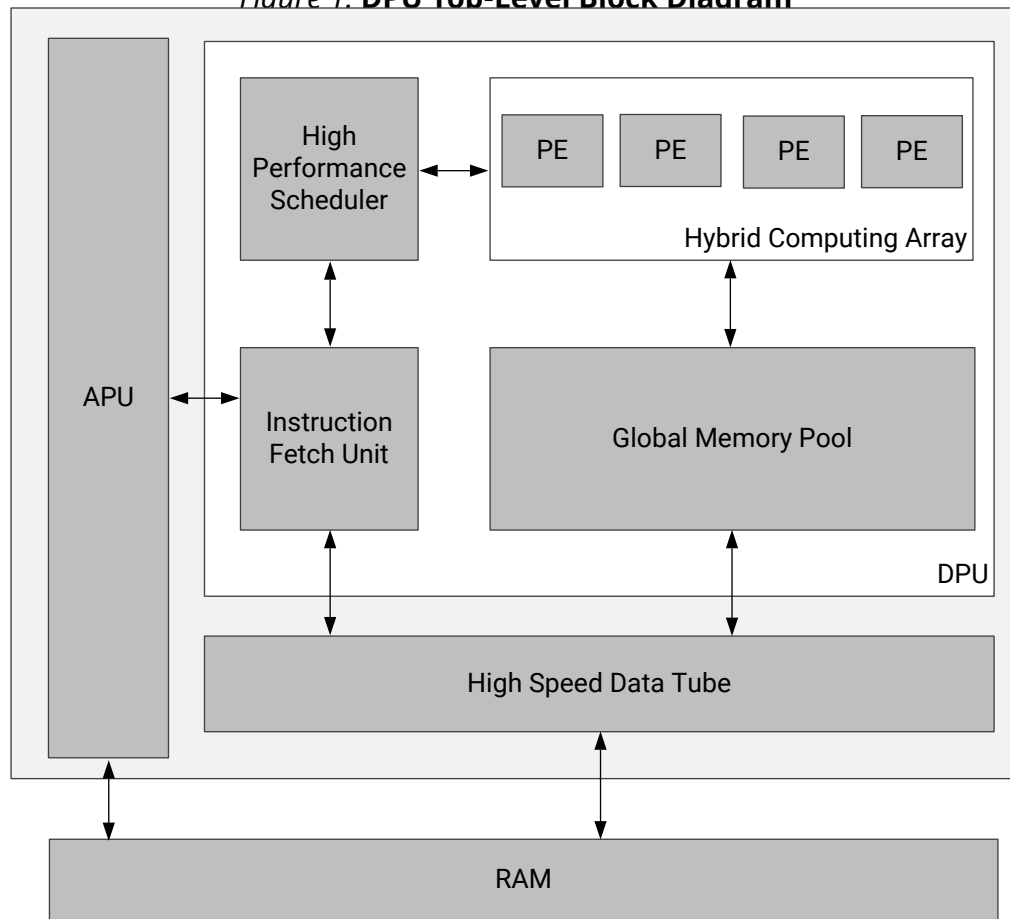
where,

- APU - Application Processing Unit

- PE - Processing Engine

- DPU - Deep Learning Processing Unit

- RAM - Random Access Memory

Development Tools

The Xilinx® Vivado® Design Suite is required to integrate the DPU into your projects. Vivado Design Suite 2019.2 or later version is recommended. Contact your local sales representative if the project requires an older version of Vivado.

The Vitis unified software platform 2019.2 is required to integrate the DPU for the Vitis flow.

Device Resources

The DPU logic resource usage is scalable across Xilinx UltraScale+™ MPSoC and Zynq®-7000 devices. For more information on resource utilization, see the DPU Configuration section.

Related Information
DPU Configuration

DPU Development Flow

The DPU 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://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge.

The Vitis AI User Guide (UG1414) describes how to use the DPU with the Vitis AI tools. The basic development flow is shown in the following figure. First, use Vivado/ 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 2: **Basic Development Flow**
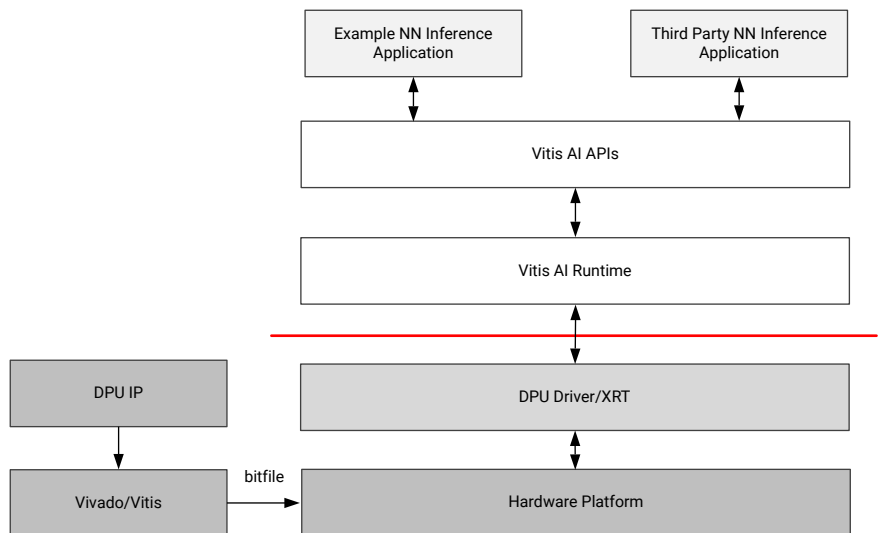
X22328-031620

Example System with DPU

The figure below shows an example system block diagram with the Xilinx® UltraScale+™ MPSoC using a camera input. The DPU 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 3: **Example System with Integrated DPU**
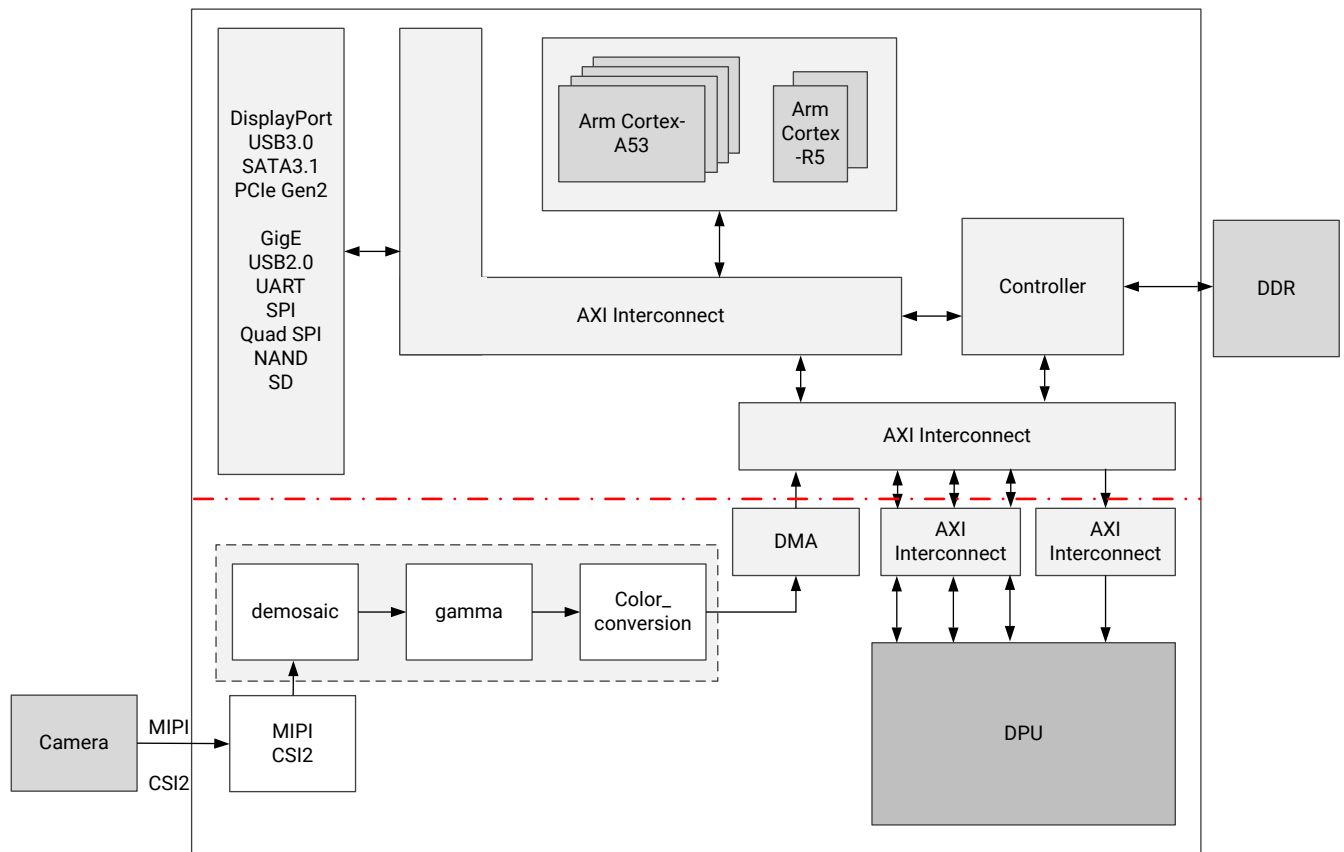
X22329-081919

Vitis AI Development Kit

The Vitis™ AI development environment consists of the Vitis AI development kit, for AI inference on the 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**
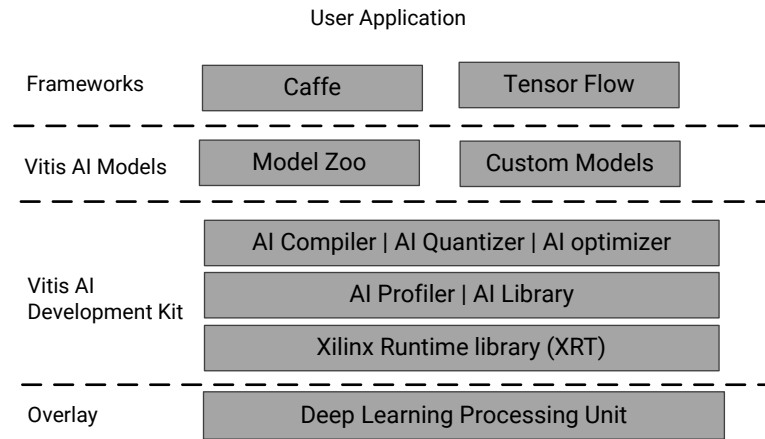
For more information on the Vitis AI development kit, see the *Vitis AI User Guide* in the *Vitis AI User Documentation* (UG1431).

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

Licensing and Ordering

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License.

Information about other Xilinx® LogiCORE™ IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Hardware Architecture

The detailed hardware architecture of the DPU is shown in the following figure. After start-up, the DPU 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 have been 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 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 5: **DPU Hardware Architecture**

X22332-022420

Send Feedback
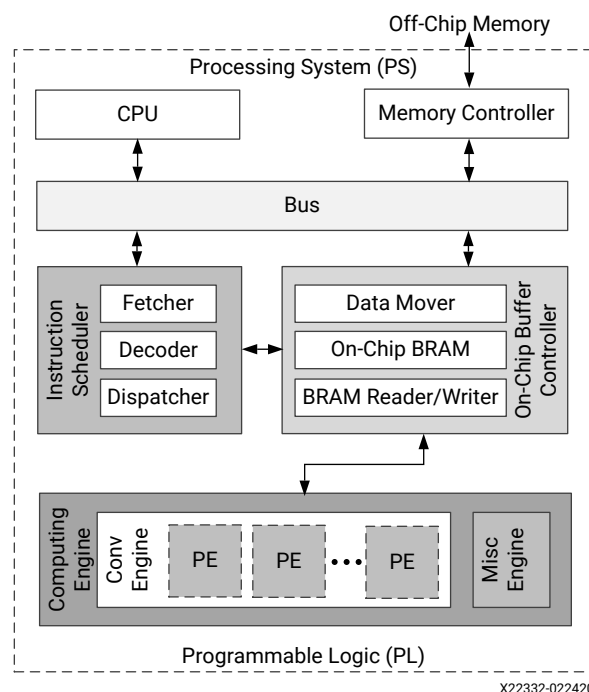
DPU with Enhanced Usage of DSP

