DPU for Convolutional Neural Network v2.0

DPU IP Product Guide

PG338 (v2.0) June 7, 2019

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
06/07/2019 Version 2.0	
DNNDK	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.
Using Clock Wizard	Updated description and figure.
Matched Routing	Updated description and figure.
Configure DPU Parameters	Updated figure.
Connect 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 Flow	Updated section.
03/26/2019 Version 1.2	
Build the PetaLinux Project	Updated descrption.
Build the Demo	Updated figure.
Demo Execution	Updated code.
03/08/2019 Version 1.1	
Table 6: Reg_dpu_base_addr	Updated descrption.
Figure 10: DPU Configuration	Updated figure.

Build the PetaLinux Project	Updated code.
Build the Demo	Updated descrption.
03/05/2019 Version 1.1	
Chapter 6: Example Design	Added chapter regarding the DPU targeted reference design.
02/28/2019 Version 1.0	
Initial release	N/A

Table of Contents

Introduction

The Xilinx® Deep Learning Processor Unit (DPU) is a configurable engine dedicated for convolutional neural network. The computing parallelism can be configured according to the selected device and application. It includes a set of efficiently optimized instructions. It can support most convolutional neural networks, such as VGG, ResNet, GoogLeNet, YOLO, SSD, MobileNet, FPN, etc.

Features

- *One slave AXI interface for accessing configuration and status registers.*
- *One 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*
    - *Configurable core number up to three*
    - *Convolution and deconvolution*
    - *Depthwise convolution*
    - *Max pooling*
    - *Average pooling*
    - *ReLU, ReLU6, and Leaky ReLU*
    - *Concat*
    - *Elementwise*
    - *Dilation*
    - *Reorg*
    - *Fully connected layer*
    - *Softmax*
    - *Batch Normalization*
    - *Split*

DPU IP Facts Table	
Core Specifics	
Supported Device Family	Zynq®-7000 SoC and UltraScale+™ MPSoC Family
Supported User Interfaces	Memory-mapped AXI interfaces
Resources	See Chapter 3: DPU Configuration
Provided with Core	
Design Files	Encrypted RTL
Example Design	Verilog
Constraint 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	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. Linux OS and driver support information are available from DPU TRD or DNNDK.
2. If the requirement is on Zynq-7000 SoC, view the notifications in Chapter 5: Development Flow.
3. For the supported versions of the tools, see the *Vivado Design Suite User Guide: Release Notes Installation, and Licensing* (UG973).
4. The DPU is driven by instructions. When the target NN, DPU hardware architecture, or the AXI data width is changed, the related .elf file which contains DPU instructions must be changed accordingly.

Introduction

The Xilinx® Deep Learning Processor Unit (DPU) is a programmable engine dedicated for convolutional neural networks. The unit contains register configure module, data controller module, and convolution computing module. There is a specialized instruction set for DPU, which enables DPU to work efficiently for many convolutional neural networks. The deployed convolutional neural network in DPU includes VGG, ResNet, GoogLeNet, YOLO, SSD, MobileNet, FPN, etc.

The DPU IP can be integrated as a block in the programmable logic (PL) of the selected Zynq®-7000 SoC and Zynq UltraScale™+ MPSoC devices with direct connections to the processing system (PS). To use DPU, you should prepare the instructions and input image data in the specific memory address that DPU can access. The DPU operation also requires the application processing unit (APU) to service interrupts to coordinate data transfer.

The top-level block diagram of DPU is shown here.

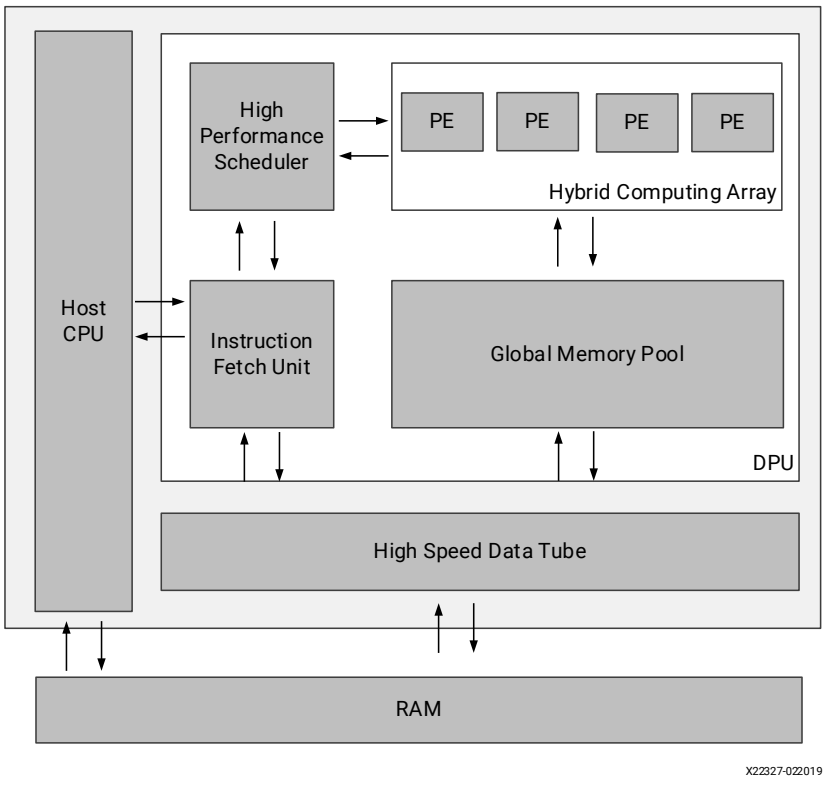

Figure 1: Top-Level Block Diagram

Development Tools

Use the Xilinx Vivado Design Suite to integrate DPU into your own project. Vivado Design Suite 2018.2 or later version is recommended. Previous versions of Vivado can also be supported. For requests, contact your sales representative.

Device Resources

The DPU logic resource is scalable across Xilinx UltraScale+ MPSoC and Zynq-7000 devices. For the detailed resource utilization, refer to Chapter 3: DPU Configuration.

How to Run DPU

The DPU operation depends on the driver which is included in the Xilinx Deep Neural Network Development Kit (DNNDK) toolchain.

You can download the free developer resources from the Xilinx website:
https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge

Refer to the *DNNDK User Guide* (UG1327) to obtain an essential guide on how to run a DPU with DNNDK tools. The basic development flow is shown in the following figure. First, use Vivado to generate the bitstream. Then, download the bitstream to the target board and install the DPU driver. For instructions on how to install the DPU driver and dependent libraries, refer to the *DNNDK User Guide* (UG1327).

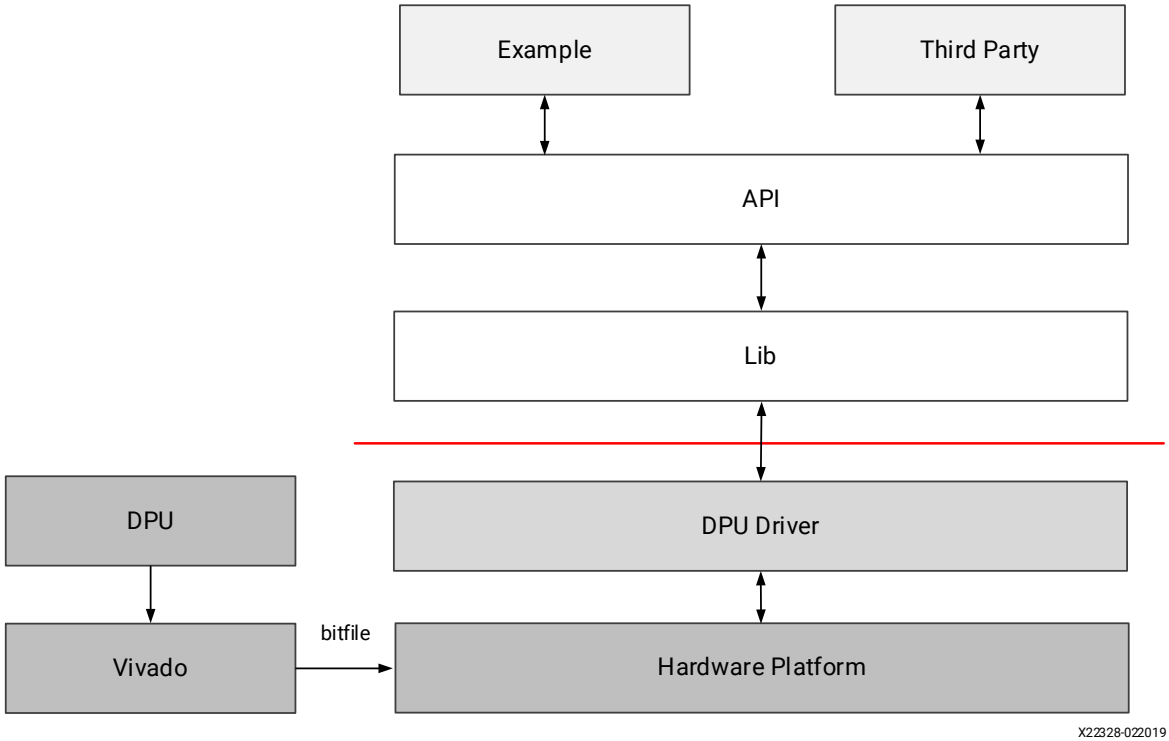

Figure 2: Basic Development Flow

Example System with DPU

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

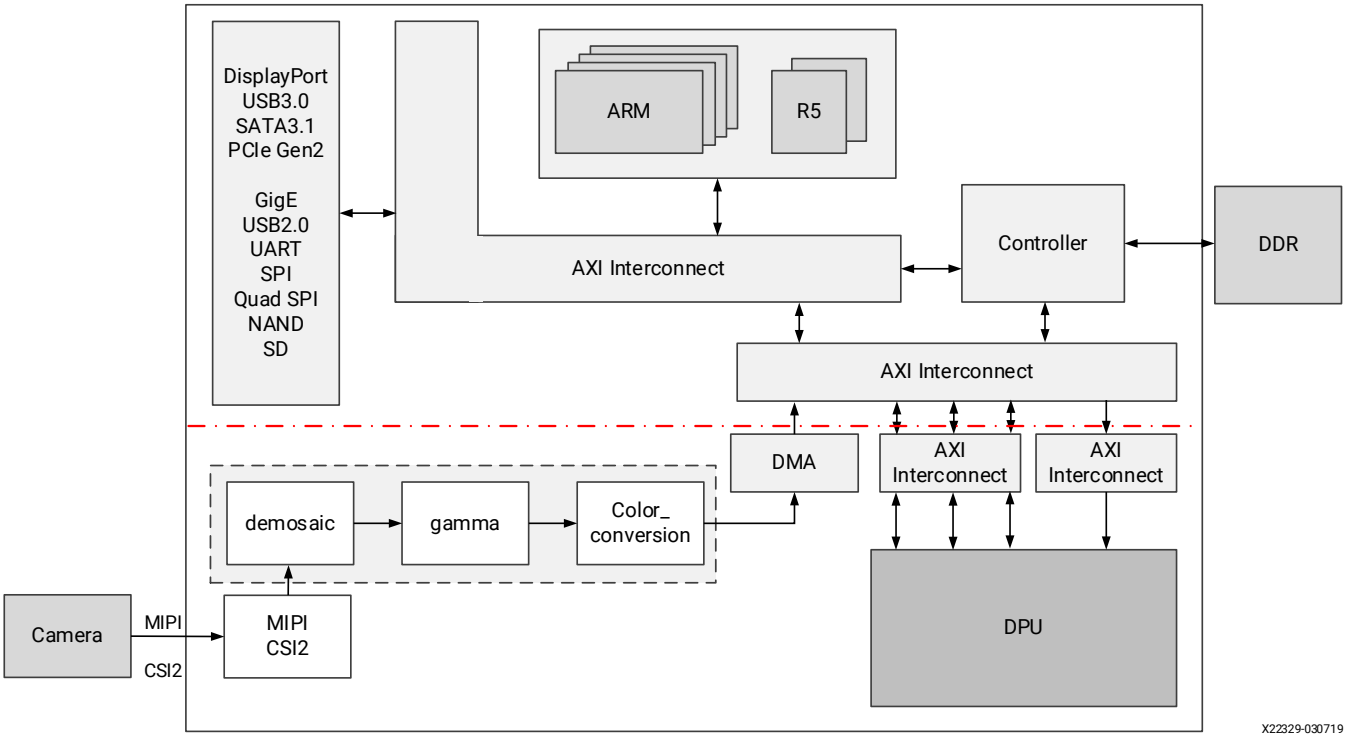

Figure 3: Example System with Integrated DPU

DNNDK

Deep Neural Network Development Kit (DNNDK) is a full-stack deep learning toolchain for inference with the DPU.