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

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

Figure 6: **Difference between DPU without DSP DDR and DPU Enhanced Usage**
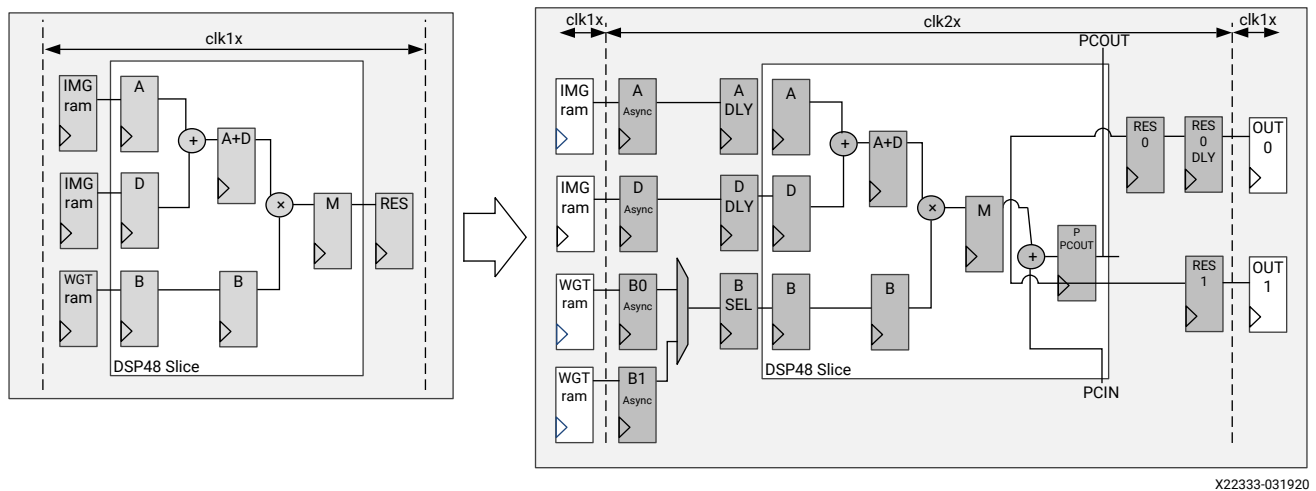
X22333-031920

Port Descriptions

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

Send Feedback

Figure 7: **Core Ports**

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

Table 1: **DPU Signal Description**

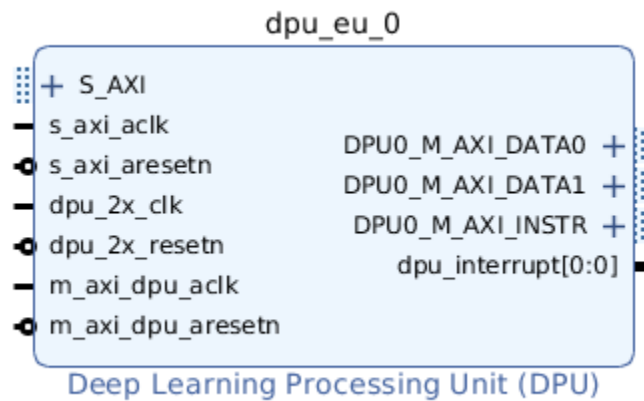| Signal Name	Interface Type	Width	I/O	Description
S_AXI	Memory mapped AXI slave interface	32	I/O	32-bit memory mapped AXI interface for registers.
s_axi_aclk	Clock	1	I	AXI clock input for S_AXI
s_axi_aresetn	Reset	1	I	Active-Low reset for S_AXI
dpu_2x_clk	Clock	1	I	Input clock used for DSP blocks in the DPU. The frequency is twice that of m_axi_dpu_aclk.
dpu_2x_resetn	Reset	1	I	Active-Low reset for DSP blocks
m_axi_dpu_aclk	Clock	1	I	Input clock used for DPU general logic.
m_axi_dpu_aresetn	Reset	1	I	Active-Low reset for DPU general logic
DPUx_M_AXI_INSTR	Memory mapped AXI master interface	32	I/O	32-bit memory mapped AXI interface for DPU instructions.
DPUx_M_AXI_DATA0	Memory mapped AXI master interface	64 or 128	I/O	64 or 128-bit memory mapped AXI interface for DPU data.
DPUx_M_AXI_DATA1	Memory mapped AXI master interface	64 or 128	I/O	64 or 128-bit memory mapped AXI interface for DPU data.
dpu_interrupt	Interrupt	1~3	O	Active-High interrupt output from DPU. The data width is determined by the number of DPU cores.
SFM_M_AXI (optional)	Memory mapped AXI master interface	128	I/O	128-bit memory mapped AXI interface for softmax data.
sfm_interrupt (optional)	Interrupt	1	O	Active-High interrupt output from softmax module.

Send Feedback

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

Signal Name	Interface Type	Width	I/O	Description
dpu_2x_clk_ce (optional)	Clock enable	1	O	Clock enable signal for controlling the input DPU 2x clock when DPU 2x clock gating is enabled.

Notes:

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

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 interface.

reg_dpu_reset

The reg_dpu_reset register controls the resets of all DPU cores integrated in the DPU IP. The lower three bits of this register control the reset of up to three DPU 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] – DPU core n reset

reg_dpu_isr

The reg_dpu_isr register represents the interrupt status of all cores in the DPU IP. The lower three bits of this register shows the interrupt status of up to three DPU 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] – DPU core n interrupt status

reg_dpu_start

The reg_dpu_start register is the start signal for a DPU core. There is one start register for each DPU 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	DPU core0 start signal.
reg_dpu1_start	0x320	32	R/W	DPU core1 start signal.
reg_dpu2_start	0x420	32	R/W	DPU core2 start signal.

reg_dpu_instr_addr

The reg_dpu_instr_addr register is used to indicate the instruction address of a DPU core. Each DPU core has a reg_dpu_instr_addr register. The reg_dpu_instr_addr is lower 28-bit valid. In the DPU processor, the real instruction-fetch address is a 40-bit signal which consist of the lower 28-bit of reg_dpu_instr_addr and 12-bit zero. 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 DPU core0 instructions. The lower 28-bit is valid.
reg_dpu1_instr_addr	0x30C	32	R/W	Start address in external memory for DPU core1 instructions. The lower 28-bit is valid.
reg_dpu2_instr_addr	0x40C	32	R/W	Start address in external memory for DPU core2 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 DPU in external memory. The width of a DPU 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 DPU core0 and reg_dpu0_base_addr0_h represents the upper eight bits of base_address0 in DPU core0.

There are eight groups of DPU base addresses for each DPU core and thus 24 groups of DPU base addresses for up to three DPU 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 DPU 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 DPU core0.
reg_dpu0_base_addr1_l	0x22C	32	R/W	The lower 32 bits of base_address1 of DPU 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 DPU core0.
reg_dpu0_base_addr2_l	0x234	32	R/W	The lower 32 bits of base_address2 of DPU 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 DPU core0.
reg_dpu0_base_addr3_l	0x23C	32	R/W	The lower 32 bits of base_address3 of DPU 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 DPU core0.
reg_dpu0_base_addr4_l	0x244	32	R/W	The lower 32 bits of base_address4 of DPU core0.
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 DPU core0.
reg_dpu0_base_addr5_l	0x24C	32	R/W	The lower 32 bits of base_address5 of DPU 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 DPU core0.
reg_dpu0_base_addr6_l	0x254	32	R/W	The lower 32 bits of base_address6 of DPU 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 DPU core0.
reg_dpu0_base_addr7_l	0x25C	32	R/W	The lower 32 bits of base_address7 of DPU 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 DPU core0.
reg_dpu1_base_addr0_l	0x324	32	R/W	The lower 32 bits of base_address0 of DPU 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 DPU core1.
reg_dpu1_base_addr1_l	0x32C	32	R/W	The lower 32 bits of base_address1 of DPU 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 DPU core1.

Send Feedback

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

Register	Address Offset	Width	Type	Description
reg_dpu1_base_addr2_l	0x334	32	R/W	The lower 32 bits of base_address2 of DPU 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 DPU core1.
reg_dpu1_base_addr3_l	0x33C	32	R/W	The lower 32 bits of base_address3 of DPU 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 DPU core1.
reg_dpu1_base_addr4_l	0x344	32	R/W	The lower 32 bits of base_address4 of DPU 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 DPU core1.
reg_dpu1_base_addr5_l	0x34C	32	R/W	The lower 32 bits of base_address5 of DPU 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 DPU core1.
reg_dpu1_base_addr6_l	0x354	32	R/W	The lower 32 bits of base_address6 of DPU core1.
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 DPU core1.
reg_dpu1_base_addr7_l	0x35C	32	R/W	The lower 32 bits of base_address7 of DPU 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 DPU core1.
reg_dpu2_base_addr1_l	0x42C	32	R/W	The lower 32 bits of base_address1 of DPU 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 DPU core2.
reg_dpu2_base_addr2_l	0x434	32	R/W	The lower 32 bits of base_address2 of DPU 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 DPU core2.
reg_dpu2_base_addr3_l	0x43C	32	R/W	The lower 32 bits of base_address3 of DPU 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 DPU core2.
reg_dpu2_base_addr4_l	0x444	32	R/W	The lower 32 bits of base_address4 of DPU 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 DPU core2.

Send Feedback

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

| Register	Address Offset	Width	Type	Description
reg_dpu2_base_addr5_l	0x44C	32	R/W	The lower 32 bits of base_address5 of DPU 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 DPU core2.
reg_dpu2_base_addr6_l	0x454	32	R/W	The lower 32 bits of base_address6 of DPU 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 DPU core2.
reg_dpu2_base_addr7_l	0x45C	32	R/W	The lower 32 bits of base_address7 of DPU 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 DPU core2.

Interrupts

The DPU generates an interrupt to signal the completion of a task. A high state on reg_dpu0_start signals the start of a DPU task for DPU core0. At the end of the task, the DPU 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 DPU cores. For example, when DPU core1 finishes a task while DPU core 0 is still working, reg_dpu_isr would contain `2'b10`.

The width of the `dpu_interrupt` signal is determined by the number of DPU cores. When the parameter DPU_NUM is set to 2, then the DPU IP contains two DPU cores, and the width of the dpu_interrupt signal is two. The lower bit represents the DPU core 0 interrupt and the higher bit represents the DPU core1 interrupt.

The interrupt connection between the DPU and the PS is described in the device tree file, which indicates the interrupt number of the DPU connected to the PS. Any interrupt pin may be used if the device tree file and Vivado assignments match. The reference connection is shown here.

Figure 8: **Reference Connection for DPU Interrupt**
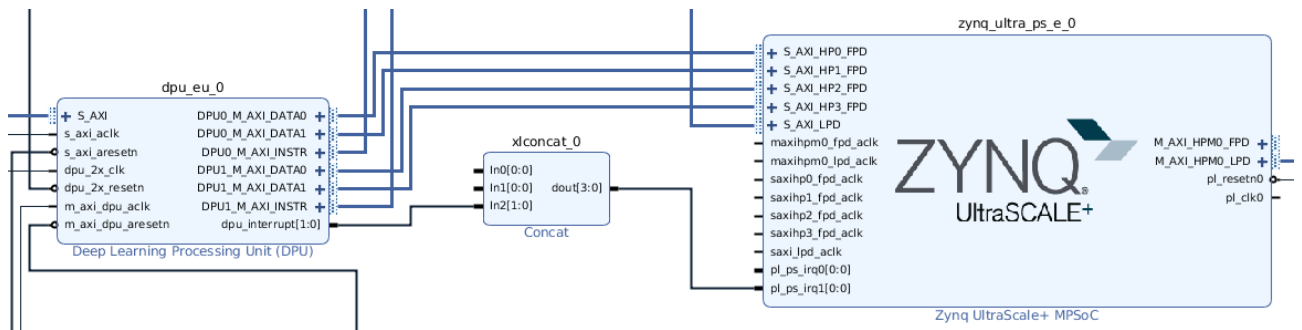
Note:

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 DPU 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 DPU cores that will be instantiated in a single DPU IP.

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

All the parameters will be described in the HWH file which is generated by the Vivado tool. The Vitis™ AI Compiler creates the matched models basing on the HWH. For more information, see the *Vitis AI User Guide* in the *Vitis AI User Documentation* (UG1431).

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

Features	Description	
Convolution	Kernel Sizes	W: 1–16 H: 1–16
	Strides	W: 1–4 H:1–4
	Padding_w	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, LeakyReLU, or ReLU6
	Dilation	dilation * input_channel ≤ 256 * channel_parallel && stride_w == 1 && stride_h == 1

Send Feedback

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

| Features	Description	
Depthwise Convolution	Kernel Sizes	W: 1–16 H: 1–16
	Strides	W: 1–4 H:1–4
	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 or ReLU6
	Dilation	dilation * input_channel ≤ 256 * channel_parallel && stride_w == 1 && stride_h == 1
Deconvolution	Kernel Sizes	W: 1–16 H: 1–16
	Stride_w	stride_w * output_channel ≤ 256 * channel_parallel
	Stride_h	Arbitrary
	Padding_w	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 or LeakyReLU
Max Pooling	Kernel Sizes	W: 1–8 H: 1–8
	Strides	W: 1–4 H:1–4
	Padding	W: 0–4 H: 0–4
Elementwise-sum	Input channel	1–256 * channel_parallel
	Input size	Arbitrary
Concat	Output channel	1–256 * channel_parallel
Reorg	Strides	stride * stride * input_channel ≤ 256 * channel_parallel
FC	Input_channel	Input_channel ≤ 2048 * channel_parallel
	Output_channel	Arbitrary

Notes:

1. The parameter channel_parallel is determined by the DPU 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. In this circumstance, the input feature map size cannot exceed the limitation of the kernel size of CONV, otherwise the Vitis AI will not be compiled correctly.

   This limitation occurs only in the Flatten+FC situation. This will be optimized in future versions.

Related Information

Configuration Options

Configuration Options

The DPU can be configured with some predefined options, which includes the number of DPU 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 DPU.

Figure 9: **DPU Configuration – Arch Tab**
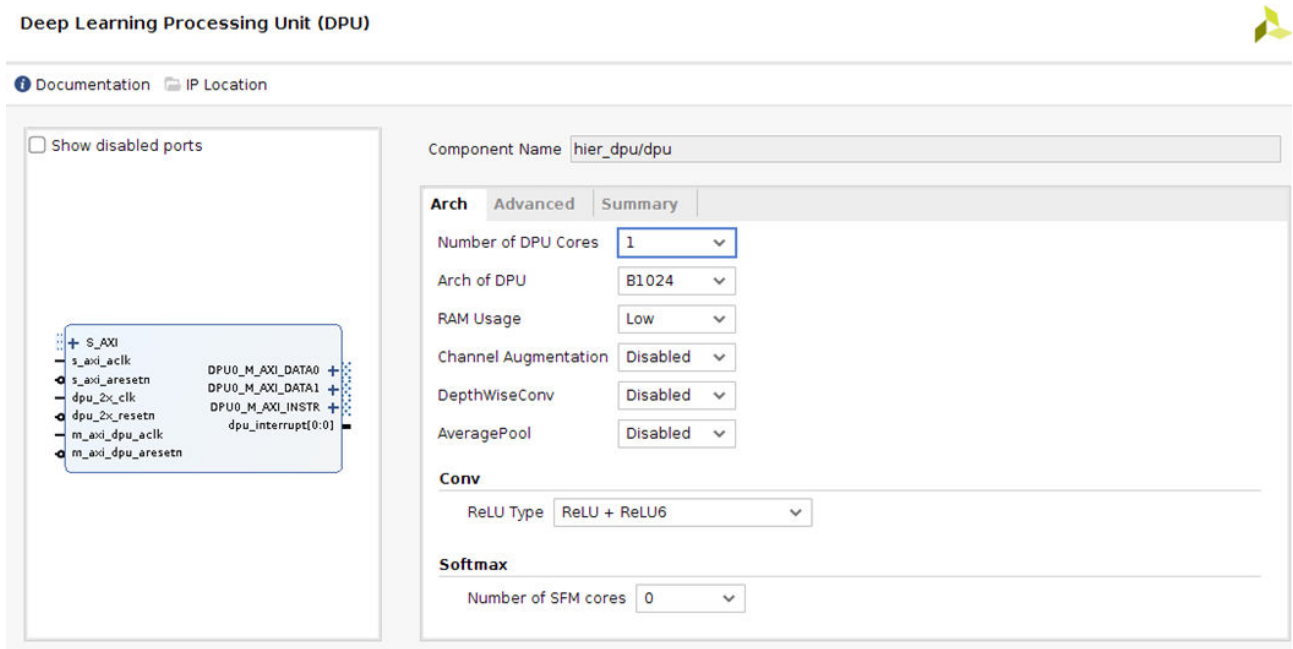
- **Number of DPU Cores:** A maximum of four cores can be selected in one DPU IP. Multiple DPU cores can be used to achieve higher performance. Consequently, it consumes more programmable logic resources.

  Contact your local Xilinx sales representative if more than four cores are required.

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

  There are three dimensions of parallelism in the DPU 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 previous table). 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 table.

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

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

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.

- **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 DPU architectures, and the setting is for all the DPU cores in the DPU IP. High RAM Usage means that the on-chip memory block will be larger, allowing the DPU more flexibility to handle the intermediate data. High RAM Usage implies higher performance in each DPU core. The number of BRAM36K blocks used in different architectures between low and high RAM Usage is illustrated in the following table.

Note: The DPU instruction set for different options of RAM Usage is different. When the RAM Usage option is modified, the DPU instructions file should be regenerated accordingly. The results are based on DPU with depthwise convolution.

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

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

- **Channel Augmentation:** Channel augmentation is an optional feature for improving the efficiency of the DPU when handling input channels much lower than the available channel parallelism. For example, the input channel of the first layer in most CNNs is three, which does not fully utilize all the available hardware channels. However, when the number of input channels is larger than the channel parallelism, then enabling channel augmentation will not make a difference.

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

Table 10: **Extra LUTs of DPU with Channel Augmentation**

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

- **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 10: **Depthwise Convolution and Pointwise Convolution**

X23133-081919

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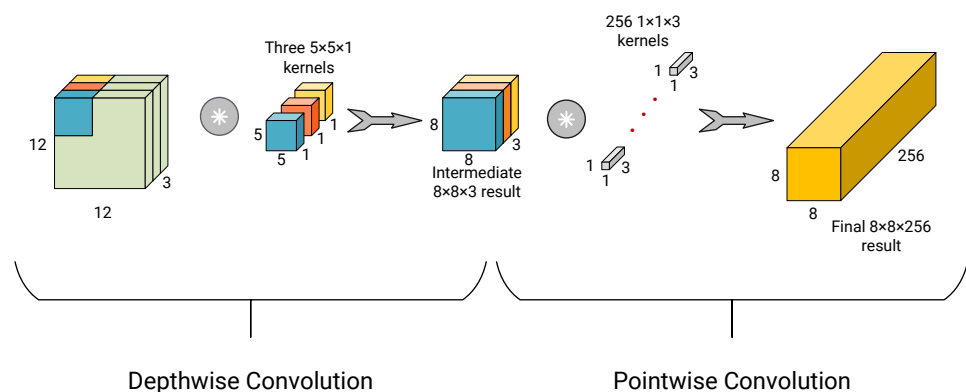
- **AveragePool:** The AveragePool option determines whether the average pooling operation will be performed on the DPU or not. The supported sizes range from 2x2, 3x3, ..., to 8x8, with only square sizes supported.

- **ReLU Type:** The ReLU Type option determines which kind of ReLU function can be used in the DPU. ReLU and ReLU6 are supported by default.

   The option "ReLU + LeakyReLU + ReLU6" means that the DPU can also include limitations on the allowed coefficients for LeakyReLU as an activation function.

   Note: LeakyRelU coefficient is fixed to 0.1.

- **Softmax:** This option allows the softmax function to be implemented in hardware. The hardware implementation of softmax can be 160 times faster than a software implementation. The hardware softmax module takes approximately 10000 LUTs, four block RAMs, and 14 DSPs. Enabling this option depends on the available hardware resources and desired throughput.

   When softmax is enabled, an AXI master interface named SFM_M_AXI and an interrupt port named `sfm_interrupt` will appear in the DPU IP. The softmax module uses `m_axi_dpu_aclk` as the AXI clock for SFM_M_AXI as well as for computation. The softmax function is not supported in Zynq®-7000 devices.

Advanced Tab

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

XILINX

Figure 11: **DPU Configuration – Advanced Tab**
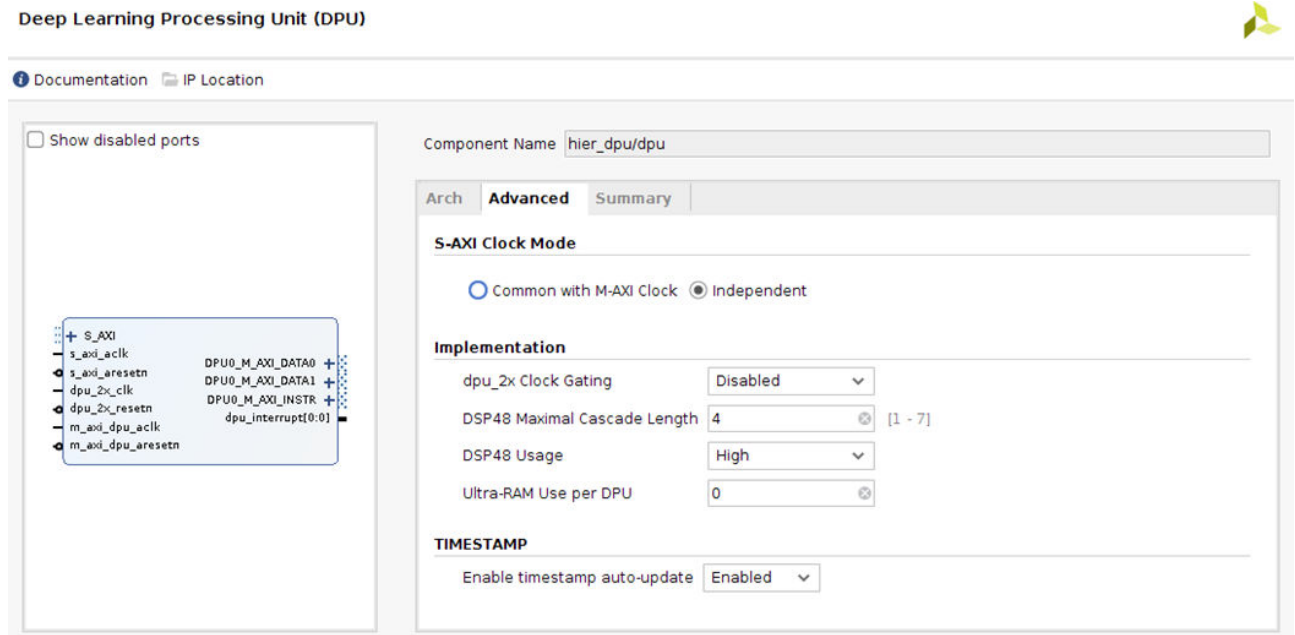
- **S-AXI Clock Mode:** `s_axi_aclk` is the S-AXI interface clock. When **Common with M-AXI Clock** is selected, `s_axi_aclk` shares the same clock as `m_axi_aclk` and the `s_axi_aclk` port is hidden. When **Independent** is selected, a clock different from `m_axi_aclk` is provided.

- **dpu_2x Clock Gating:** `dpu_2x` clock gating is an option for reducing the power consumption of the DPU. When the option is enabled, a port named `dpu_2x_clk_ce` appears for each DPU 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 DPU 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. Note that `dpu_2x` clock gating is not supported in Zynq®-7000 devices.

- **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 DPU convolution module. When low DSP usage is selected, the DPU IP will use DSP slices only for multiplication in the convolution. In high DSP usage mode, the DSP slice will be used for both multiplication and accumulation. Thus, the high DSP usage consumes more DSP slices and less LUTs. The logic utilization for high and low DSP usage is shown in the following table. The data is based on the DPU in the Xilinx ZCU102 platform without Depthwise Convolution, Average Pooling, Channel Augmentation, and Leaky ReLU features.

Send Feedback

Note: DSP Cascade is not supported in Zynq-7000 devices and it is locked to 1.

Table 11: **Resources for Different DSP Usage**

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

- **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 block consists of two block RAM 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 DPU 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 8, each memory unit in the DPU needs at least two UltraRAM blocks.

  The DPU 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 DPU. An example of block RAM and UltraRAM utilization is shown in the Summary tab section.

- **Timestamp:** When enabled, the DPU records the time that the DPU project was synthesized. When disabled, the timestamp keeps the value at the moment of the last IP update. The timestamp information can be obtained using the Vitis™ AI tools.

  Note: Most of the DPU configuration settings can be accessed by the Vitis AI tools. The following figure shows the information read by the Vitis AI tools.

Figure 12: **Timestamp Example**

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 DPU instruction set version number.

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

Send Feedback

DPU Performance on Different Devices

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

Table 12: **DPU Performance (GOPs) on Different Devices**

Device	DPU Configuration	Frequency (MHz)	Peak Theoretical Performance (GOPs)
Z7020	B1152x1	200	230
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.

Note: All the accuracy is based on an 8-bit-fixed quantization.

Table 13: **Performance of Different Models**

Network Model	Workload (GOPs per image)	Input Image Resolution	Accuracy (DPU)	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 Xilinx pruning tool.

I/O Bandwidth Requirements

When different neural networks run in the DPU, the I/O bandwidth requirement will change depending on which neural network is currently being executed. Even the I/O bandwidth requirement of different layers in one neural network are different. The I/O bandwidth requirements for some neural networks, averaged by layer, have been tested with one DPU 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 DPU architectures (B1152 and B4096).

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

Table 14: **I/O Bandwidth Requirements for DPU-B1152 and DPU-B4096**

Network Model	DPU-B1152		DPU-B4096	
	Peak (MB/s)	Average (MB/s)	Peak (MB/s)	Average (MB/s)
Inception-v1	1704	890	4626	2474
ResNet50	2052	1017	5298	3132
SSD ADAS VEHICLE	1516	684	5724	2049
YOLO-V3-VOC	2076	986	6453	3290

If one DPU 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 and Resets

Introduction

There are three clock domains in the DPU 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 DPU**

X22334-072219
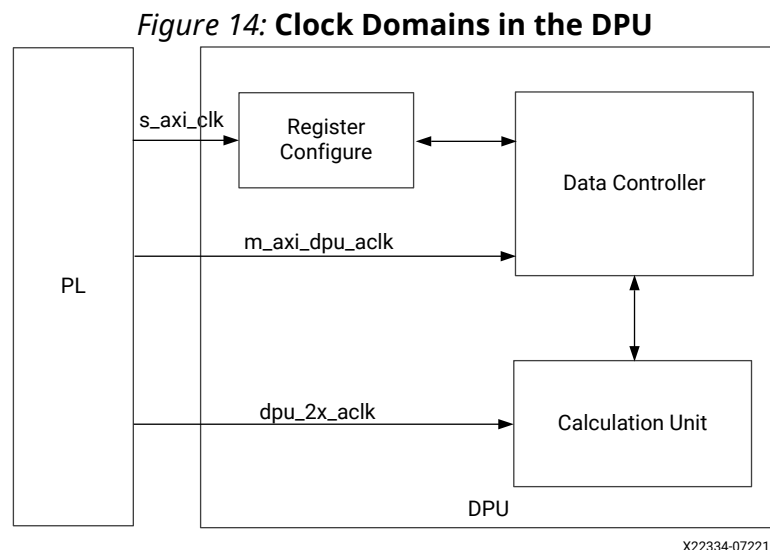
Register Clock

`s_axi_aclk` is used for the register configuration module. This module receives the DPU 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 DPU 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.