As shown in the following figure, DNNDK is composed of Deep Compression Tool (DECENT), Deep Neural Network Compiler (DNNC), Neural Network Runtime (N2Cube), and DPU Profiler.

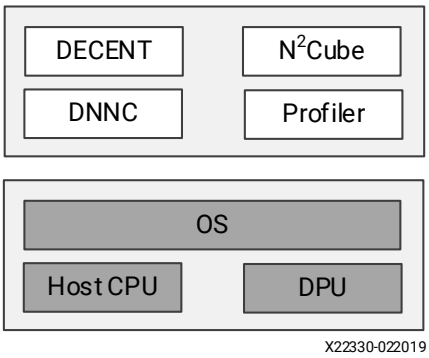

Figure 4: DNNDK Toolchain

The DPU instructions are generated offline with DNNC and the instruction file has a suffix elf. The instruction is strongly related to the DPU architecture, target neural network, and the AXI data width. When these configurations are changed, the instruction file must be regenereated accordingly. The following figure illustrates the hierarchy of executing deep learning applications on the target hardware platform with DPU.

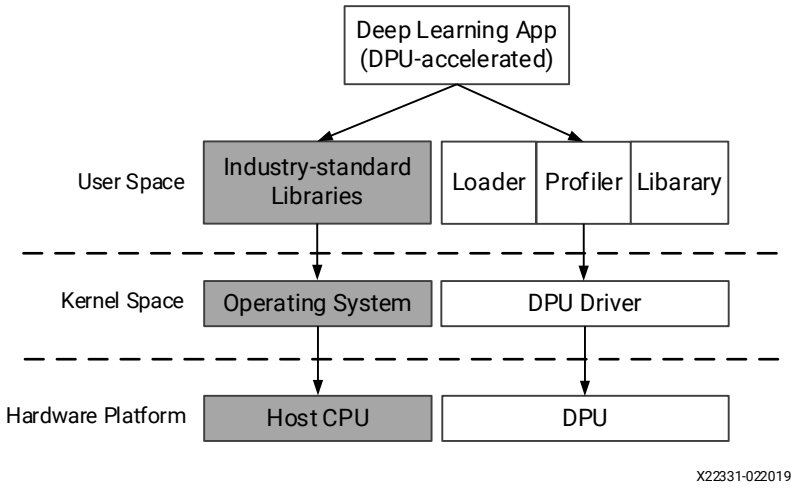

Figure 5: Application Execution Hierarchy

In the DPU design flow, if you want to deploy your own neural networks into the DPU platform, download the latest DNNDK package and compile your model by DNNDK tools to suit the DPU platform. The latest Xilinx DNNDK_V3.0 package can be accessed by https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge.

Within the DNNDK v3.0 package, there are two versions of DNNC binaries available for Ubuntu 14.04 and 16.04 individually under the folders of `xilinx_dnndk_v3.0/host_x86/pkgs/ubuntu14.04/` and `xilinx_dnndk_v3.0/host_x86/pkgs/ubuntu16.04/`. The dnnc-dpu1.4.0 is for DPU with low RAM Usage and the dnnc-dpu1.4.0.1 is for DPU with high RAM Usage. Copy the correct version of the DNNC binary to your host machine.

Licensing and Ordering Information

This IP module is provided at no additional cost under the terms of the Xilinx End User License.

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

Hardware Architecture

The detailed hardware architecture of DPU is shown in the following figure. After start-up, DPU fetches instructions from the off-chip memory and parses instructions to operate the computing engine. The instructions are generated by the DNNDK compiler where substantial optimizations have been performed.

To improve the efficiency, abundant on-chip memory in Xilinx® devices is used to buffer the intermediate data, input, and output data. The data is reused as much as possible to reduce the memory bandwidth. Deep pipelined design is used for the computing engine. Like other accelerators, the computational arrays (PE) take full advantage of the fine-grained building blocks, which includes multiplier, adder, accumulator, etc. in Xilinx devices.

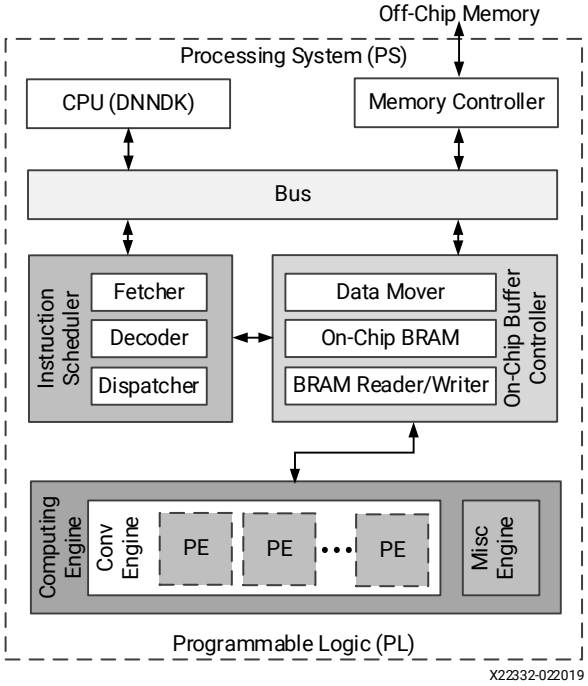

Figure 6: DPU Hardware Architecture

DSP with Enhanced Usage (DPU_EU)

In the previous DPU version, the general logic and DSP slices work in the same clock domain, though technically the latter can run at a higher frequency. To enhance the usage of DSP slices in DPU, the advanced DPU_EU version was designed.

The EU in "DPU_EU" means Enhanced Usage of DSP slices. DSP Double Data Rate (DDR) technique is used to improve the performance achieved with the device. Therefore, two input clocks for DPU is needed, one for general logic, and the other for DSP slices. The difference between DPU and DPU_EU is shown here.

All DPU mentioned in this document refer to DPU_EU, unless otherwise specified.

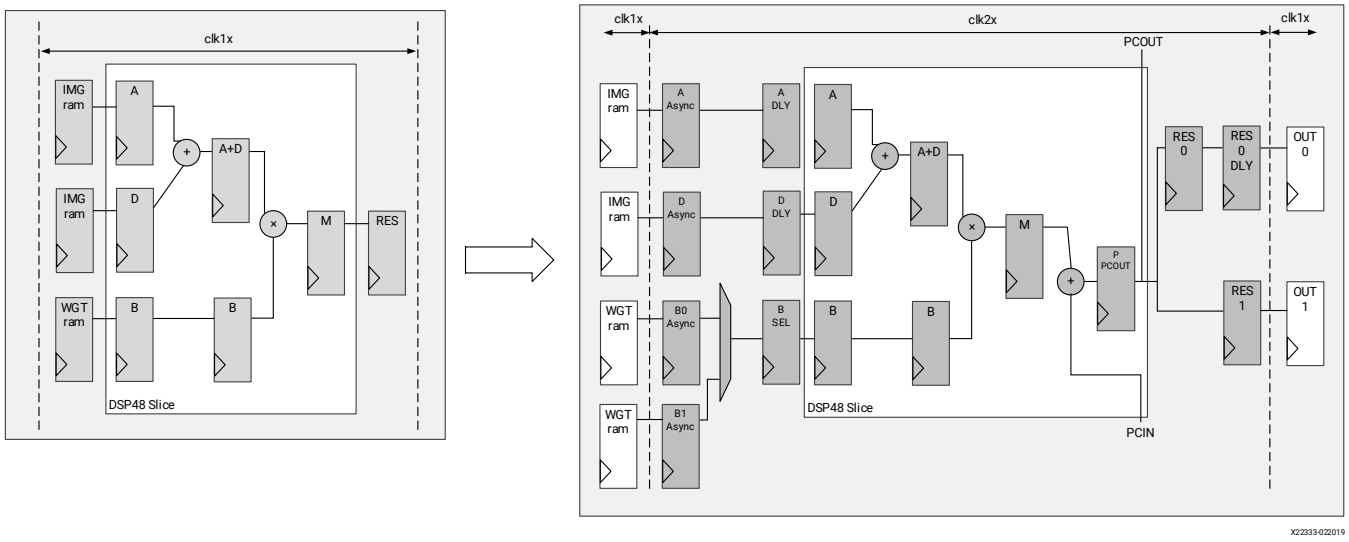

Figure 7: Difference between DPU and DPU_EU

Port Descriptions

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

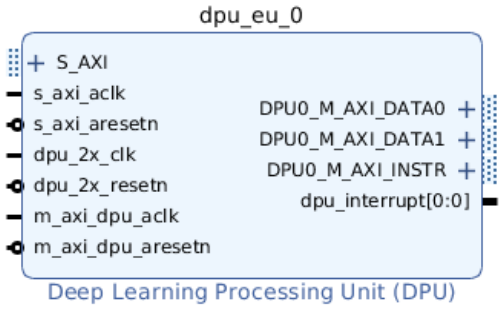

Figure 8: DPU_EU IP Port

The DPU I/O signals are listed and described in Table 1.

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 unit in DPU. The frequency is two times of m_axi_dpu_aclk.
dpu_2x_resetn	Reset	1	I	Active-Low reset for DSP unit
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 instruction of DPU.
DPUx_M_AXI_DATA0	Memory mapped AXI master interface	128	I/O	128-bit Memory mapped AXI interface for DPU data accessing.
DPUx_M_AXI_DATA1	Memory mapped AXI master interface	128	I/O	128-bit Memory mapped AXI interface for DPU data accessing.
dpu_interrupt	Interrupt	1~3	O	Active-High interrupt output from DPU. The data width is decided by the DPU number.
SFM_M_AXI (optional)	Memory mapped AXI master interface	128	I/O	128-bit Memory mapped AXI interface for the data accessing of softmax module.
sfm_interrupt (optional)	Interrupt	1	O	Active-High interrupt output from softmax module.

Notes:

1. The softmax interface only appears when the softmax option in the DPU is set as enable.

Register Space

The DPU IP implements registers in the programmable logic. Table 2 shows the DPU IP registers. These registers are accessible from the host CPU 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 respectively. All the reset signals are active-High. The details of reg_dpu_reset is shown in Table 2.

Table 2: Reg_dpu_reset

Register	Address Offset	Width	Type	Description
Reg_dpu_reset	0x004	32	R/W	[0] – reset of DPU core 0 [1] – reset of DPU core 1 [2] – reset of DPU core 2

Reg_dpu_isr

The reg_dpu_isr register represents the interrupt status of all DPU cores integrated in the DPU IP. The lower three bits of this register shows the interrupt status of up to three DPU cores respectively. The details of reg_dpu_irq is shown in Table 3.

Table 3: Reg_dpu_isr

Register	Address Offset	Width	Type	Description
Reg_dpu_isr	0x608	32	R	[0] – interrupt status of DPU core 0 [1] – interrupt status of DPU core 1 [2] – interrupt status of DPU core 2

Reg_dpu_start

The reg_dpu_start register is the start signal for DPU core. There is one start register for each DPU core. The details of reg_dpu_start is shown in Table 4.

Table 4: Reg_dpu_start

Register	Address Offset	Width	Type	Description
Reg_dpu0_start	0x220	32	R/W	Control the start-up of DPU core0.
Reg_dpu1_start	0x320	32	R/W	Control the start-up of DPU core1.
Reg_dpu2_start	0x420	32	R/W	Control the start-up of DPU core2.

Reg_dpu_instr_addr

The reg_dpu_instr_addr register is used to indicate the instruction address of DPU core. There are three registers which are reg_dpu0_instr_addr, reg_dpu1_instr_addr, and reg_dpu2_instr_addr. The details of reg_dpu_instr_addr are shown in Table 5.

Table 5: Reg_dpu_instr_addr

Register	Address Offset	Width	Type	Description
Reg_dpu0_instr_addr	0x20c	32	R/W	[0] –The instruction start address in external memory for DPU core0.
Reg_dpu1_instr_addr	0x30c	32	R/W	[0] –The instruction start address in external memory for DPU core1.
Reg_dpu2_instr_addr	0x40c	32	R/W	[0] –The instruction start address in external memory for DPU core2.

Reg_dpu_base_addr

The reg_dpu_base_addr register is used to indicate the address of input image and parameters for DPU in the external memory. The width of dpu_base_addr 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 dpu_base_addr. The reg_dpu0_base_addr0_l represents the lower 32 bits of the base address0 in DPU core0, and the reg_dpu0_base_addr0_h represents the upper eight bits of the base address0 in DPU core0.

There are eight groups of DPU base address for each DPU core and in total 24 groups of DPU base address for up to three DPU cores. The details of reg_dpu_base_addr are shown in Table 6.