Send Feedback

Data Controller Clock

The primary function of the data controller module is to schedule the data flow in the DPU IP. The data controller module works with `m_axi_dpu_aclk`. The data transfer between the DPU 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 DPU 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 DPU 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**
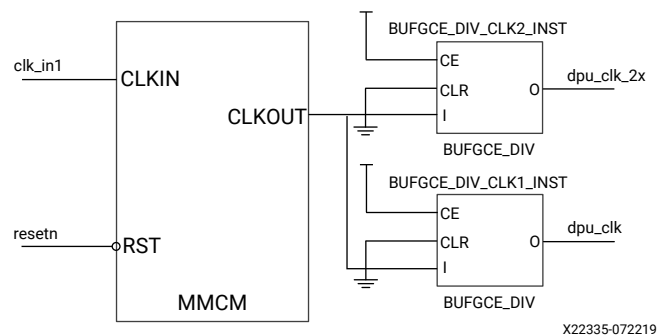
An MMCM and two BUFGCE_DIV blocks can be instantiated to design this circuit. The frequency of `clk_in1` is arbitrary and the frequency of output clock CLKOUT in the MMCM should be the frequency of `dpu_clk_2x`. BUFGCE_DIV_CLK1_INST divides the frequency of CLKOUT by two. `dpu_clk` and `dpu_clk_2x` are 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**
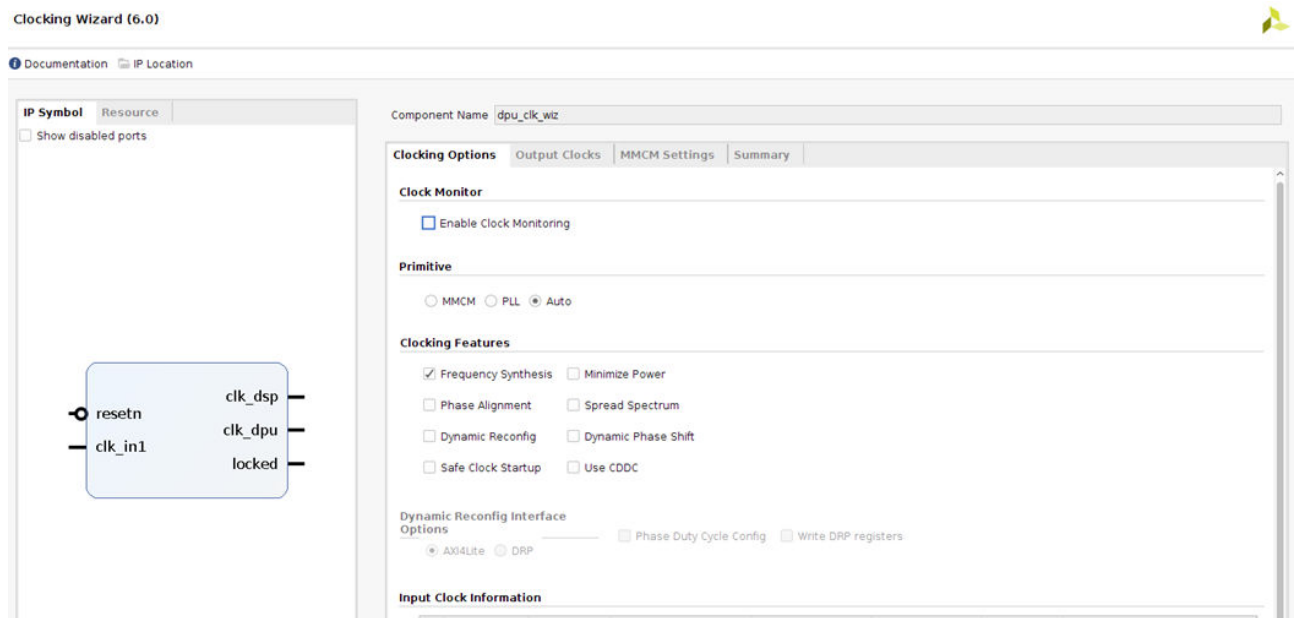
In addition, the Matched Routing must be selected for `m_axi_dpu_aclk` and `dpu_2x_clk` in the Output Clocks tab of the Clock Wizard IP. When the Matched Routing setting enables the two clocks that are both generated through a BUFGCE_DIV, the skew between the two clocks decreases significantly. The related configuration is shown in the following figure.

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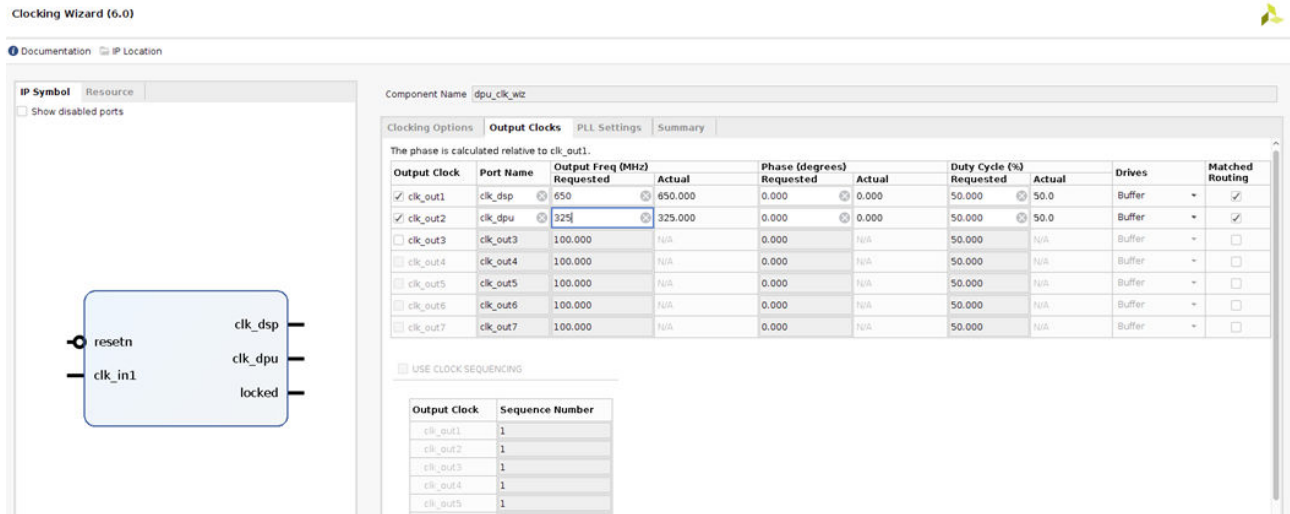

Add CE for dpu_2x_clk

The `dpu_2x` clock gating option can reduce the power consumption of the DPU. When the option is enabled, the number of generated `clk_dsp` should be equal to the number of DPU 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 DPU.

Figure 18: **Configure Clock Wizard with Buffer CE**
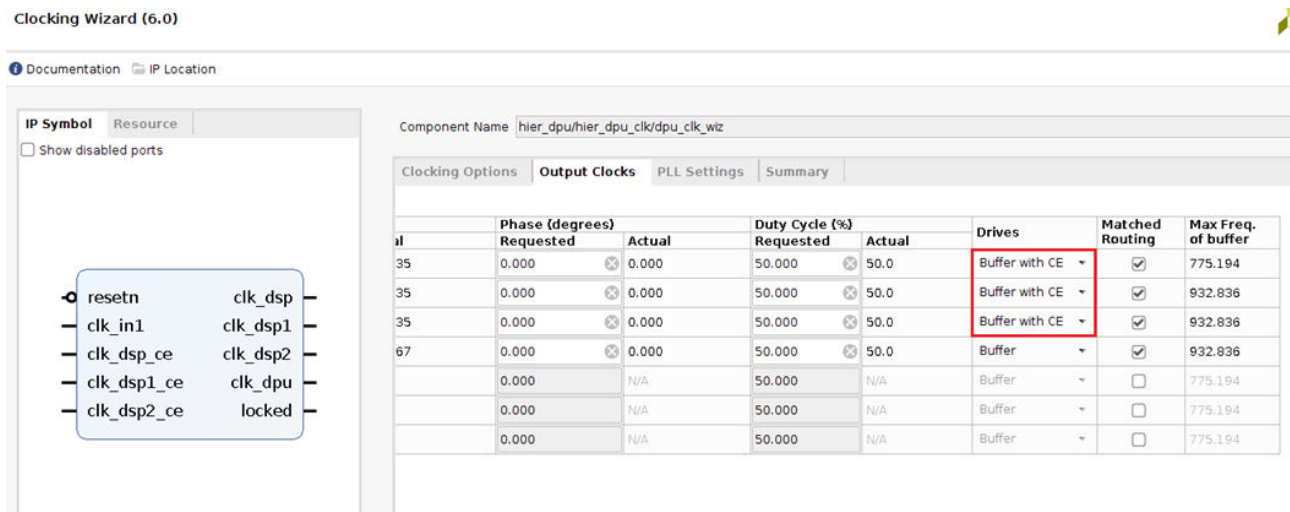
After configuring the clock wizard, the `clock_dsp_ce` should be connected to the corresponding port in the DPU. The connections are shown in the following figure.

Figure 19: **Clock CE and DPU Connections**
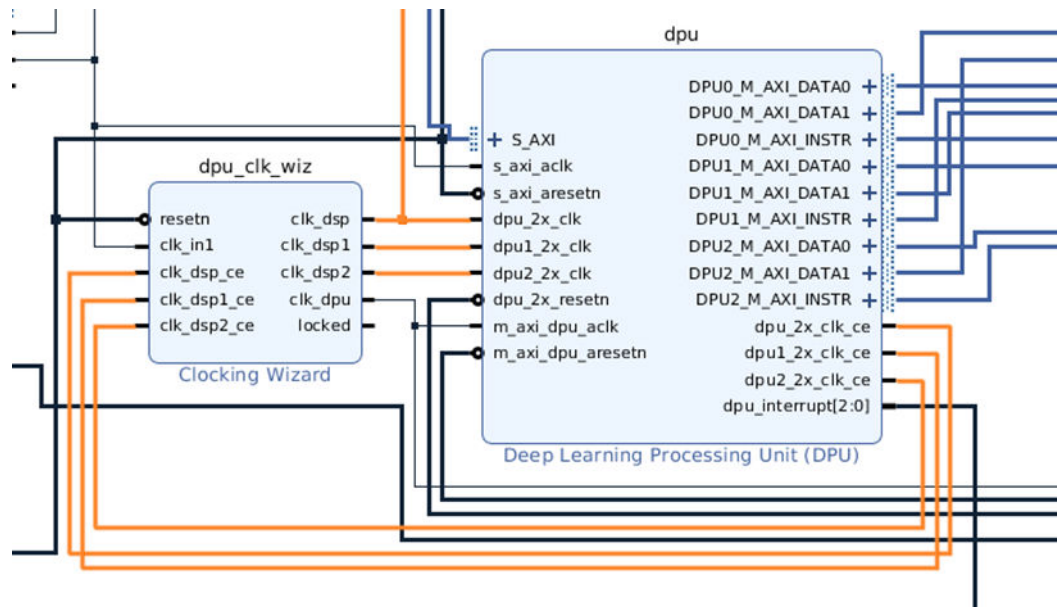
Reset

There are three input clocks for the DPU 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 DPU might not work properly. A Processor System Reset IP block is recommended to generate a synchronized reset signal. The reference design is shown here.