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 the 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 the base address0 of DPU core0.
Reg_dpu0_base_addr1_l	0x22C	32	R/W	The lower 32 bits of the 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 the base address1 of DPU core0.
Reg_dpu0_base_addr2_l	0x234	32	R/W	The lower 32 bits of the 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 the base address2 of DPU core0.
Reg_dpu0_base_addr3_l	0x23C	32	R/W	The lower 32 bits of the 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 the base address3 of DPU core0.
Reg_dpu0_base_addr4_l	0x244	32	R/W	The lower 32 bits of the 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 the base address4 of DPU core0.
Reg_dpu0_base_addr5_l	0x24C	32	R/W	The lower 32 bits of the 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 the base address5 of DPU core0.
Reg_dpu0_base_addr6_l	0x254	32	R/W	The lower 32 bits of the 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 the base address6 of DPU core0.
Reg_dpu0_base_addr7_l	0x25C	32	R/W	The lower 32 bits of the 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 the base address7 of DPU core0.
Reg_dpu1_base_addr0_l	0x324	32	R/W	The lower 32 bits of the 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 the base address0 of DPU core1.
Reg_dpu1_base_addr1_l	0x32C	32	R/W	The lower 32 bits of the 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 the base address1 of DPU core1.
Reg_dpu1_base_addr2_l	0x334	32	R/W	The lower 32 bits of the 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 the base address2 of DPU core1.
Reg_dpu1_base_addr3_l	0x33C	32	R/W	The lower 32 bits of the 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 the base address3 of DPU core1.
Reg_dpu1_base_addr4_l	0x344	32	R/W	The lower 32 bits of the 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 the base address4 of DPU core1.
Reg_dpu1_base_addr5_l	0x34C	32	R/W	The lower 32 bits of the 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 the base address5 of DPU core1.
Reg_dpu1_base_addr6_l	0x354	32	R/W	The lower 32 bits of the 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 the base address6 of DPU core1.
Reg_dpu1_base_addr7_l	0x35C	32	R/W	The lower 32 bits of the 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 the base address7 of DPU core1.
Reg_dpu2_base_addr1_l	0x42C	32	R/W	The lower 32 bits of the 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 the base address1 of DPU core2.
Reg_dpu2_base_addr2_l	0x434	32	R/W	The lower 32 bits of the 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 the base address2 of DPU core2.
Reg_dpu2_base_addr3_l	0x43C	32	R/W	The lower 32 bits of the 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 the base address3 of DPU core2.
Reg_dpu2_base_addr4_l	0x444	32	R/W	The lower 32 bits of the 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 the base address4 of DPU core2.
Reg_dpu2_base_addr5_l	0x44C	32	R/W	The lower 32 bits of the 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 the base address5 of DPU core2.
Reg_dpu2_base_addr6_l	0x454	32	R/W	The lower 32 bits of the 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 the base address6 of DPU core2.
Reg_dpu2_base_addr7_l	0x45C	32	R/W	The lower 32 bits of the 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 the base address7 of DPU core2.

Interrupts

Upon the completion of one DPU task, an interrupt from DPU occurs to signal the completion of the task. The active-High of the Reg_dpu0_start means the start of a DPU task for DPU core0. At the end of the DPU task, DPU sends an interrupt and one bit in the register 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 the DPU core1 finishes a task while the DPU core 0 is still working, the Reg_dpu_isr is set as 2'b10.

The data width of dpu_interrupt is determined by the number of DPU cores. When the parameter of DPU_NUM is set to 2, it means the DPU IP is integrated with two DPU cores, and the data width of the dpu_interrupt signal is two bits. 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 PS is described in the Device Tree file, which indicates the interrupt number of DPU connected to the PS. The reference connection is shown as Figure 9.

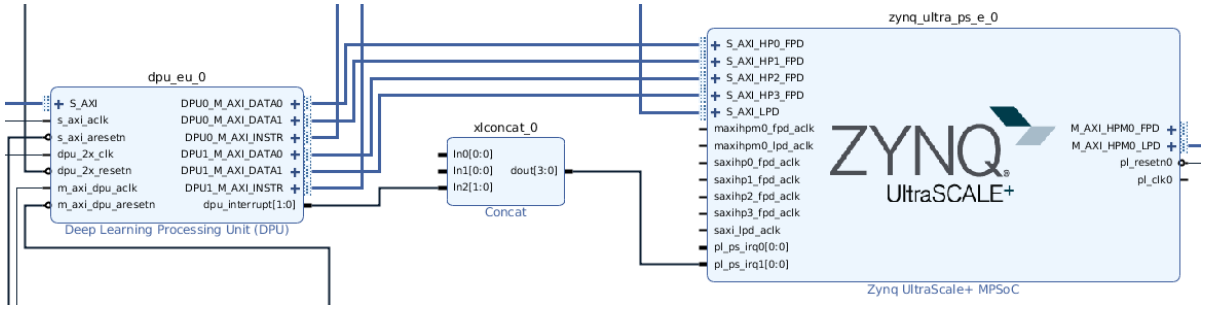

Figure 9: Reference Connection for DPU Interrupt

Notes:

1. If DPU is integrated in MPSoC and working with the DNNDK package, you should connect the dpu_interrupt at the bit 10 in the irq signal of PS. For example, if the DPU_NUM is set as 2, the 2-bit dpu_interrupt should connect with irq10 and irq11 of PS. When the DPU is integrated in Zynq®-7000 devices, the dpu interrupt should be connect at the bit 0 in the irq signal of PS.
2. If the option of softmax in DPU is set as enable, then the interrupt of softmax should connect at the bit 14 in the irq signal of PS when DPU is working with the DNNDK package. For Zynq-7000 devices, it is bit 3.
3. irq7~irq0 corresponds to pl_ps_irq0[7:0].
4. irq15~irq8 corresponds to pl_ps_irq1[7:0].

Introduction

The DPU IP provides some user-configurable parameters to optimize the resources or the support of different features. You can select different configurations to use on the preferred DSP slices, LUT, block RAM (BRAM), and UltraRAM utilization based on the programmable logic resources that are allowed. There is also an option to determine the number of DPU cores that will be used.

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

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

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

	Stride_h	Arbitrary
	Padding_w	1: kernel_w-1
	Padding_h	1: kernel_h-1
	Input Size	Arbitrary
	Input Channel	1 – 256 * channel_parallel
	Output Channel	1 – 256 * channel_parallel
	Activation	ReLU & LeakyReLU
Max Pooling	Kernel Sizes	W: 1-16 H: 1-16
	Strides	W: 1-4 H:1-4
	Padding	W: 1-4 H: 1-4
Element Wise	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, the channel_parallel of DPU-B1152 is 12, the channel_parallel of DPU-B4096 is 16.

Configuration Options

You can configure the DPU with some predefined options which includes DPU core number, DPU convolution architecture, DSP cascade, DSP usage, and UltraRAM usage. These options enable the DPU IP configurable in terms of DSP slice, LUT, block RAM, and UltraRAM utilization. The following figure shows the DPU configuration.

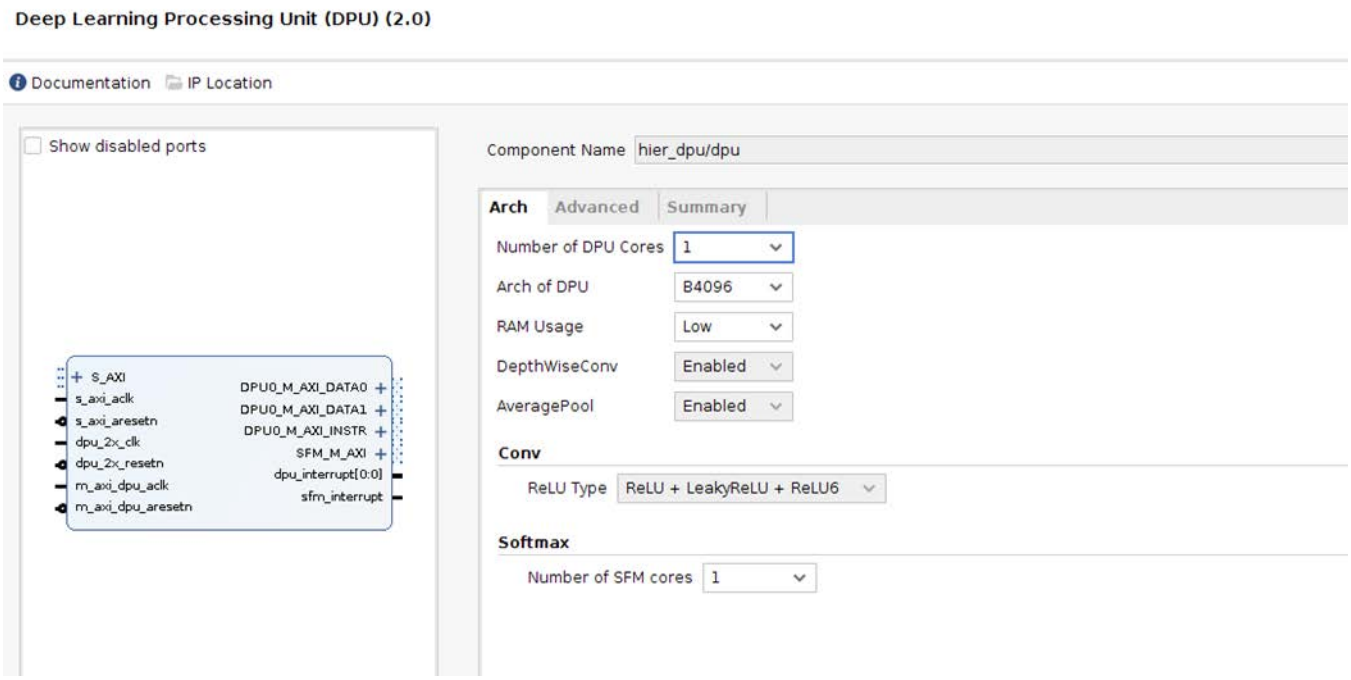

Figure 10: DPU Configuration – Arch Tab

DPU Core Number

You can set up to three DPU cores in one IP. Multiple DPU cores can be used to achieve higher performance. Consequently, it consumes more programmable logic resource.

If the requirement is to integrate more than three cores, send the request to a Xilinx® sales representative.

DPU Convolution Architecture

The DPU IP can be configured with different convolution architectures which is related to the parallelism of the convolution unit. The optional architecture for DPU IP includes 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. The different convolution architecture requires different programmable logic resource. The larger convolution architecture can achieve higher performance with more resources. The parallelism for different convolution architecture is listed in Table 8.

Table 8: Parallelism for Different Convolution Architecture

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 finishes a multiplication and an accumulation, which are two operations. So, the peak 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 block. The on-chip memory block consists of some amounts of RAMs which can be instantiated by BRAM and UltraRAM. The option of RAM Usage determines the whole size of on-chip memory block in different DPU architecture, and the setting is for all the DPU cores in the DPU IP. The high RAM Usage means that the on-chip memory block will be larger, and it will be more flexible for DPU to handle the intermediate data. The higher RAM Usage means higher performance in each DPU core. The number of BRAM36K in different architecture between low and high RAM Usage is illustrated in Table 9.

Note that the DPU instruction set for different options of RAM Usage is different. When the option of RAM Usage has changed, the DPU instructions file should be regenerated correspondingly.

Table 9: Number of BRAM36K in Different Architecture 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

DepthwiseConv

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

While in depthwise separable convolution case, the depthwise convolution is the first step. Depthwise convolution performs convolution for each feature map separately. As shown in the following figure, you can see the depthwise separable convolution. The second step is to perform pointwise convolution, which is also standard convolution with kernel size 1x1. The option of depthwise convolution is set to be enable now, and the support for disable option will be updated in a future version.

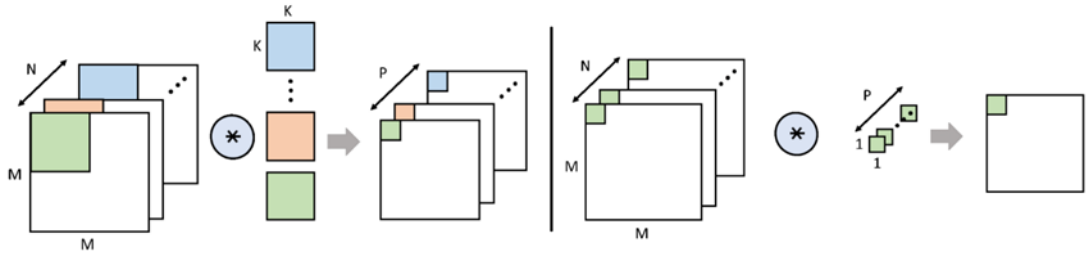

Figure 11: Depthwise Convolution and Pointwise Convolution

The parallelism of Depthwise Convolution is half of the parallelism of pixel.

AveragePool

The option of AveragePool determines whether an average pool module is instantiated in the DPU. The supported range size of average pool is from 2x2, 3x3, ..., to 8x8 in the DPU. When the AveragePool is enabled, the average pool operation can be processed in the DPU. Note that the option of AveragePool is set to enable in this version, and this migth be updated in a future version.