Send Feedback

Figure 20: **Reference Design for Resets**

Send Feedback
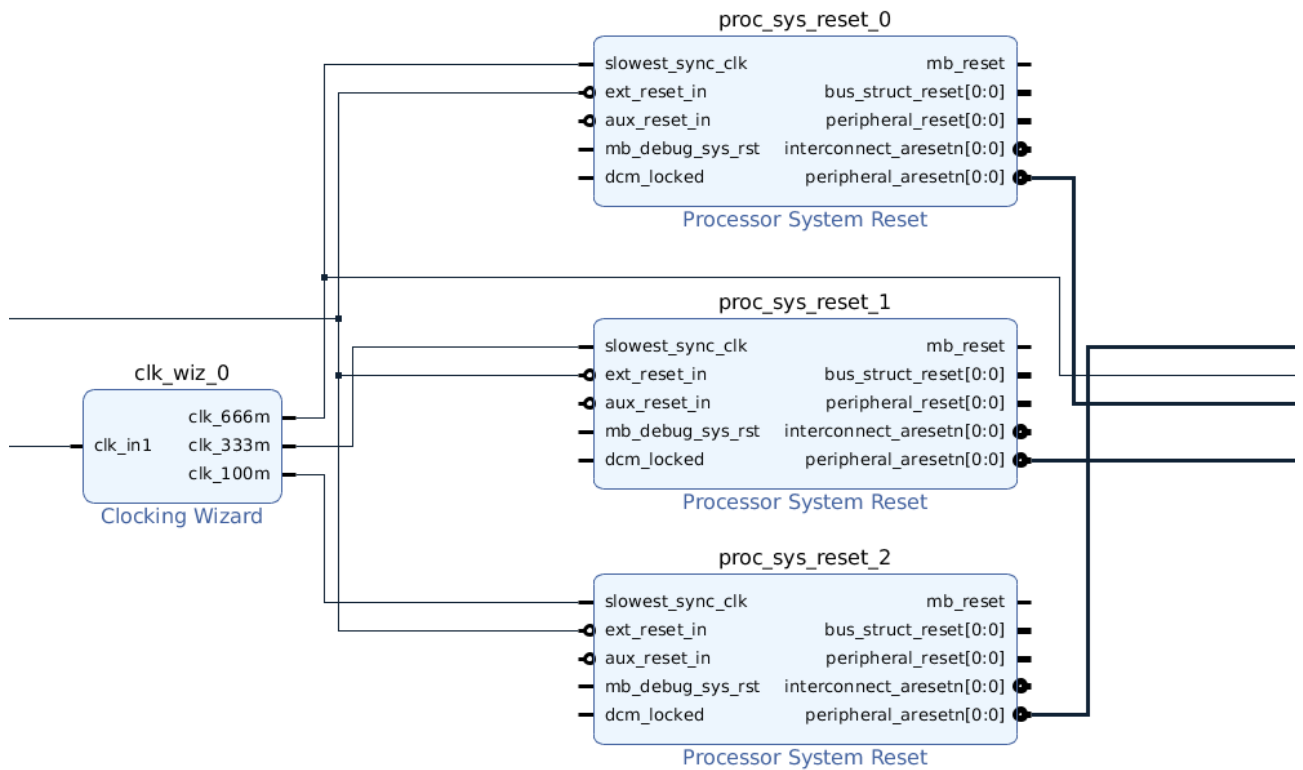
# Development Flow

Customizing and Generating the Core in MPSoC

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

- Add DPU IP into Repository or Upgrade DPU from a Previous Version
- Add DPU IP into Block Design
- Configure DPU Parameters
- Connecting a DPU with a Processing System in the Xilinx SoC
- Assign Register Address for DPU
- Generate Bitstream
- Generate BOOT.BIN
- Device Tree

Add DPU IP into Repository or Upgrade DPU from a Previous Version
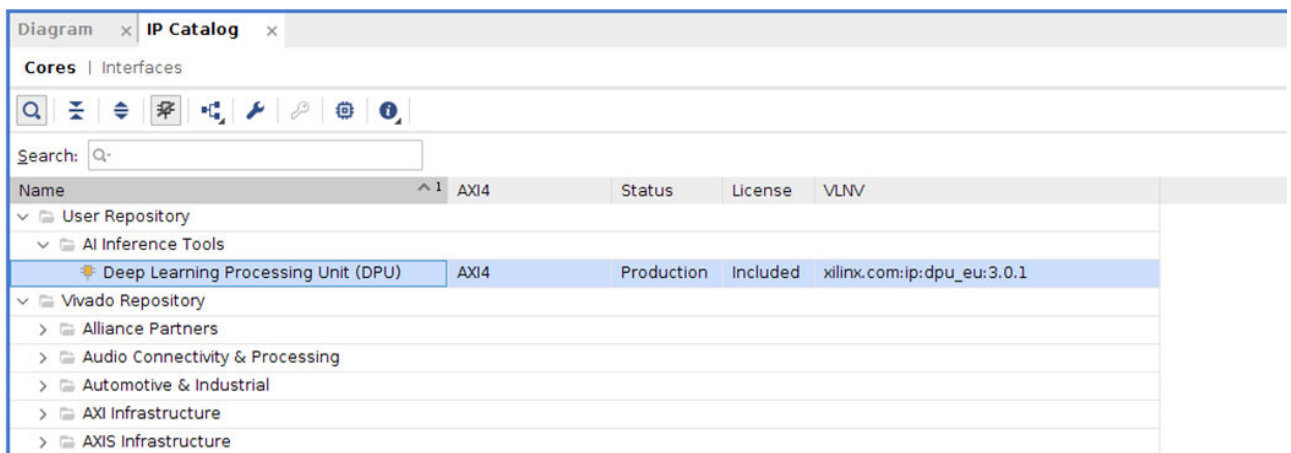
In the Vivado Integrated Design Environment (IDE), click **Project Manager → IP Catalog**. In the IP Catalog tab, right-click and select **Add Repository** (see figure below), then select the location of the DPU IP.

Figure 21: **Add Repository**
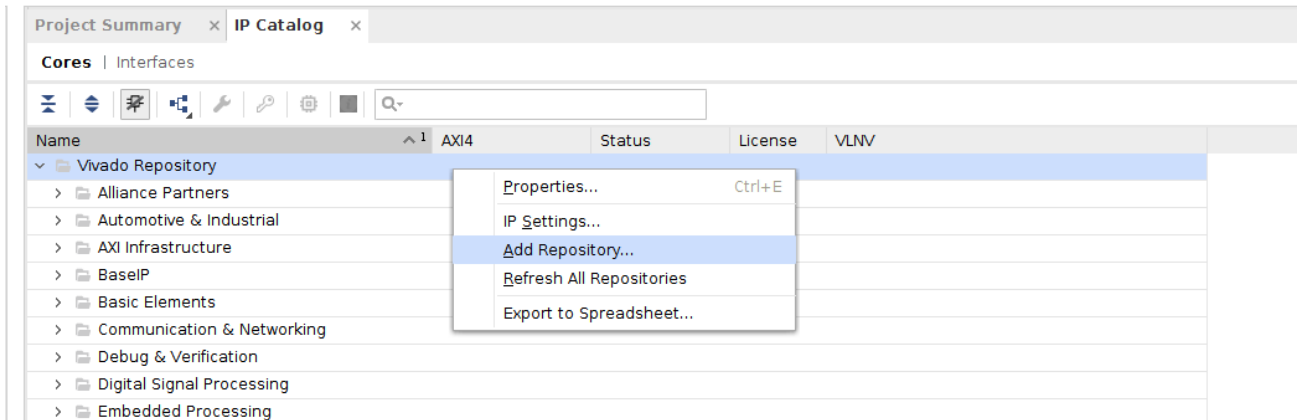
The DPU IP will appear in the IP Catalog page.

Figure 22: **DPU IP in Repository**

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

1. Delete the old DPU IP in the block design and IP repository.

2. Add the new DPU IP into the IP repository.

3. Add the new DPU IP into the block design.

Add DPU IP into Block Design

Search for DPU in the block design interface and add the DPU IP into the block design. The procedure is shown in the following figures.
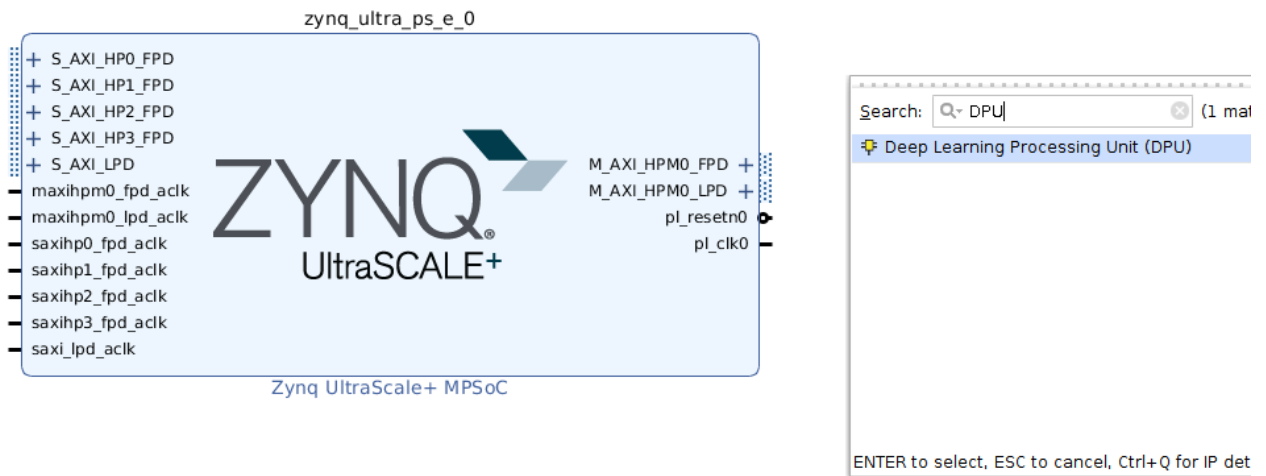
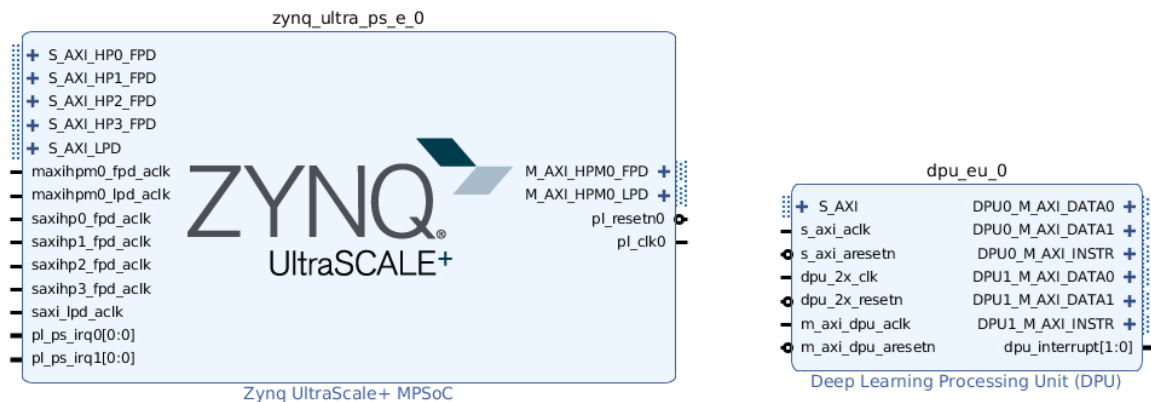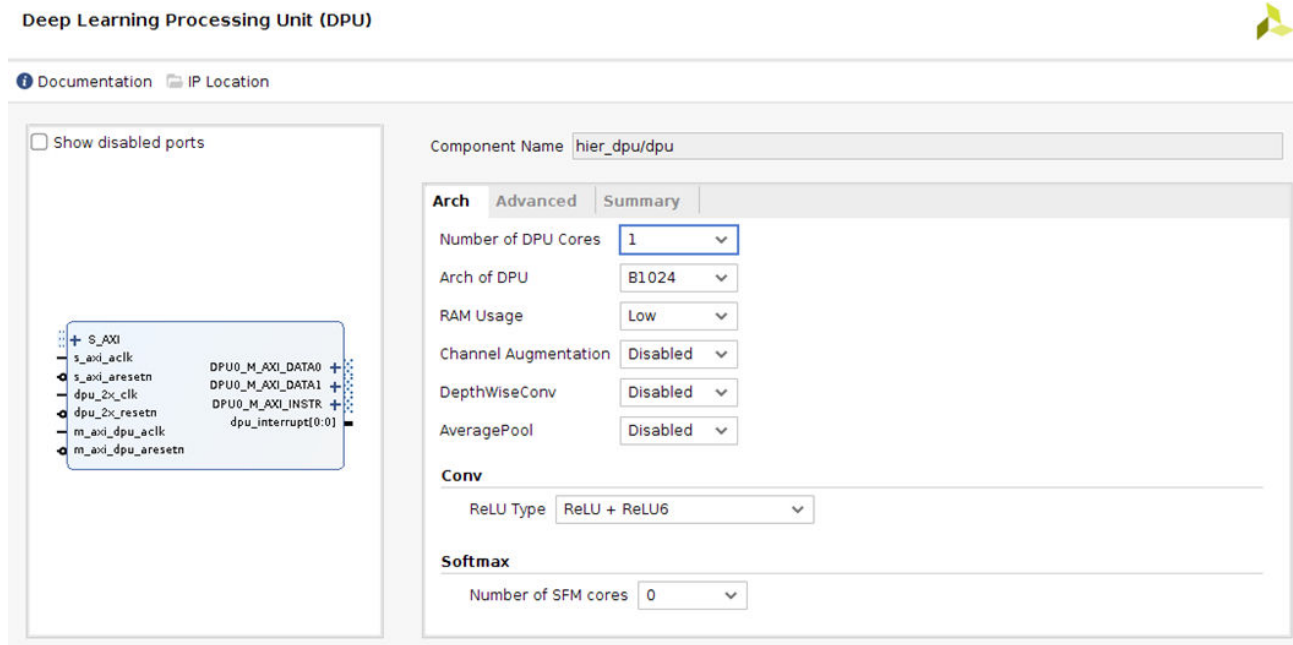*Figure 23:* **Search DPU IP**

Figure 24: **DPU IP into Block Design**

Configure DPU Parameters

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

Send Feedback

Figure 25: **Configure DPU**

Related Information
DPU Configuration

Connecting a DPU with a Processing System in the Xilinx SoC

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

The DPU IP can be connected to the processing system (PS) with an AXI Interconnection IP as long as the DPU 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 DPU performance. Therefore, Xilinx recommends that each master interface in the DPU is connected to the PS through a direct connection rather than through an AXI Interconnect IP when the AXI slave ports of the PS are enough.