Softmax

The softmax function can be processed in the DPU IP and the core number of softmax can be set as 0 or 1. When the number of softmax cores is set as 1, a dedicated module for softmax will be instantiated in the DPU. The hardware implementation of softmax can achieve about 160 times acceleration compared to software. It should be noted that the hardware softmax module takes approximately 10000 LUTs, 4.5 BRAMs, and 21 DSP. You can choose whether to use hardware modules to accelerate softmax according to the resource's limitation.

When the 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 is working in the same clock domain of the DPU. The AXI clock for SFM_M_AXI is also the m_axi_dpu_aclk.

ReLU Type

The option of ReLU Type determines which kind of ReLU function can be applied in the DPU. The option "ReLU + LeakyReLU + ReLU6 "means that DPU can apply ReLU, LeakyReLU, or ReLU6 as the

activation function at runtime. The ReLU type is forced to be “ReLU + LeakyReLU + ReLU6”in this version. Flexible configuration for ReLU type might be available in a future version.

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

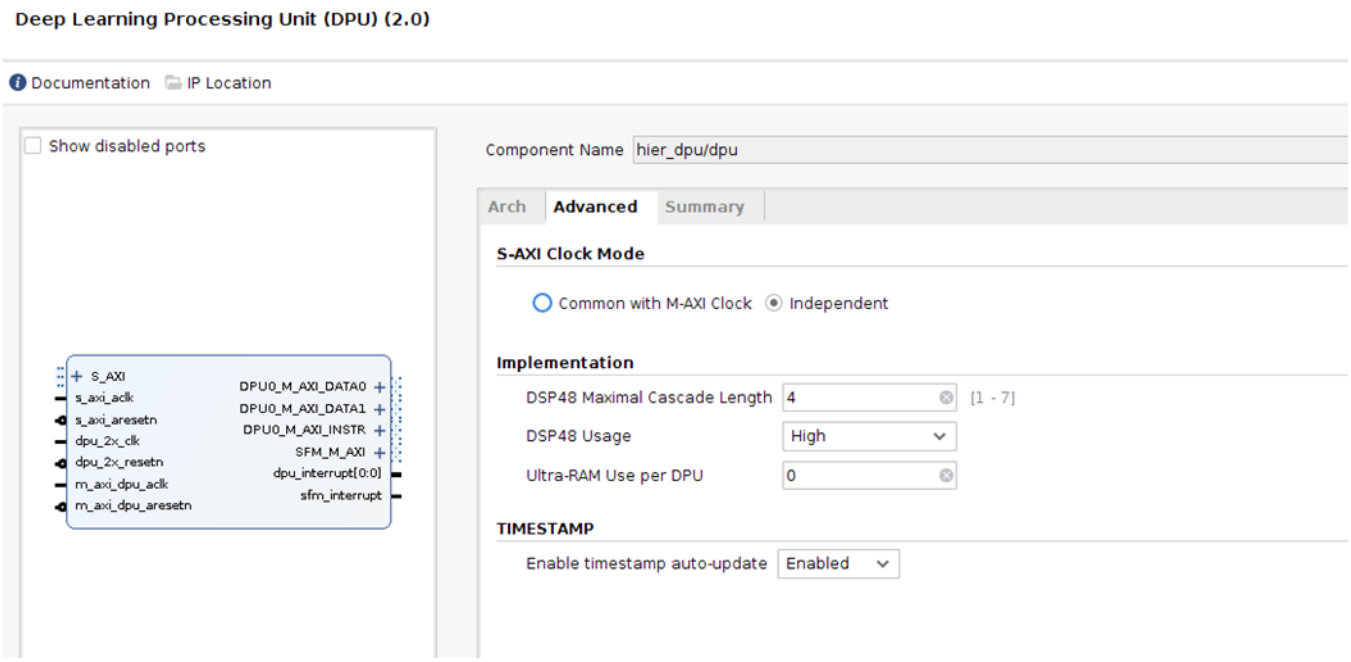

Figure 12: DPU Configuration – Advanced Tab

S-AXI Clock Mode

The s_axi_aclk is the working clock for S-AXI interface. The configuration for S-AXI Clock mode sets whether you use the M-AXI Clock for S-AXI interface. When the option is selected for common clock with M-AXI Clock, the S-AXI interface uses the same m_axi_aclk clock with M-AXI interface, and the s_axi_aclk port is hidden. The option of independent means the S-AXI Clock is an independent clock and you can set arbitrary frequency for the s_axi_aclk.

DSP Cascade

You can select the maximal length of DSP48E slice cascade chain. Typically, the larger cascade length indicates less logic resources, but it might lead to worse timing. The smaller cascade length might use more fabric resources which is not economical for small devices. Xilinx recommends selecting the mid-value, which is 4, in the first iteration and adjust the value if the timing is not met.

DSP Usage

You can select whether DSP48E slices are used for the accumulation in the DPU convolution module. If the low DSP usage is selected, the DPU IP will use DSP slices for multiplication only in the convolution. In the high DSP usage mode, the DSP slice will be used for both multiplication and accumulation. As a result, the high DSP usage consumes more DSP slices and less LUT. The difference

of logic utilization between high and low DSP usage is shown in Table 10. The data is tested on the Xilinx ZCU102 platform without Depthwise Conv, Average Pooling, ReLU6, and Leaky ReLU features.

Table 10: Resources of Different DSP Usage

High DSP Usage					Low DSP Usage				
Arch	**LUT**	**Register**	**BRAM**	**DSP**	**Arch**	**LUT**	**Register**	**BRAM**	**DSP**
B512	20177	31782	69.5	98	B512	20759	33572	69.5	66
B800	20617	35065	87	142	B800	21050	33752	87	102
B1024	27377	46241	101.5	194	B1024	29155	49823	101.5	130
B1152	28698	46906	117.5	194	B1152	30043	49588	117.5	146
B1600	30877	56267	123	282	B1600	33130	60739	123	202
B2304	34379	67481	161.5	386	B2304	37055	72850	161.5	290
B3136	38555	79867	203.5	506	B3136	41714	86132	203.5	394
B4096	40865	92630	249.5	642	B4096	44583	99791	249.5	514

UltraRAM

There are two kinds of on-chip memory resources in Zynq® UltraScale+™ devices: block RAM and UltraRAM. The proportion of each kind of memory resources is dependent on the target chip. Each BRAM unit consists of two bram18K slices which can be flexibly configured to a memory block with sizes of 9b*4096, 18b*2048, or 36b*1024. While the UltraRAM is a fixed-size memory with the size of 72b*4096. However, the depth of essential memory unit in the DPU is always equal to 2048 and the width is equal to ICP*8 bit. For a DPU architecture of B1024, the ICP is 8, then the width of essential memory unit is 8*8 bit and each memory unit can be instantiated with one UltraRAM unit. When the ICP is greater than 8, each memory unit in the DPU needs at least two UltraRAM.

The DPU utilizes the BRAM as a memory unit by default. For a target platform with both BRAM and UltraRAM resources, configure the number of UltraRAM to determine how many UltraRAMs are used with replacing a number of BRAMs. The number of UltraRAM should be set as a multiple of the number of UltraRAM required for the essential memory unit in the DPU. A summary of the BRAM and UltraRAM utilization is shown in Figure 14.

Timestamp

There exists a timestamp for recording the time information in the DPU. The option of timestamp is set as enable by default. The enable means that the DPU records the synthesis time of the DPU project. The option of disable means that the timestamp keeps the value at the moment of the last IP update. The timestamp information can be read through DNNDK tools.

Note: *Most of the DPU configurations can be accessed by the DNNDK tools. The figure below shows the information read by the DNNDK tools.*

```
root@zcu102:~# dexplorer -w
[DPU IP Spec]
IP  Timestamp   : 2019-04-18 16:30:00
DPU Core Count  : 1

[DPU Core List]
DPU Core        : #0
DPU Enabled     : Yes
DPU Arch        : B3136F
DPU Target      : v1.4.0
DPU Freqency    : 333 MHz
DPU Features    : Avg-Pooling, LeakyReLU/ReLU6, Depthwise Conv
```

Figure 13: Timestamp

Summary Tab

After finishing the configuration in the Arch and Advanced tabs of the DPU, basic information is displayed in the Summary tab. The target version is the version of instruction set for the DPU. The recommended IRQ of DPU and SFM are for easier integration with DNNDK.

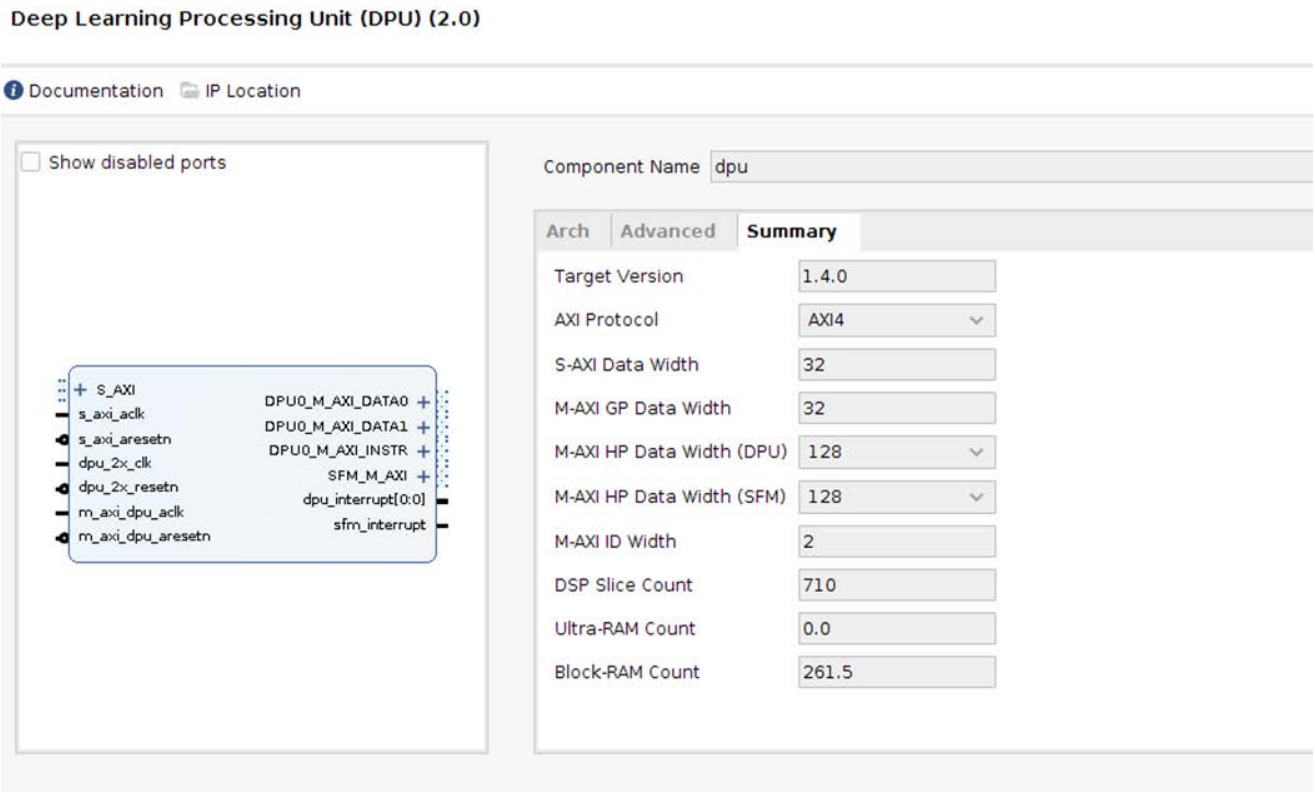

Figure 14: Summary Page of DPU Configuration

DPU Performance on Different Devices

Table 11 shows the peak performance of the DPU on different devices.

Table 11: DPU_EU Performance (GOPs) on Different Device

Device	DPU Configuration	Frequency (MHz)	Peak Performance
Z7020	B1152x1	200	230 Gops
ZU2	B1152x1	370	426 Gops
ZU3	B2304x1	370	852 Gops
ZU5	B4096x1	350	1.4 Tops
ZU7EV	B4096x2	330	2.7 Tops
ZU9	B4096x3	333	4.1 Tops

Performance of Different Models

In this section, the performance of several models is given for reference. The result was measured on the Xilinx ZCU102 board with 3x B4096 cores at 294 MHz and DNNDK v3.0, shown in Table 12.

Table 12: 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	460.2
ResNet50	7.7	224*224	Top-1: 0.7338	168.2
MobileNet_v2	0.6	299*299	Top-1: 0.6352	573.3
SSD_ADAS_VEHICLE	6.3	480*360	mAP: 0.3887	271.6
SSD_ADAS_PEDESTRIAN	5.9	640*360	mAP: 0.5649	220.7
SSD_MobileNet_v2	6.6	480*360	mAP: 0.2931	137.9
YOLO-V3-VOC	65.4	416*416	mAP: 0.8153	41.4
YOLO-V3_ADAS	5.5	512*256	mAP: 0.5301	235.4

Notes:

1. Some 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 is different. 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 Table 13. The table only shows the number of two commonly used DPU (B1152 and B4096). Note that when multiple DPU cores run in parallel, each core might not be able to run at full speed due to the limitation of I/O bandwidth.

Table 13: 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).

Introduction

There are three clock domains in the DPU IP: the register, 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 shall be configured correctly.

Clock Domain

The following figure shows the three clock domains.

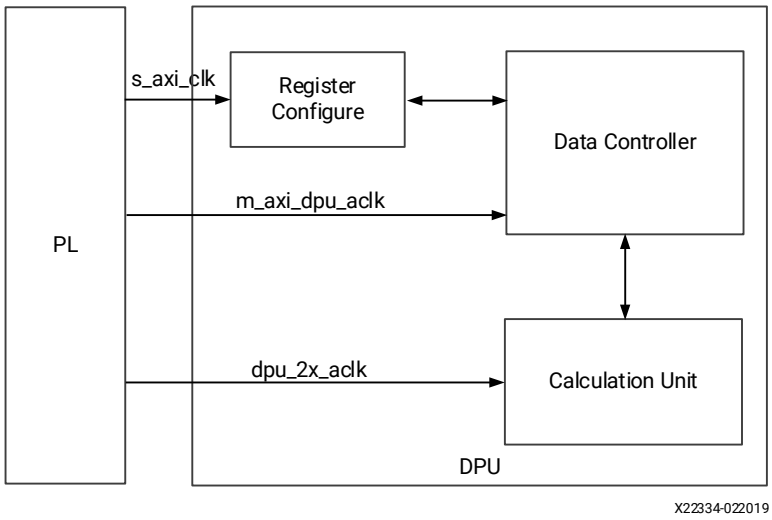

Figure 15: Clock Domain in DPU

Register Clock

The input `s_axi_clk` is used for the register configure module. This module receives the DPU configure data though the S_AXI interface and the related clock of S_AXI is `s_axi_clk`. The S_AXI clock can be configured as common with the M-AXI clock or as an independent clock. The register for DPU configure is updated at a very low frequency and most of those registers are configured 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 the frequency of 100 MHz.

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 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. You should connect `m_axi_dpu_aclk` with the AXI_MM master clock.

Computation Clock

The DSP slices in the computation unit module are in the `dpu_2x_clk` domain, which doubles the clock frequency of the data controller module. Therefore, the frequency of the `dpu_2x_clk` should be twice the `m_axi_dpu_aclk`. Furthermore, the two related clocks must be edge-aligned.

Reference Clock Generation

There are three input clocks for the DPU in which the frequency of the `dpu_2x_clk` should be two times the `m_axi_dpu_aclk` and the two clocks must be synchronous to meet the timing closure. The recommended circuit design is shown here.

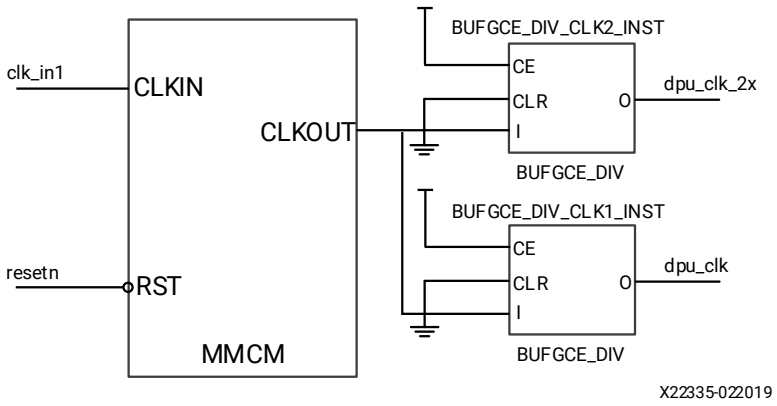

Figure 16: Reference Circuit

You can instantiate an MMCM and two BUFGCE_DIV 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`. The BUFGCE_DIV_CLK1_INST obtains the clock of whichever frequency is half of the `dpu_clk_2x`. The `dpu_clk` and `dpu_clk_2x` are generated by the same clock, so they are synchronous. The two BUFGCE_DIVs enable the skew between the two clocks to significantly decrease, which helps with timing closure.

Using Clock Wizard

Instantiating the Xilinx clock wizard IP can implement the above circuit. Though the maximum frequency of AXI-HP interfaces in Xilinx UltraScale+ MPSoC is 333 MHz, setting the frequency of `m_axi_dpu_aclk` as 333 MHz will cause serious phase error in the Clock Wizard. 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 that the parameter of the Primitive must be set to Auto.

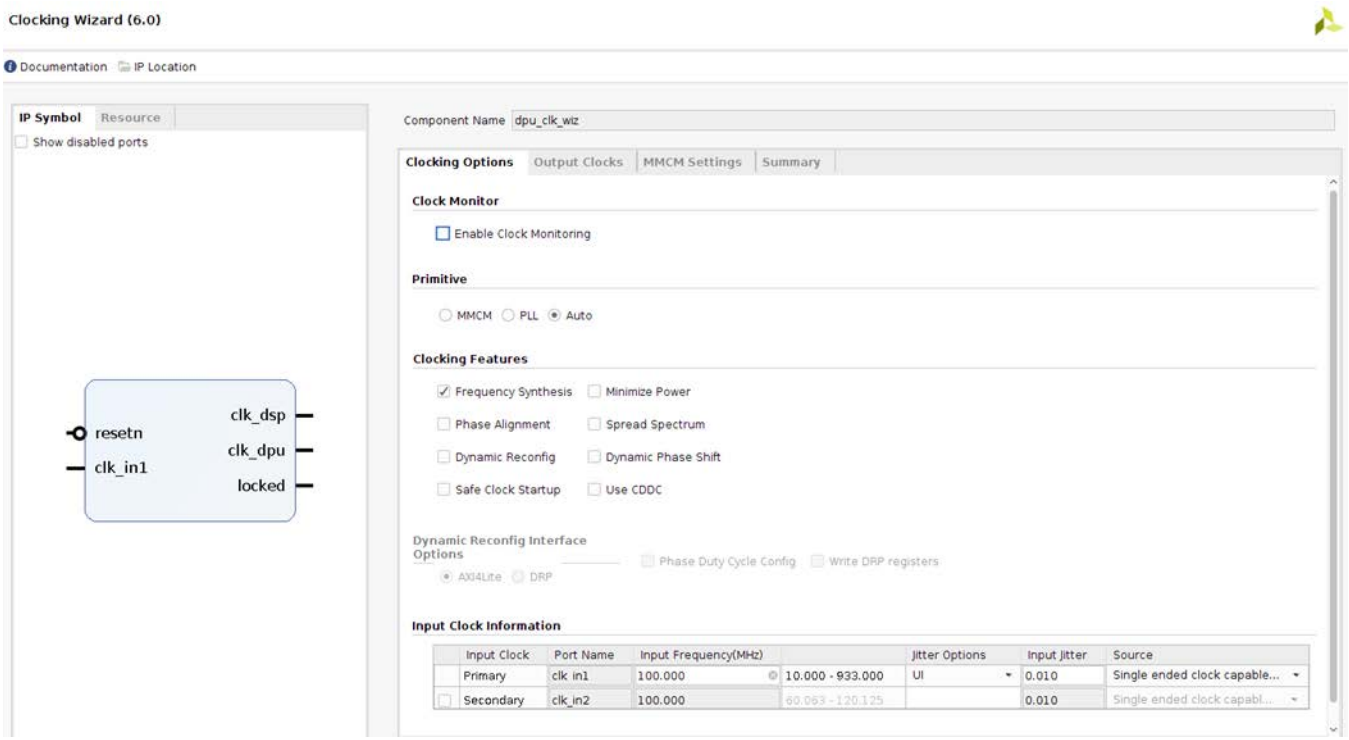

Figure 17: Recommended Clocking Options of Clock Wizard

Matched Routing

Select the Matched Routing for the 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 has significantly decreased. The related configuration is shown in the following figure.

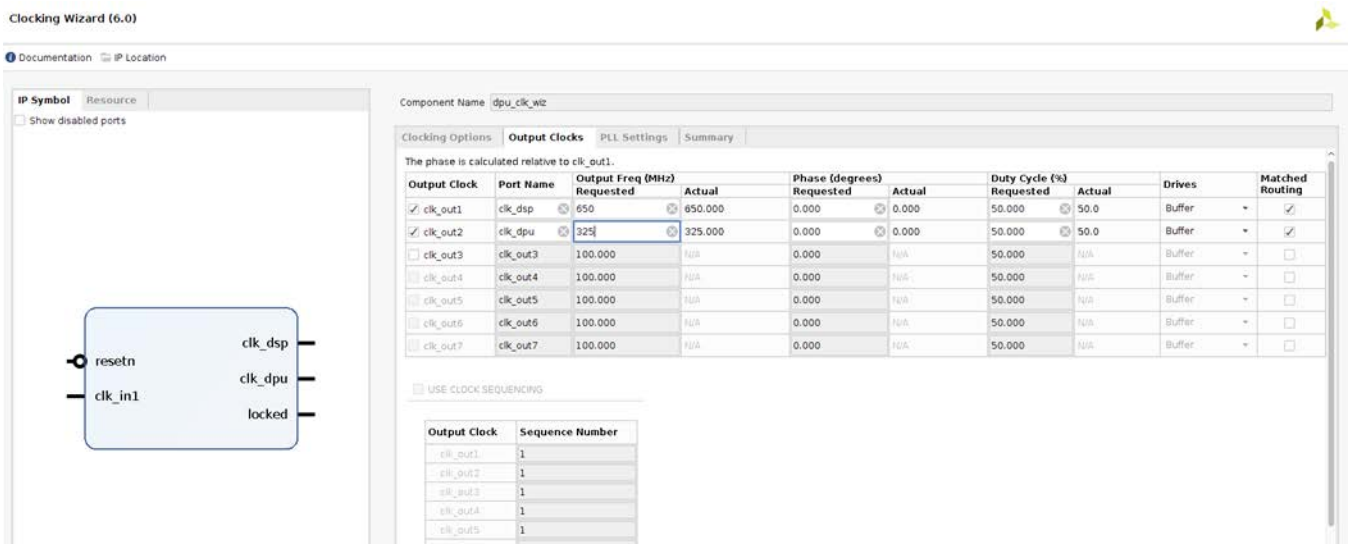

Figure 18: Matched Routing in Clock Wizard

Reset

There are three input clocks for the DPU IP and each clock has a corresponding reset. You must guarantee each pair of clocks and resets is generated in a synchronous clock domain. If the related clocks and resets are not matched, the DPU might not work properly. A recommended solution is to instantiate a Processor System Reset IP to generate a matched reset for each clock. The reference design is shown here.

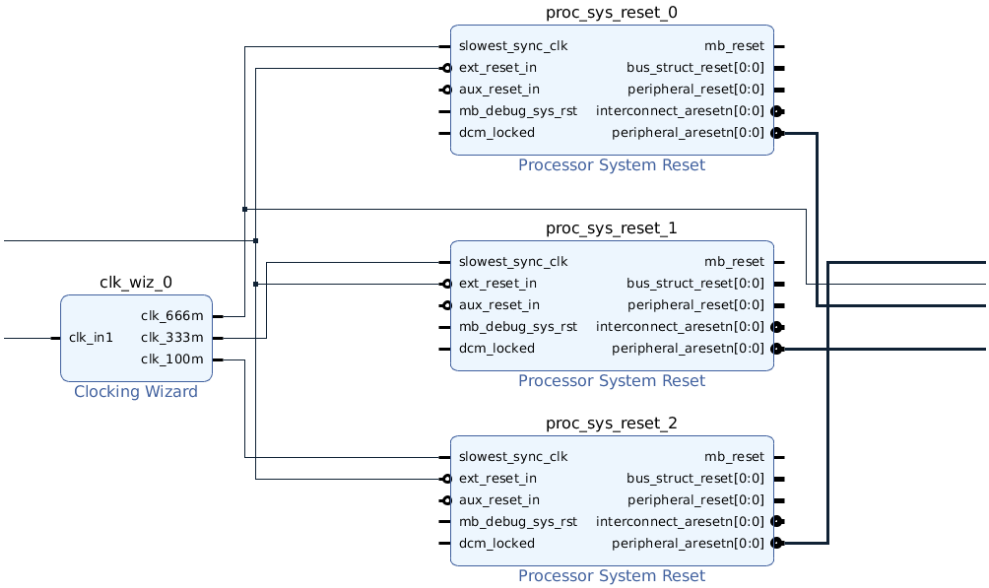

Figure 19: Reference Design for Resets

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
- Add DPU IP into Block Design
- Configure DPU Parameters
- Connect 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

In the Vivado GUI, click **Project Manager** > **IP Catalog**. In the IP Catalog tab, right-click and select **Add Repository** (Figure 20), then select the location of the DPU IP. This will appear in the IP Catalog page (Figure 21).