When the AXI slave ports of the PS are insufficient for the DPU, 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 DPU 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 DPU 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

Xilinx recommends that the slave port of DPU be connected to M_AXI_HPM0_LPD of the PS.

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

Figure 26: **DPU and PS Connections for MPSoC**

Send Feedback
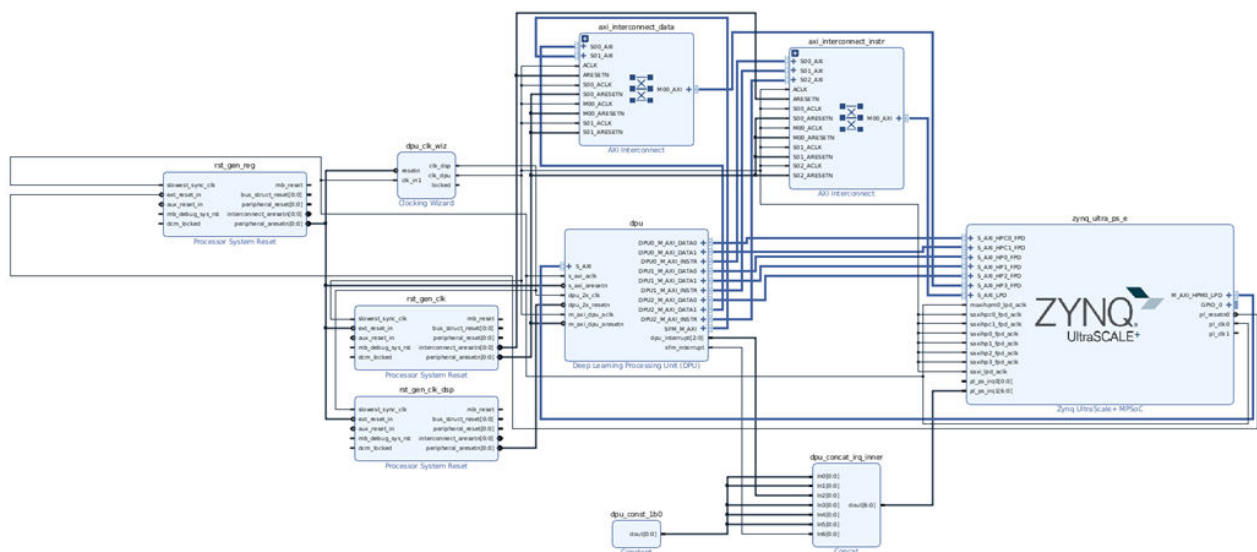
Assign Register Address for DPU

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

Note: The DPU 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 DPU are shown here.

Figure 27: **DPU Address Assignment**

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 28: **Generate Bitstream**

Send Feedback
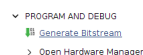
Generate BOOT.BIN

Vivado® Design Suite or PetaLinux can be used to generate the `BOOT.BIN` file. For boot image creation 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 DPU device needs to be configured correctly under the PetaLinux device tree so that the DPU driver can work properly. Create a new node for the DPU and place it as the child node of "amba" in the device tree `system-user.dtsi`, which is located under `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`. The parameters to the DPU and Softmax node are listed and described in the following table.

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>;
             ....
}
```

The parameters are described in the following table.

Table 15: **Device Tree Fields**

Parameter	Description
dpu	Node entry for DPU device. This does not need to be modified.
dpu->compatible	Fixed value set to "xilinx,dpu".
dpu->base-addr	DPU base register address assigned in the hardware design.
dpucore->compatible	Fixed value set to "xilinx,dpucore".
dpucore->interrupt-parent	Point to interrupt control device. ***Note***: "intc" for Zynq-7000 devices and "gic" for Zynq UltraScale+ devices.

Send Feedback

Table 15: **Device Tree Fields** *(cont'd)*

Parameter	Description
dpucore->interrupts	Interrupt configuration for the DPU IP cores. There are three fields for each DPU core, and the second value in each field corresponds to the interrupt number. The interrupt numbers must match the hardware configuration. For the above sample, the triplet "0x0 106 0x1" is for DPU core 0 with interrupt number 106, and the triplet "0x0 107 0x1" is for DPU core 1 with interrupt number 107. The other two values in the triplet "0x0" and "0x1" are fixed values and do not need to be changed.
dpucore->core-num	Number of DPU cores specified in the hardware configuration.
softmax->compatible	Fixed value set to "xilinx, smfc".
softmax->interrupt-parent	Point to interrupt control device. ***Note***: "intc" for Zynq-7000 devices and "gic" for Zynq UltraScale+ MPSoC devices.
softmax->interrupts	Interrupt configuration for the Softmax in DPU. The second value in this field corresponds to the interrupt number. The interrupt numbers must match the hardware configuration. For the above sample, the triplet "0x0 110 0x1" is for the Softmax with interrupt number 110. The other two values in the triplet "0x0" and "0x1" are fixed values and do not need to be changed.
softmax ->core-num	This value is fixed to "0x1" if softmax is added to the project in the hardware configuration.

The DPU description in the device tree should always be consistent with the configuration in the DPU hardware project, especially the interrupts. When the interrupts have been changed in the DPU project, the description in the device tree should be modified accordingly.
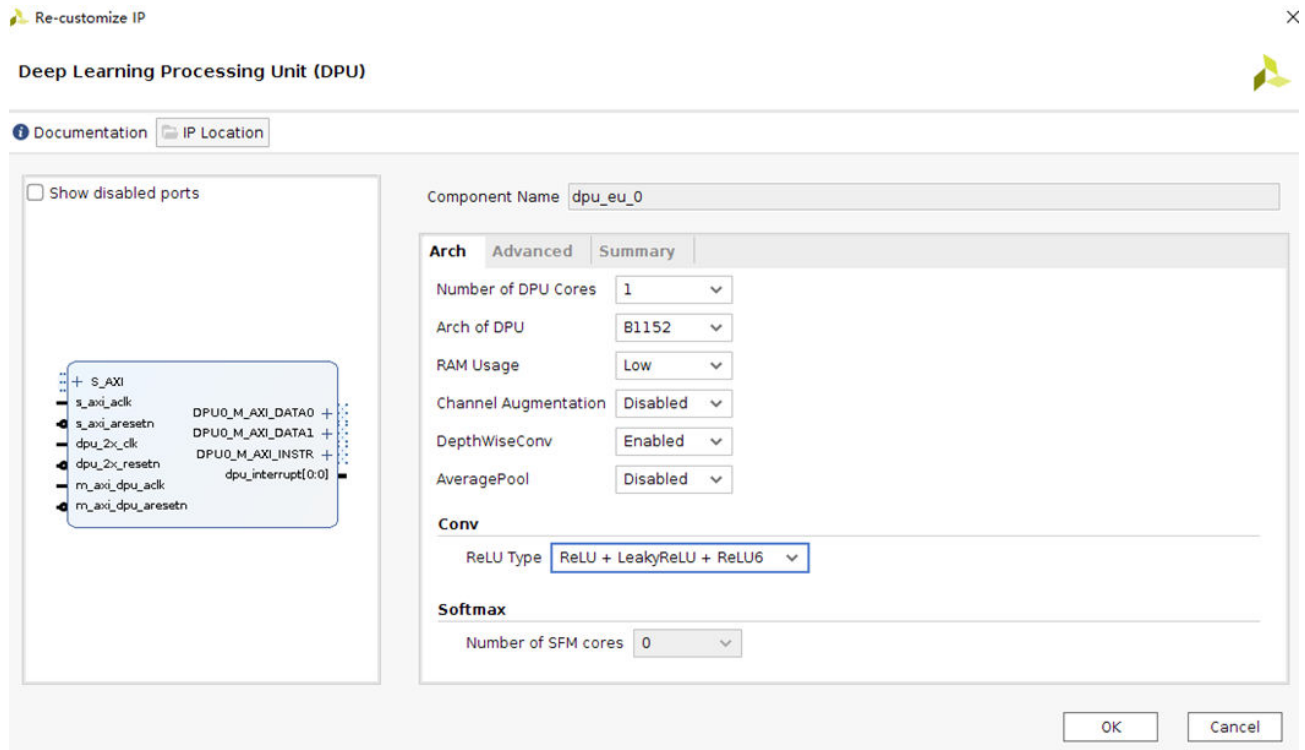
Customizing and Generating the Core in Zynq-7000 Devices

The latest DPU can be integrated into a Zynq®-7000 device project with some limitations:

1. When integrating the DPU IP into a Zynq-7000 device project, a new Vivado project must be created with the target device selected as a Zynq-7000 part. Simply changing the target device of an existing Vivado project with a DPU from a Zynq® UltraScale+™ MPSoC to a Zynq-7000 will not work.

2. The hardware softmax module is not supported in Zynq-7000 devices. The option of softmax cores is set as 0 and cannot be changed. This may change in a future release.

3. The maximum data width of an AXI port in the processing system (PS) of the Zynq-7000 is 64 bits. The data width of the DPU will be modified from 128 bits to 64 bits. When the data width of the AXI interface is changed, the instruction file must be regenerated by Vitis AI accordingly.

The default configuration for the DPU in Zynq-7000 devices is shown below:

Figure 29: **DPU Configuration in Zynq-7000 Devices**

Cutomizing and Generating the Core in Vitis Integrated Design Environment (IDE)
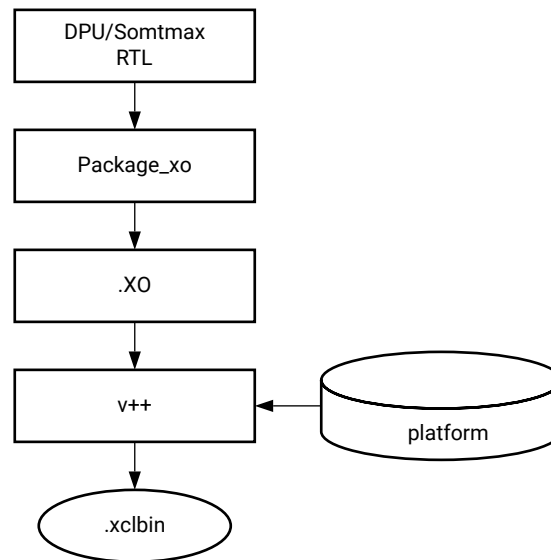
The Vitis™ embedded software development flow has the following two methods to generate the core:

1. GUI Flow

2. Command Flow

It is recommended to use the command flow, as it is more convenient and flexible. If you want to use the GUI flow, see the tutorial here.

The DPU build process is shown in the following figure:

Send Feedback

Figure 30: **DPU Build Process**

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

- **Makefile and other scripts:** The DPU 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 environment can be used to simplify this process. The `.xo` file are 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 DPU 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.

  All the DPU parameters are configured in the `dpu_conf.vh` file. You can configure the DPU by different definition.

- **Architecture:** The DPU can configure the hardware architecture, including: B512, B800, B1024, B1600, B2304, B3136, and B4096. For the B4096, the definition is as follows:

  ```
  `define B4096
  ```

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

  When the UltraRAM number is enabled, set the UltraRAM number as the following,

  - `define def_UBANK_IMG_N` - 4

Send Feedback

- `define def_UBANK_WGT_N` - **17**

- `define def_UBANK_BIAS` - **1**

There are some recommended UltraRAM numbers for different architectures. You can also adjust the numbers according to the resource usage of the entire project.

Table 16: **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 DWXC_ENABLE ``

  Disable - `` `define DWCV_DSIABLE ``

- **AveragePool:** Enable - `` `define POOL_AVG_ENABLE ``

  Disable - `` `define POOL_AVG_DISABLE ``

- **RELU Type:** There are four options of RELU type, they are:

  1. RELU_RELU6
  2. RELU_LEAKRELU_RELU6

  If you want to use the RELU+LeakyRelu+ReLU6, define like the following:

  ```
  `define RELU_LEAKYRELU_RELU6
  ```

- **DSP Usage:** High - `` `define DSP48_USAGE_HIGH ``

  Low - `` `define DSP48_USAGE_LOW ``

- **Set the DPU Number:** The DPU core number, by default, is set to 1. Add the [connectivity] property to configure the DPU number as follows:

  ```
  [connectivity]
  ```

  ```
  nk=dpu_xrt_top:2
  ```

Send Feedback

The project will integrate two DPUs.

- **Specify Connectivity for DPU Ports:** Specify the connectivity to the various ports in the system for the DPU. Add the [connectivity] property to configure the DPU 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 DPU, then the ports can be shared.

Add the [connectivity] property to specify the DPU 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 will use the Performance_Explore strategy.

Send Feedback

Example Design

Introduction

The Xilinx® DPU targeted reference design (TRD) provides instructions on how to use the DPU 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. It can also be used for a Zynq®-7000 SoC platform using the same flow. 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

Vivado TRD Overview

The TRD creates an image classification application running a popular deep neural network model, Resnet50, on a Xilinx® Zynq® UltraScale+™ MPSoC. The overall functionality of the TRD is partitioned between the processing system (PS) and programmable logic (PL), where the DPU resides for optimal performance.

The following figure shows the TRD block diagram. The host communicates with the ZCU102 board through an Ethernet or UART port. The input images for the TRD are stored in an SD card. When the TRD is running, the input data is loaded into DDR memory, then the DPU reads the data from DDR memory and writes the results back to DDR memory. Results are shown on the host screen from the APU through Ethernet or UART.

Figure 32: **DPU TRD Overview**
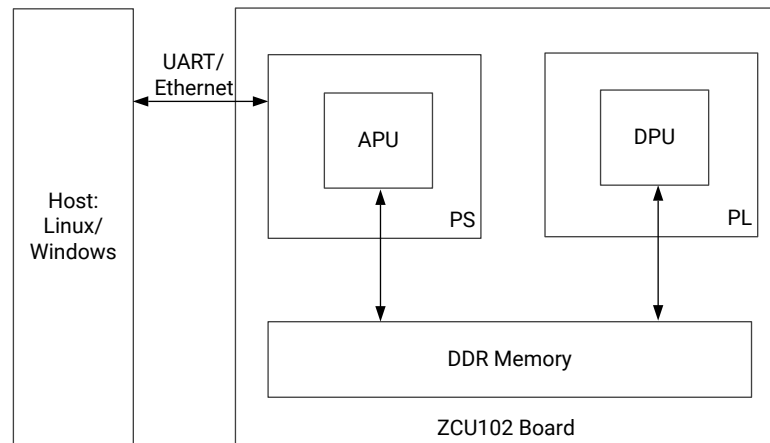
X23041-072519

Requirements

The following summarizes the requirements of the TRD.

Target platform:

- ZCU102 evaluation board, production silicon. See *ZCU102 Evaluation Board User Guide* (UG1182).

Xilinx tools:

- Vivado Design Suite 2019.2
- PetaLinux 2019.2
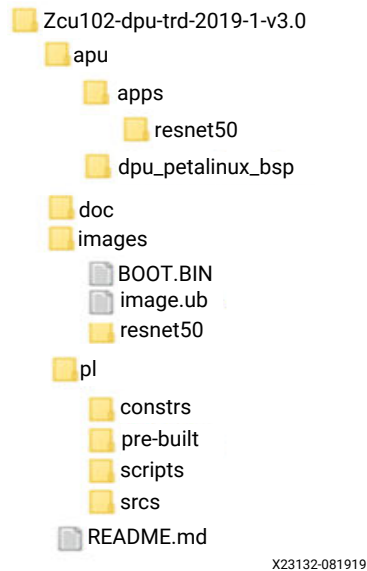
Hardware peripherals:

- SD card
- Ethernet
- UART

Linux or Windows host system:

- Serial terminal
- Network terminal

Send Feedback

Design Files

Design files are stored in the following directory structure.

Figure 33: **Directory Structure**


```
Zcu102-dpu-trd-2019-1-v3.0
    apu
        apps
            resnet50
        dpu_petalinux_bsp
    doc
    images
        BOOT.BIN
        image.ub
        resnet50
    pl
        constrs
        pre-built
        scripts
        srcs
    README.md
```
X23132-081919

*Note*: DPU_IP is in the `pl/srcs/dpu_ip/` directory.

Hardware Design

This section describes how to create the DPU reference design project using the Xilinx Vivado Design Suite and generate the bit file. The parameters of the DPU IP in the reference design are configured accordingly. Both the connections of the DPU interrupts and the addresses for the DPU in the reference design should not be modified. If those connections or address are modified, the reference design might not work properly.

Board Setup

The following figure shows the ZCU102 board with interfaces identified.

Send Feedback

Figure 34: **ZCU102 Board**
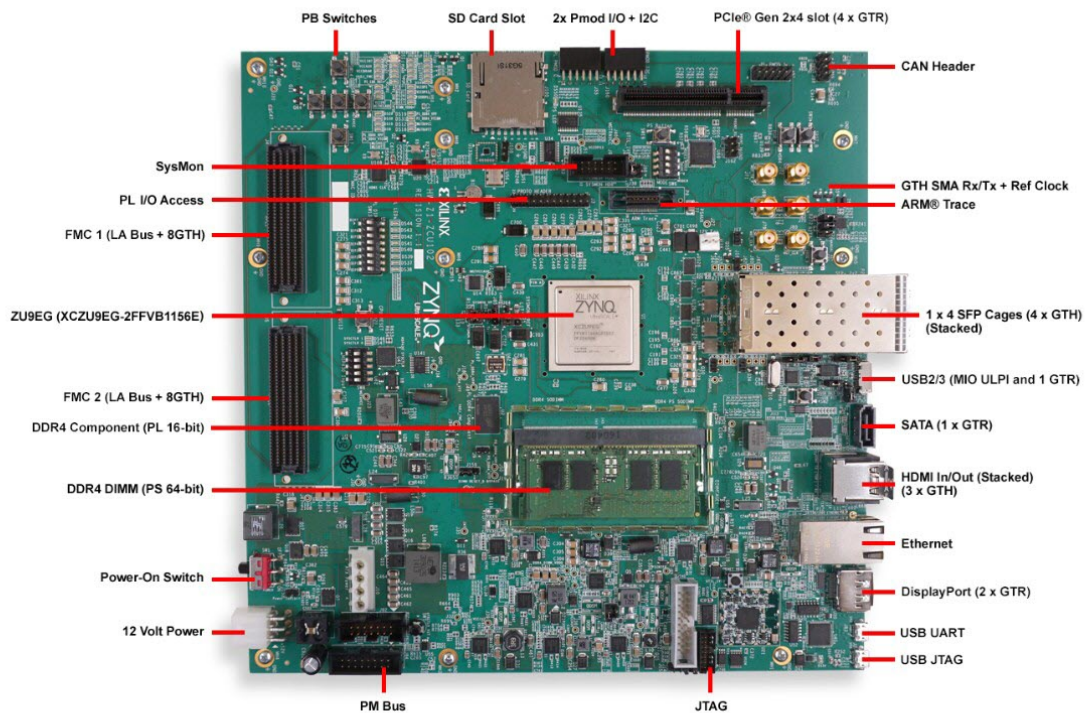
ZCU Board Configuration

1. Connect the Micro USB cable to the ZCU102 Board Micro USB UART (J83) port and the other end into an open USB port on the host PC. This cable will be used for UART over USB communication.

2. Insert the SD card burned with the TRD image file into the SD card slot.

3. Set the SW6 switches and configure the boot setting to boot from SD as shown here.

4. Connect 12V power to the ZCU102 6-Pin Molex connector.

5. Switch on SW1 to power on the ZCU102 board.

Project Build Flow

This section describes how to build the reference Vivado project with Vivado 2019.2. For information about setting up your Vivado environment, refer to the *Vivado Design Suite User Guide: Getting Started* (UG910).

Building the hardware design consists of the following steps:

Building the Hardware Design on Linux

1. Open a Linux terminal.

2. Change the directory to $TRD_HOME/pl.

3. Create the Vivado IP integrator project and invoke the GUI by running the following command:

```
% vivado -source scripts/trd_prj.tcl
```

**Building the Hardware Design on Windows**

1. Select **Start → All Programs → Xilinx Design Tools → Vivado 2019.2 → Vivado 2019.2** .

2. On the Quick Start screen, click **Tcl Console**.

3. Type the following command in the Tcl console:

```
cd $TRD_HOME/pl
source scripts/trd_prj.tcl
```

After running the scripts, the Vivado IP integrator block design appears as shown.
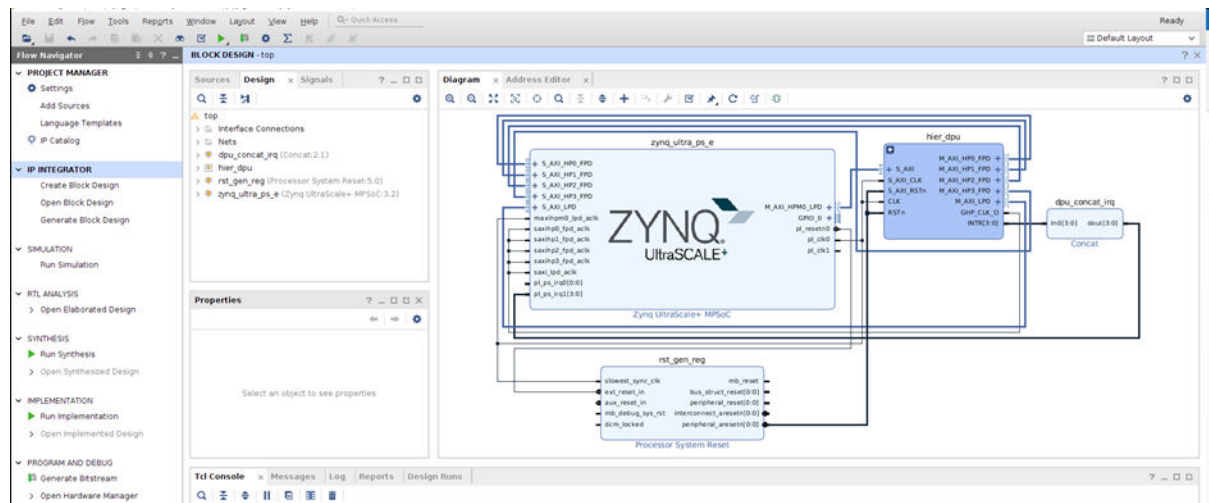
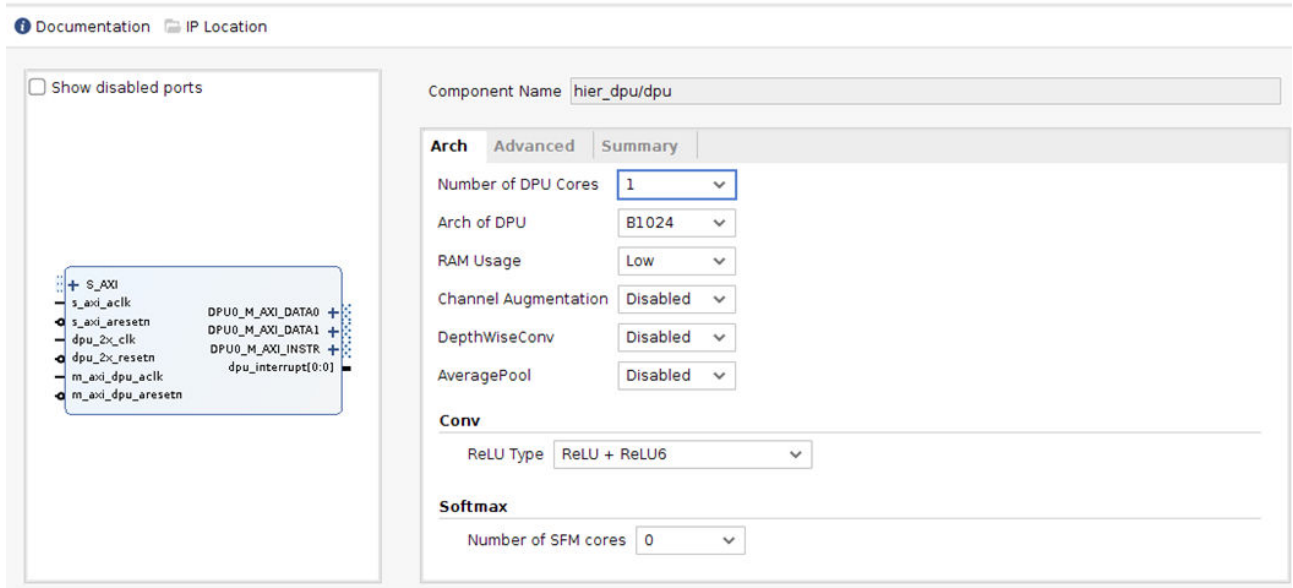4. In the GUI, click **Generate Bitstream** to generate the bit file as shown.
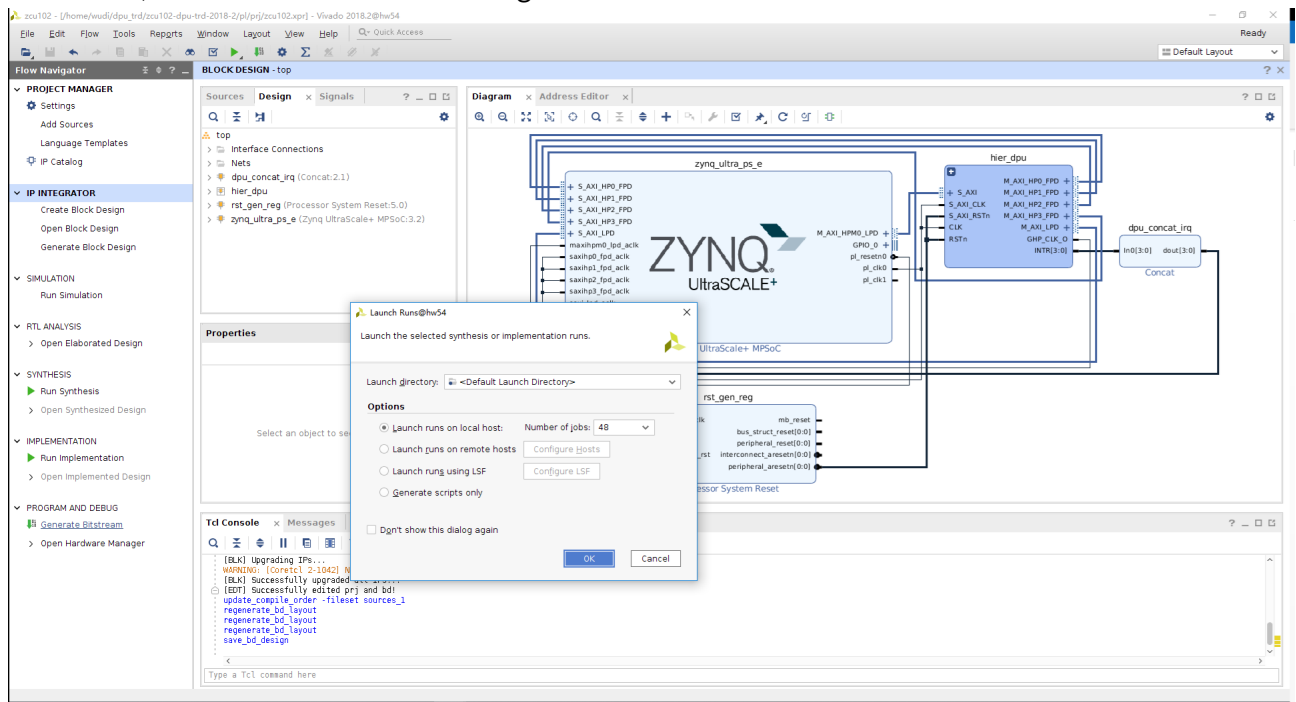
DPU Configuration

The version of the DPU IP integrated in the TRD is DPU_v3.2.

Figure 35: **DPU Configuration Page**

The DPU parameters can be configured in case of different resource requirements. For more information on the DPU parameters, see the DPU Configuration section.

Related Information
Configure DPU Parameters

Software Design

This section describes how to generate `BOOT.BIN` using the PetaLinux tools.

PetaLinux Design Flow

Install PetaLinux as described in the *PetaLinux Tools Documentation: Reference Guide* (UG1144).

Set PetaLinux Variable

Set the following PetaLinux environment variable $PETALINUX:

```
% source<path/to/petalinux-installer>/Petalinux-v2019.2/petalinux-v2019.2-
final/settings.sh
% echo $PETALINUX
% export TRD_HOME=<path/to/downloaded/zipfile>/zcu102-dpu-trd-2019-2
```

Build the PetaLinux Project

Use the following commands to create the PetaLinux project.

```
% cd $TRD_HOME/apu/dpu_petalinux_bsp% petalinux-create -t project -s xilinx-
dpu-trd-zcu102-v2019.2.bsp
% cd zcu102-dpu-trd-2019-2
% petalinux-config --get-hw-description=$TRD_HOME/pl/pre-built --
silentconfig
% petalinux-build
```

If the pre-built design is used, use the value of `--get-hw-description` as shown in the previous command.

If a new generated/modified design is needed, change `$TRD_HOME/pl/pre-built` to `$TRD_HOME/pl/prj/zcu102.sdk`.

Create BOOT.BIN

Use the following to create the `BOOT.BIN` file:

```
% cd images/linux
% petalinux-package --boot --fsbl zynqmp_fsbl.elf --u-boot u-boot.elf --
pmufw pmufw.elf --fpga system.bit
```

Build the Demo

This section describes how to build the resnet50 example from the source. The pre-built resnet50 under `$TRD_HOME/images` can be used to skip this step. The following example uses five threads to run the classification task. The DPU runtime will schedule cores automatically according to your hardware design.

1. First, extract the software environment file:

    ```
    % cd $TRD_HOME/apu/apps
    % ./sdk.sh -d ./sdk -y
    ```

    The pre-generated `sdk.sh` under `$TRD_HOME/apu/apps` can be used or use `petalinux-build -s` to generate a new `sdk.sh`. If a "permission is denied" error occurs when running `sdk.sh`, run `chmod 777 sdk.sh` to resolve. After the software environment file has been extracted, source the environment setup script each time you wish to build this demo in a new shell session.

2. Build the resnet50 example:

    ```
    % cd $TRD_HOME/apu/apps/resnet50
    % make
    ```

    The newly generated resnet50 is in the directory `$TRD_HOME/apu/apps/resnet50`.

Demo Execution

This section describes how to run the executables generated by the TRD. Connect to the ZCU102 board through UART.

Note: The login/password for the ZCU102 board is `root/root`. The HWH file is located at `$TRD_HOME/pl/srcs/top/hw_handoff/top.hwh`. See the *Vitis AI User Guide* in the *Vitis AI User Documentation* (UG1431) for more details..

To run the demo:

1. After generating the `BOOT.BIN` file, copy `BOOT.BIN` and `image.ub` (which is in `image/linux` folder) to the SD card.

2. Copy the resnet50 directory in `$TRD_HOME/images` to the SD card.

3. Use the pre-built resnet50 in `$TRD_HOME/images/resnet50` or copy the newly generated resnet50 in `$TRD_HOME/apu/apps/resnet50/build/` to the resnet50 directory on the SD card.

    A screenshot is shown in below.

The input image name is displayed in each line beginning with "Load image", the names are also the expected result for the input image. The predicted results of the DPU are shown below, and the top-5 prediction probabilities are printed. If the name of the loaded image is approximated by one of the Top-5 predictions in most of the results, the DPU is working properly.

```
root@zcu102-dpu-trd-v2018:/media/card/resnet50# ./resnet50

##################################################
Warning:
The DPU in this TRD can only work 8 hours each time!
Please consult Sales for more details about this!
##################################################

total image : 10

Load image: bird1.png
[Top 0] prob = 0.122421   name = marmoset,
[Top 1] prob = 0.122421   name = tiger cat,
[Top 2] prob = 0.095341   name = tiger, Panthera tigris,
[Top 3] prob = 0.074252   name = Persian cat,
[Top 4] prob = 0.057828   name = tabby, tabby cat,

Load image: automobile1.png
[Top 0] prob = 0.798495   name = moving van,
[Top 1] prob = 0.024112   name = minibus,
[Top 2] prob = 0.018779   name = police van, police wagon, paddy wagon, patrol wagon, wagon, black Maria,
[Top 3] prob = 0.011390   name = trailer truck, tractor trailer, trucking rig, rig, articulated lorry, semi,
[Top 4] prob = 0.011390   name = passenger car, coach, carriage,

Load image: deer1.png
[Top 0] prob = 0.325294   name = rotisserie,
[Top 1] prob = 0.153658   name = throne,
[Top 2] prob = 0.044024   name = barrel, cask,
[Top 3] prob = 0.034286   name = tobacco shop, tobacconist shop, tobacconist,
[Top 4] prob = 0.034286   name = safety pin,
```

```
Load image: horse1.png
[Top 0] prob = 0.651236   name = gazelle,
[Top 1] prob = 0.088135   name = hartebeest,
[Top 2] prob = 0.068640   name = impala, Aepyceros melampus,
[Top 3] prob = 0.053457   name = sorrel,
[Top 4] prob = 0.025251   name = llama,

Load image: ship1.png
[Top 0] prob = 0.623744   name = speedboat,
[Top 1] prob = 0.108390   name = yawl,
[Top 2] prob = 0.108390   name = catamaran,
[Top 3] prob = 0.039875   name = trimaran,
[Top 4] prob = 0.024185   name = lakeside, lakeshore,

Load image: truck4.png
[Top 0] prob = 0.814788   name = thresher, thrasher, threshing machine,
[Top 1] prob = 0.066882   name = moving van,
[Top 2] prob = 0.052088   name = trailer truck, tractor trailer, trucking rig, rig, articulated lorry, semi,
[Top 3] prob = 0.014923   name = tractor,
[Top 4] prob = 0.005490   name = tow truck, tow car, wrecker,

[Time]87005us
[FPS]114.936

##################################################
Warning:
The DPU in this TRD can only work 8 hours each time!
Please consult Sales for more details about this!
##################################################
```

Send Feedback

Vitis TRD Flow

Following are the requirements of the Vitis™ TRD flow:

1. Target platforms: ZCU102 evaluation board, production silicon - For more information, see *ZCU102 Evaluation Board User Guide* (UG1182).

2. Xilinx tools:

   - Vitis 2019.2

3. Hardware peripherals:

   - SD

   - Ethernet

   - UART

4. Linux or Windows host system:

   - Serial terminal

   - Network terminal

Hardware Design

The DPU and Softmax must be located in the `$TRD_HOME/dpu_ip` directory.

The Vitis Design Suite 2019.2 includes the zcu102_base platform. The following tutorials assumes that the Vitis and XRT environment variables are set as following:

1. Open a linux terminal and set the Linux as Bash mode.

```
% source <vitis install path>/Vitis/2019.2/settings64.sh
% source /opt/xilinx/xrt/setup.sh
```

The default setting of DPU is B4096, URAM_DISABLE, RAM_USAGE_LOW, CHANNEL_AUGENTATION_ENABLE, DWCV_ENABLE, POOL_AVG_ENABLE, RELU_LEAKYRELU_RELU6, DSP48_USAGE_LOW, MPSOC which defined in the `dpu_conf.vh` file.

Note: Add the location of platform to SDX_PLATFORM using the`$ export SDX_PLATFORM=<platform path>/zcu102_base/zcu102_base.xpfm`.

2. Build the hardware design

```
% cd $TRD_HOME/prj/Vitis
```

   - For building the DPU only:

```
% make KERNEL=DPU DEVICE=zcu102
```

Send Feedback

- For building the DPU and Softmax

```
% make KERNEL=DPU_SM DEVICE=zcu102
```

3. The generated SD card files will be placed in `$TRD_HOME/prj/Vitis/binary_container_1/sd_card`.

Software Design

This section describes how to run the Resnet50 example from the source code.

You can use the pre-built Resnet50 under `$TRD_HOME/app/Vitis`. The pre-built is based on the ZCU102 platform.

Demo Execution

This section describes how to run the executables generated by the TRD.

1. Connect to the ZCU102 board through UART.

2. To run, copy all the files from `$TRD_HOME/prj/Vitis/binary_container_1/sd_card` to the SD Card.

3. Copy all the files from `$TRD_HOME/app/Vitis` directory to the SD Card.

4. After the linux boot, run the following:

```
% cd $HOME
% cp sd_card/dpu.xclbin /usr/lib
% tar -xvf /mnt/app/Vitis/resnet50.tar.gz
% cp  /mnt/app/Vitis/models/resnet50.elf .
% env LD_LIBRARY_PATH=samples/lib samples/bin/resnet50 img/
bellpeppe-994958.JPEG
```

The resnet50 program will execute the image classification task.The result is shown in the following figure:

5. The HWH file has been copied to `$TRD_HOME/prj/Vitis/binary_container_1/sd_card`. Or you can get it from `./binary_container_1/link/vivado/vpl/prj/prj.srcs/sources_1/bd/<platform nam>/hw_handoff/<paltform name>.hwh"`

For more details on the HWH file, see the *Vitis AI User Guide* in the *Vitis AI User Documentation* (UG1431).

Send Feedback

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. *DNNDK User Guide* (UG1327)

2. *Vitis AI User Guide* in the *Vitis AI User Documentation* (UG1431)

3. *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209)

4. *PetaLinux Tools Documentation: Reference Guide* (UG1144)

5. *ZCU102 Evaluation Board User Guide* (UG1182)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
03/23/2020 Version 3.2	
Entire document	• Updated the Vivado flow and Vitis flow for DPU_v1.4.1. • Replaced the description of DNNDK and DNNC to Vitis AI Kit and Vitis AI Compiler. • Updated the maximum of DPU core number from three to four and modified the descriptions accordinly.
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.
Chapter 4: 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.
Chapter 5: Clocking and Resets	Updated whole chapter.
Add CE for dpu_2x_clk	Added section.
Chapter 6: Development Flow	Updated whole chapter.

Section	Revision Summary
Add DPU IP into Repository or Upgrade DPU from a Previous Version	Updated section.
Customizing and Generating the Core in Zynq-7000 Devices	Updated figure.
Chapter 7: 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.
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.
Add CE for dpu_2x_clk	Updated description and figure.
Configure DPU Parameters	Updated figure.
Connecting a DPU with a Processing System in the Xilinx SoC	Updated section.
Assign Register Address for DPU	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 7: Example Design	Added chapter regarding the DPU targeted reference design.
02/28/2019 Version 1.0	
Initial release.	N/A

Send Feedback

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including 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