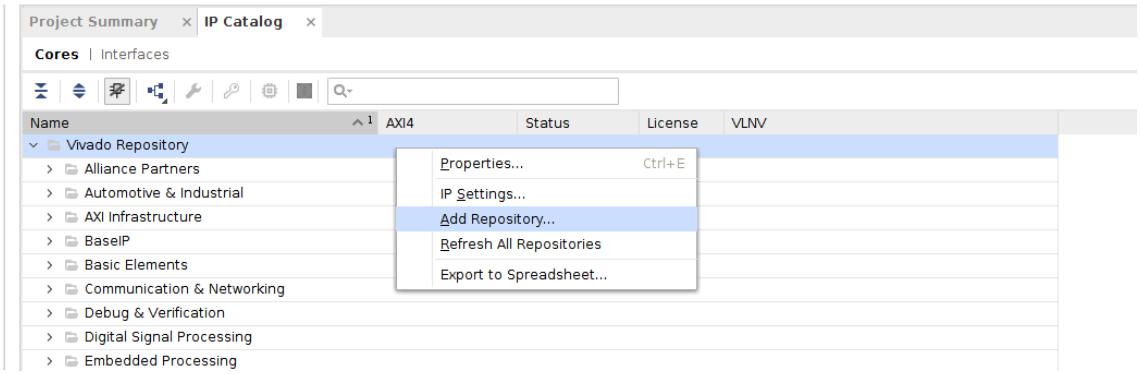

Figure 20: Add Repository

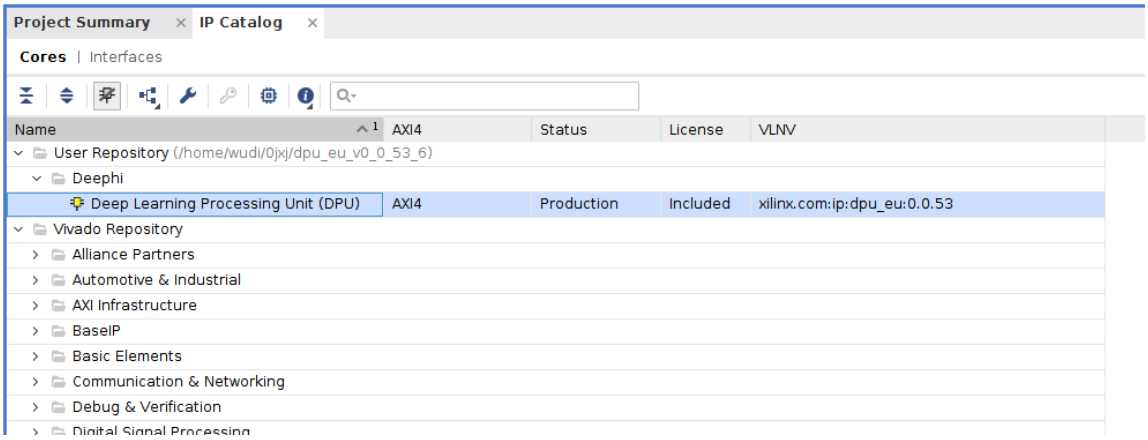

Figure 21: DPU IP in Repository

Add DPU IP into Block Design

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

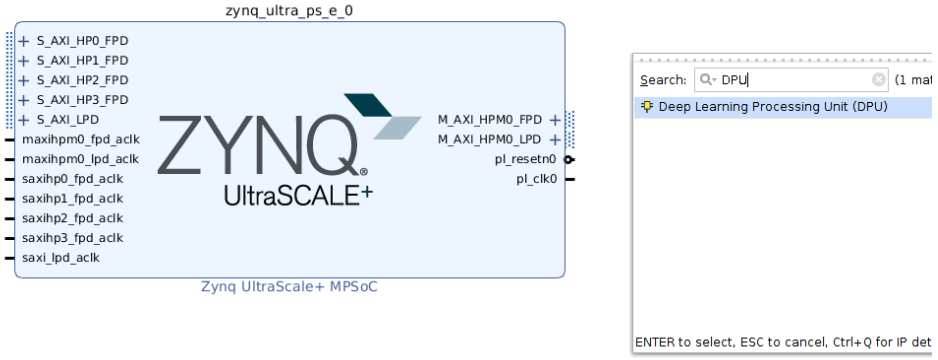

Figure 22: Search DPU IP

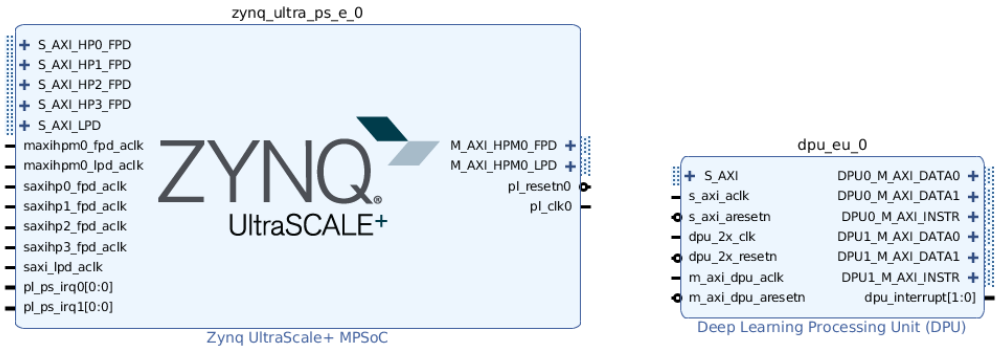

Figure 23: Add DPU IP into Block Design

Configure DPU Parameters

You can configure the DPU IP as shown in the following figure. The details about these parameters can be found in Chapter 3: DPU Configuration.

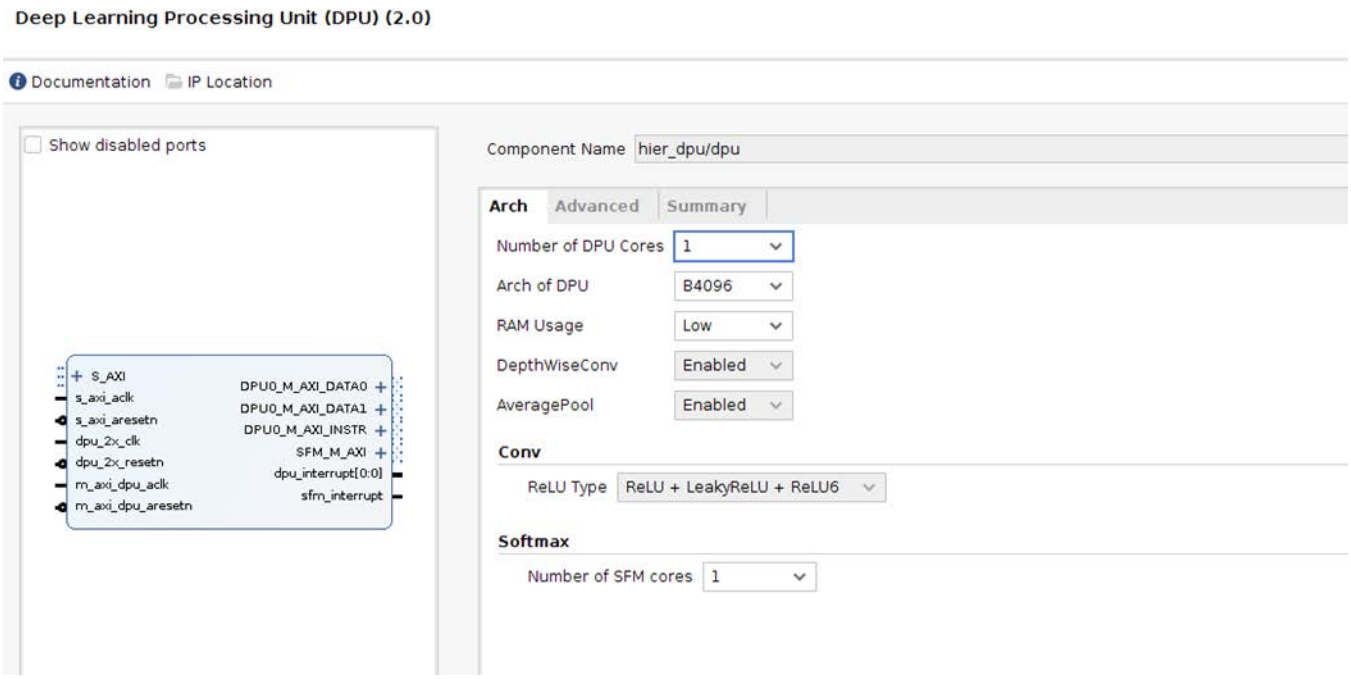

Figure 24: Configure DPU

Connect 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 accessing.

The DPU IP can be connected to the processing system (PS) with any kinds of interconnections if the DPU can correctly access the DDR memory space. Generally, when the data transferred through an Interconnect IP, the delay of data transaction will increase. The delay of the data transmission between the DPU and 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 PS is enough.

When the AXI slave ports of PS is insufficient for the DPU, an AXI interconnect for connection is inevitable. The two AXI master ports for data fetching is a high bandwidth required port and the AXI master port for instruction fetching is a low bandwidth required 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 with the PS as much as possible. The master ports of the DPU core with higher priority (smaller number, like DPU0) are recommended to be directly connected to the slave ports of PS with higher priority (smaller number, like S_AXI_HP0_FPD).

For example, if there are three DPU cores and one SFM core, therefore seven master ports, S_AXI_HP1~3, and S_AXI_HPC0, then four slave ports are available. Xilinx suggests the connection pairs as:

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

The slave port of DPU is recommended to connect with the M_AXI_HPM0_LPD of PS.

A reference connection between the DPU and PS in the Xilinx UltraScale+™ MPSoC is shown. The core number of DPU is set as 3, and the Softmax function is enabled.

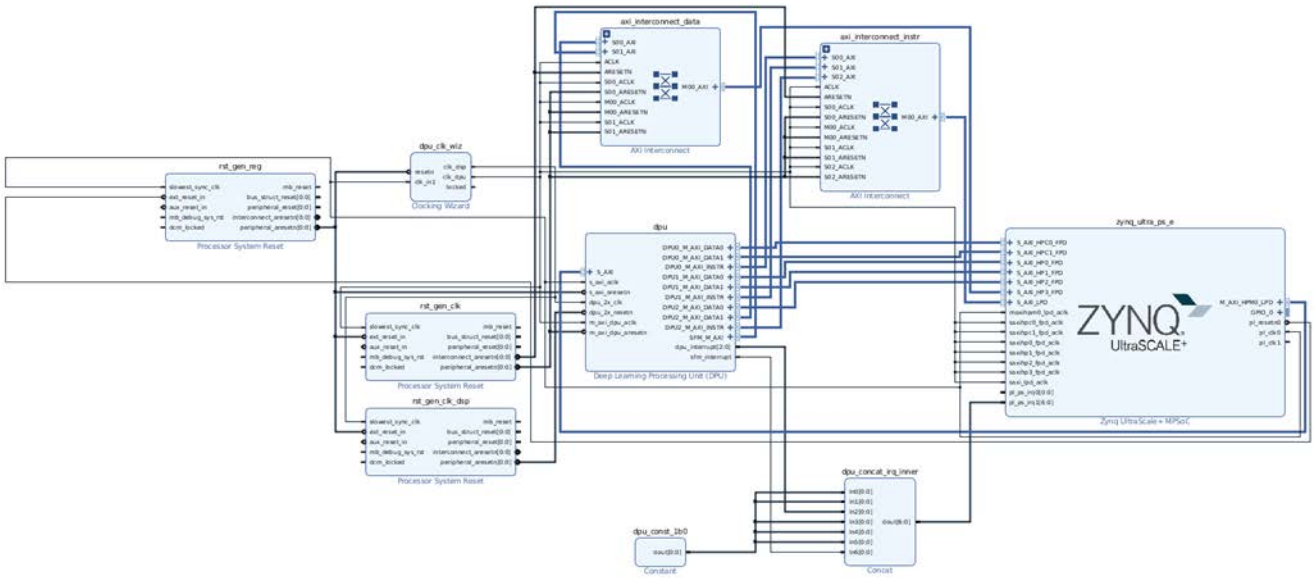

Figure 25: Connect DPU with PS of MPSoC

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: *When building a custom system with the pre-built Linux environment in the DNNDK package, the DPU slave interface must be connected to the M_AXI_HPM0_LPD PS Master and the DPU base address must be set to 0x8F00_0000 with a range of 16 MB in MPSoC series. The DPU register address in the driver and device tree file in the DNNDK package is fixed at 0x8F00_0000. If the address in the driver and device tree file is the same as the address assigned in Vivado, you can connect the DPU slave interface to any master interface in the PS and allocate any address for the DPU.*

When the DPU is integrated in Zynq-7000 devices, the DPU base address must be set to 0x4F00_0000 with a range of 16 MB with DNNDK package.

The reference address assignments of the DPU with the DNNDK package are shown here.

Diagram × **Address Editor** ×

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

Figure 26: Assign DPU Address

Generate Bitstream

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

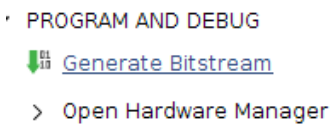

Figure 27: Generate Bitstream

Generate BOOT.BIN

You can use the Vivado SDK or PetaLinux to generate the BOOT.BIN file. For boot image creation using the Vivado SDK, refer to the *Zynq UltraScale+ MPSoC Embedded Design Tutorial* (UG1209). For PetaLinux, use 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 could probe and work properly. Create a new node for 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 node are listed and described in the following table.

A device tree configuration sample is shown below:

```
amba {
            ...
            dpu {
                  compatible = "xilinx,dpu";
                  base-addr = <0x8f000000>;

                  dpucore {
                      compatible = "xilinx,dpucore";
                      interrupt-parent = <&intc >;
                      interrupts = <0x0 106 0x1 0x0 107 0x1>;
                      core-num = <0x3>;
                  };
            };
            ....
}
```

The parameters list in the following Table 14.

Table 14: Fields in Device Tree and Corresponding Description

Parameter	Description
dpu	Node entry for DPU device. Keep it as fixed and does not need to be modified.
dpu ->compatible	Fix value "xilinx,dpu".
dpu ->base-addr	DPU register address assigned in the hardware design.
dpucore->compatible	Fix value "xilinx,dpu".
dpucore->interrupt-parent	Point to interrupt control device.
dpucore->interrupts	Interrupt configuration to DPU IP cores. There are three fields for each DPU core, and the second one is for interrupt number. Modify the interrupt numbers according to your customized hardware environment. For above sample, pair "0x0 106 0x1" is for DPU core 0 with interrupt number 106, and pair "0x0 107 0x1" is for DPU core 1 with interrupt number 107. The other two fields "0x0" and "0x1" are fixed values and needn't to be changed.
dpucore->core-num	How much DPU core is implemented in the hardware.

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 fixed correspondingly, otherwise the DPU will not work properly.

Customizing and Generating the Core in Zynq-7000 Devices

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

1. *When integrating the DPU IP into a Zynq-7000 project, the Vivado project must be created as a new project with the target project part selected as a Zynq-7000 series chip. It is not allowed to just change the target part of a Vivado project with the DPU from Zynq MPSoC series to Zynq 7000 devices.*
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. The support for softmax in Zynq-7000 devices might be updated in a future version.*
3. *The max data width of AXI port in the processing system (PS) of Zynq-7000 is 64-bit. The data width of the DPU will be modified from a 128-bit to 64-bit if the project target is Zynq-7000 devices. When the data width of the AXI interface is changed, the instruction file must be regenerated by DNNC accordingly.*

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

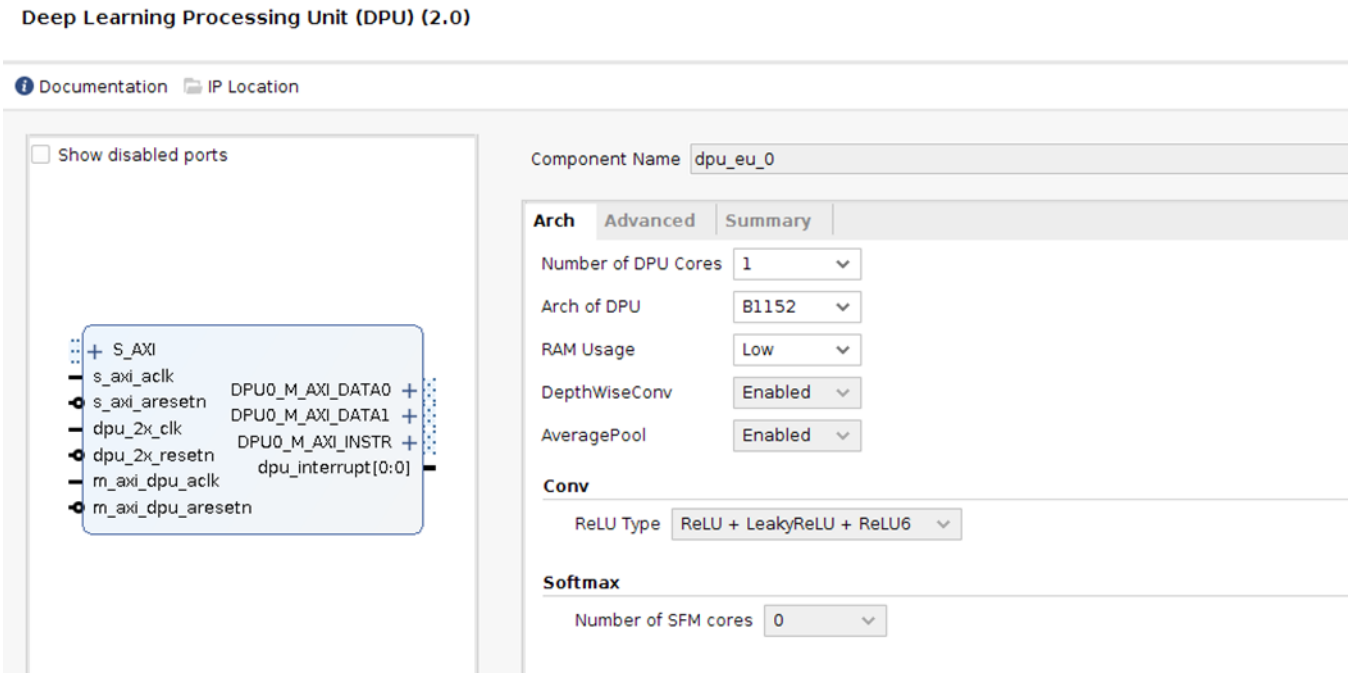

Figure 28: DPU Configuration in Zynq-7000 Devices

Introduction

The Xilinx® DPU targeted reference design (TRD) provides instructions on how to use DPU with a Xilinx SoC platform to build and run deep neural network applications. The TRD uses the Vivado® IP integrator flow for building the hardware design and 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 with the same flow. The TRD can be accessed by this link: https://www.xilinx.com/products/intellectual-property/dpu.html#overview.

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

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

DPU TRD Overview

The TRD creates an image classification application running a popular deep neural network model, Resnet50, on a Xilinx UltraScale+ MPSoC device. The overall functionality of the TRD is partitioned between the Processing System (PS) and Programmable Logic (PL), where DPU resides for optimal performance.

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

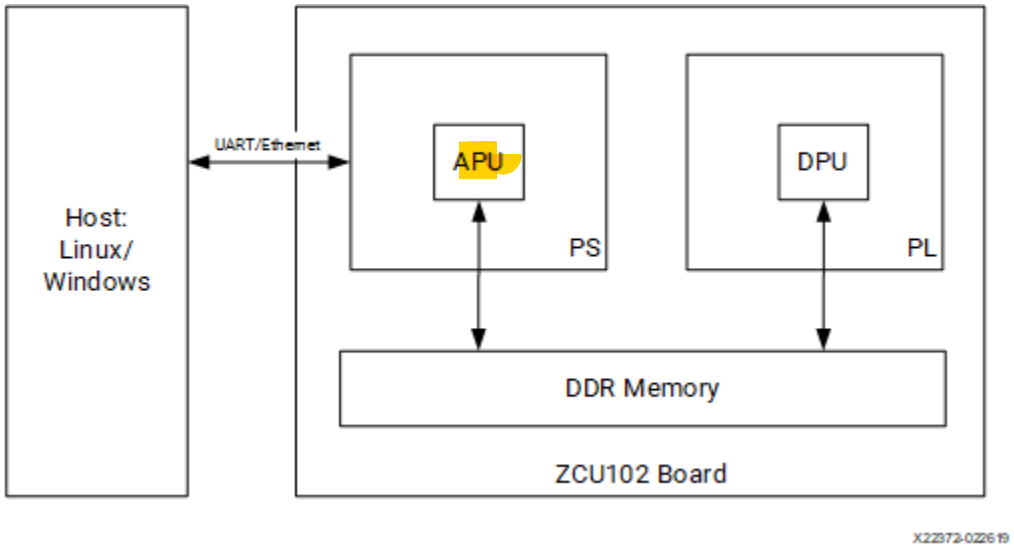

Figure 29: DPU TRD Overview

The application code used in the DPU TRD is from `main.cc` in the Resnet50 example in the DNNDK package. For more information about the DNNDK package, refer to the *DNNDK User Guide* (UG1327).

Requirements

The following summarizes the requirements of the TRD.

Target platforms:

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

Xilinx tools:

- Vivado Design Suite 2018.2
- PetaLinux 2018.2

Hardware peripherals:

- SD
- Ethernet
- UART

Linux or Windows host system:

- Serial terminal
- Network terminal

Design Files

Design files are in the following directory structure.

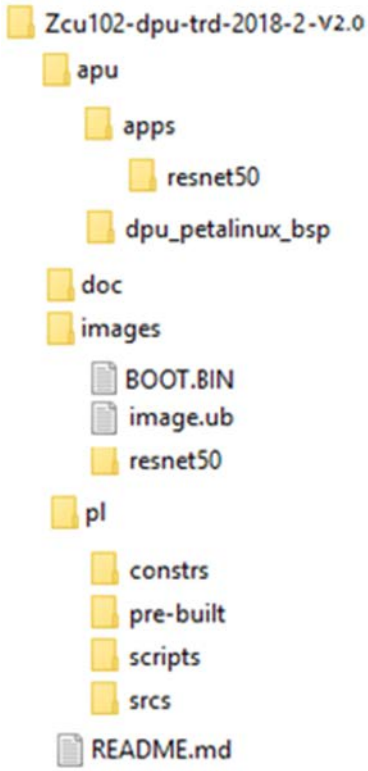

Figure 30: Directory Structure

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

Hardware Design Flow

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

Board Setup

The following figure shows the ZCU102 board with interfaces identified.

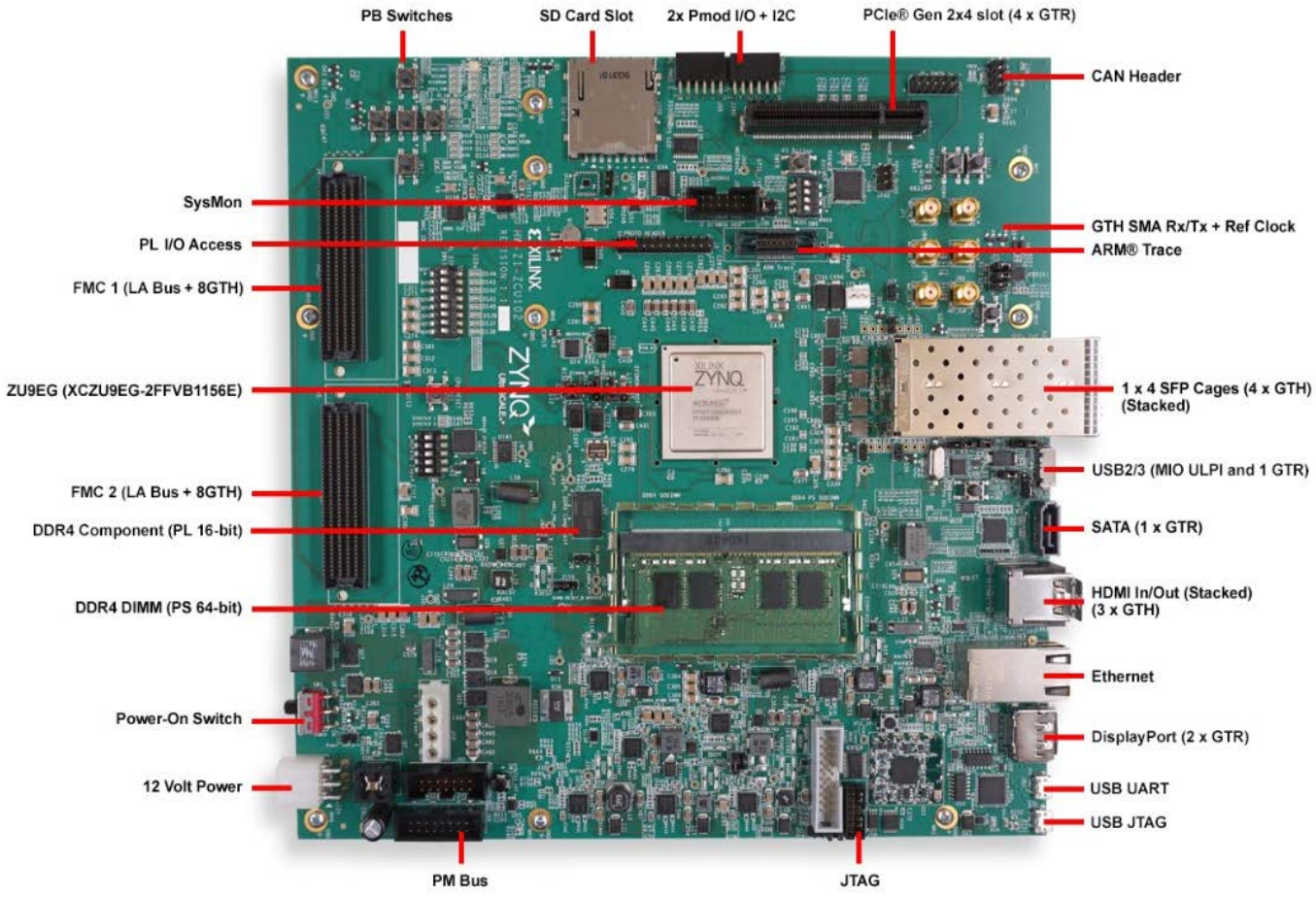

Figure 31: ZCU102 Board

ZCU102 Board Configuration

1. Connect the Micro USB cable into the ZCU102 Board Micro USB UART (J83) and the other end into an open USB port on the host PC. This cable is used for UART over USB communication.
2. Insert the SD card with the content of image folder into the SD card slot.
3. Set the SW6 switches and configure the boot setting to boot from SD as shown here.

Figure 32: Boot from SD

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 is about how to build the reference Vivado project with Vivado 2018.2. For information about setting up your Vivado environment, refer to the *Vivado Design Suite User Guide* (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 2018.2 > Vivado 2018.2**.
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 in the following figure.

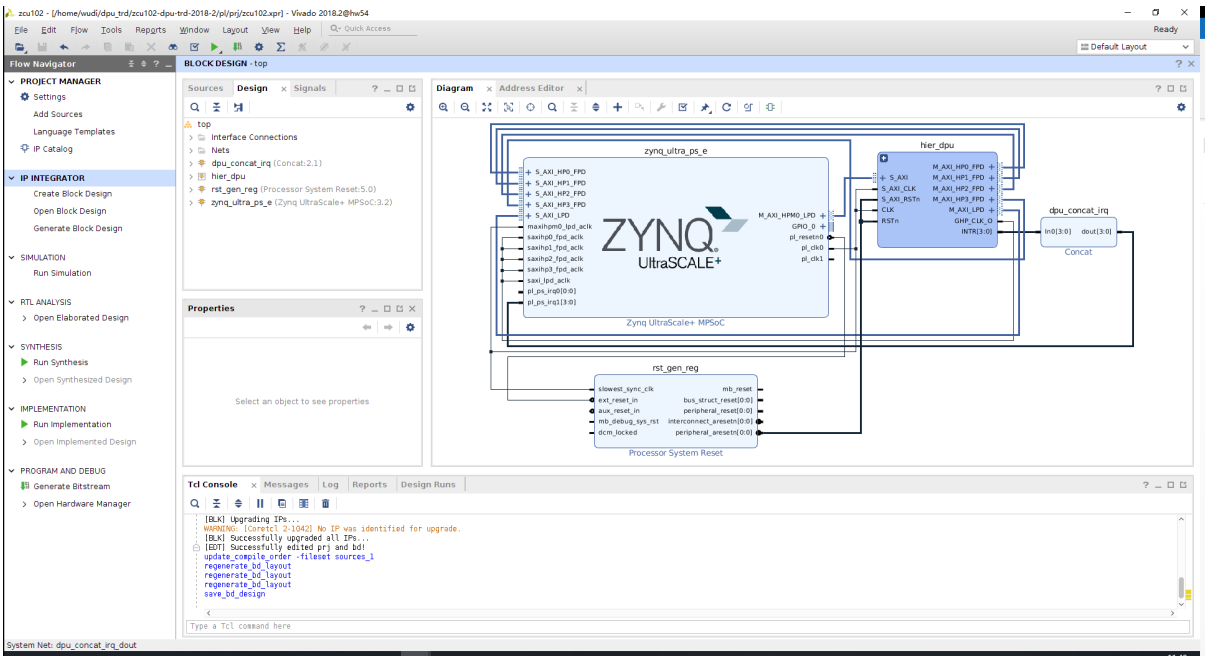

Figure 33: TRD Block Design

4. In the GUI, click **Generate Bitstream** to generate the bit file, as shown in the following figure.

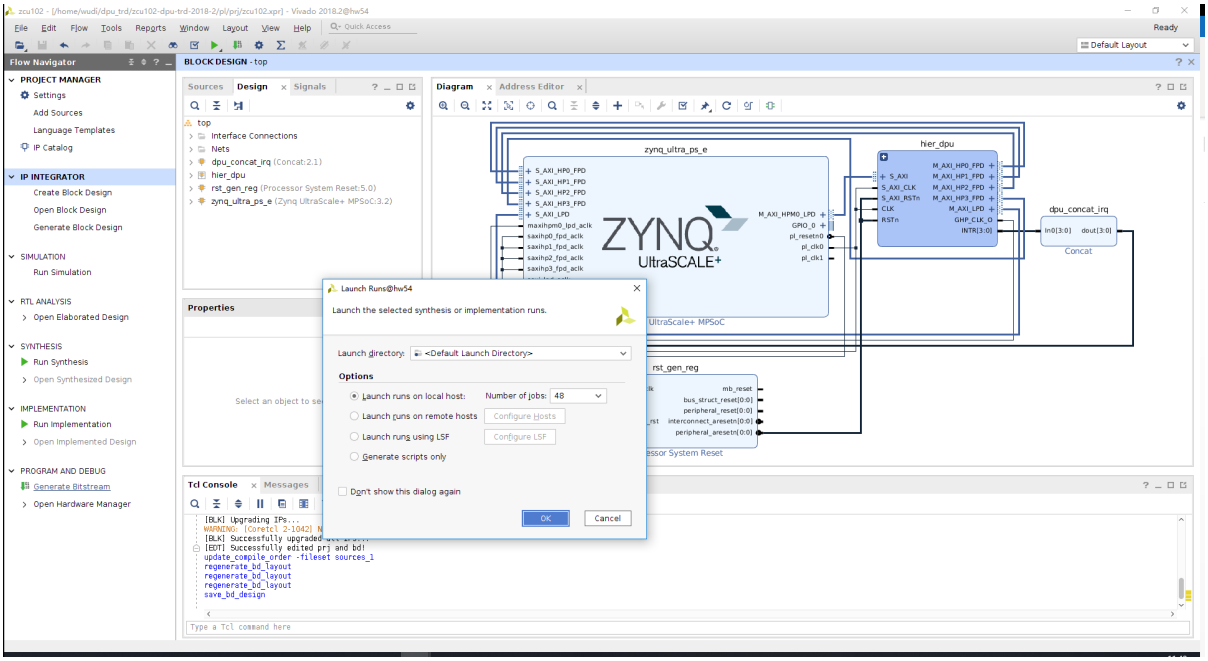

Figure 34: Generate Bitstream

DPU Configuration

The version of the DPU IP integrated in the TRD is DPU_v2.0. The default parameters of DPU in the reference design project is shown in the following figure.

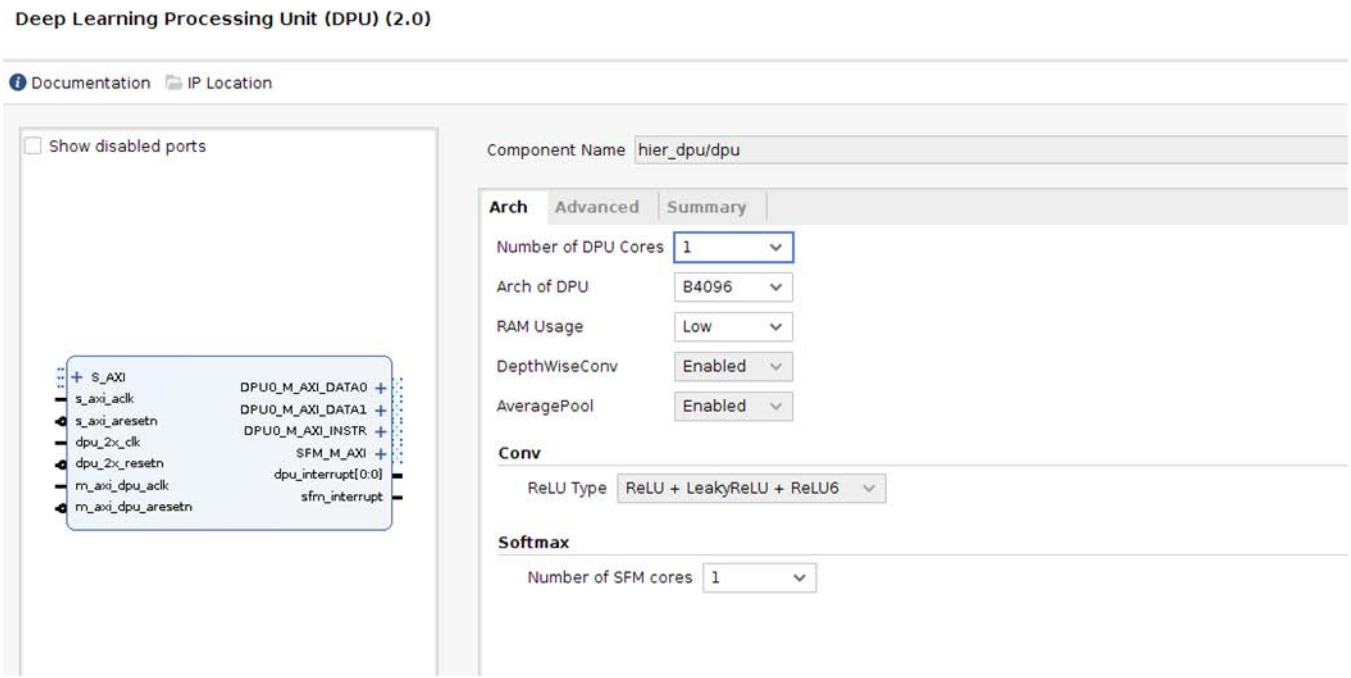

Figure 35: DPU Configuration Page

Those parameters of DPU can be configured in case of different resource requirements. For more information about DPU and its parameters, refer to Chapter 3: DPU Configuration.

Software Design Flow

This section shows how to generate BOOT.BIN using the PetaLinux build.

PetaLinux Design Flow

Install PetaLinux

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-v2018.2/petalinux-v2018.2-final/settings.sh
% echo $PETALINUX
% export TRD_HOME=<path/to/downloaded/zipfile>/zcu102-dpu-trd-2018-2
```

Build the PetaLinux Project

Use the following commands to create the PetaLinux project.

```
% cd $TRD_HOME/apu/dpu_petalinux_bsp
% petalinux-create -t project -s xilinx-dpu-trd-zcu102-v2018.2.bsp
% cd zcu102-dpu-trd-2018-2
% petalinux-config –-get-hw-description=$TRD_HOME/pl/pre-built --oldconfig
% petalinux-build
```

If pre-built design is needed, use the value of `––get-hw-description` as previous command.

If new generated/modified design is needed, please change the `$TRD_HOME/pl/pre-built` to `$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 SDK:

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

   You can use the pre-generated `sdk.sh` under `$TRD_HOME/apu/apps` or use `petalinux-build -s` to generate your own `sdk.sh`. If the permission is denied running `sdk.sh`, run `chmod 777 sdk.sh` to resolve. When the SDK 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:

   ```
   % 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 on the ZCU102 board is `root/root`.

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.
4. Insert the SD card into the ZCU102 and boot up the board. After the Linux boot, run as follows:

```
% cd /media/card/resnet50/
% ./resnet50
```

The screenshot is shown in the following figure.

The input images name is displayed in each line beginning with "Load image", the names are also the expected result of the input image. The predicted results of DPU is below, and the top-5 prediction probability of image classification are printed. If the Top-0 prediction results describe the names, 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!
##################################################
```

Figure 36: Running Results

References

These documents provide supplemental material useful with this product guide:

1. *DNNDK User Guide* (UG1327)
2. *Zynq UltraScale+ MPSoC Embedded Design Tutorial* (UG1209)
3. *PetaLinux Tools Documentation Reference Guide* (UG1144)
4. *ZCU102 Evaluation Board User Guide* (UG1182)

Please Read: Important Legal Notices