DNNDK User Guide

UG1327 (v1.5) June 7, 2019

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
06/07/2019 Version 1.5	
Chapter 1: Quick Start	• Updated the example package name for the DNNDK v3.0 release. • Added a note in the Installing the DNNDK Host Tools section. • Updated Figure 2.
Chapter 2: Copyright and Version	• Updated board information. • Updated component version information.
Chapter 7: Network Compilation	• Updated the `--dpu` parameter description.
04/29/2019 Version 1.4	
Chapter 1: Quick Start	• Removed all content related to DP-8020 and DP-N1 boards. • Updated installation information; added installing TensorFlow version details. • Updated information for Ultra96, ZCU102, and ZCU104 boards. • Removed DNNDK examples.
Chapter 2: Copyright and Version	• Updated board information. • Updated component version information.
Chapter 3: Upgrading and Porting	• Added information about Upgrading from v3.0.
Chapter 5: Network Deployment Overview	• Added TensorFlow model information. • Added information about network compression for TensorFlow version. • Added content about compiling Caffe ResNet-50 and TensorFlow ResNet-50. • Updated Output Kernels. • Updated Programming with DNNDK.
Chapter 6: Network Compression	• Updated DECENT Working Flow. • Updated information about using DECENT for Caffe version. • Added information about DECENT_Q for TensorFlow version.
Chapter 7: Network Compilation	• Updated information about using DNNC with both deployment and dummy compilation modes. • Updated details about compiling ResNet50.

Chapter 12: Programming APIs	Added v2.06 information to the following APIs: • dpuSetInputTensorInCHWInt8() • dpuSetInputTensorInCHWFP32() • dpuSetInputTensorInHWCInt8() • dpuSetInputTensorInHWCFP32() • dpuGetOutputTensorInCHWFP32() • dpuGetOutputTensorInHWCInt8() • dpuGetOutputTensorInHWCFP32() • dpuSetInputImage() • dpuSetInputImage2()
02/28/2019 Version 1.3	
Downloading DNNDK	Added DNNDK support for non-GPU host machines.
Setting Up the Host	
Chapter 3: Upgrading and Porting	
Network Compression	
DECENT Overview	
02/19/2019 Version 1.2	
Downloading DNNDK	Updated links to Xilinx product page with information about DNNDK-supported evaluation boards.
Setting Up the Evaluation Board	
Setting Up the DP-8020 Evaluation Board	
Setting Up the DP-N1 Board	
02/07/2019 Version 1.1	
Setting Up the Host	Updated information. Added liability notice.
01/22/2019 Version 1.0	
General updates	Initial Xilinx release.

XILINX

Table of Contents

Downloading DNNDK

The Deep Neural Network Development Kit (DNNDK) package can be freely downloaded after registration from the Xilinx website: https://www.xilinx.com/products/design-tools/ai-inference/edge-ai-platform.html.

Using a DNNDK-supported evaluation board is recommended to allow you to become familiar with the DNNDK toolchain. Refer to https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge for more details about the DNNDK-supported evaluation boards.

The package name for the DNNDK v3.0 release is `xilinx_dnndk_v3.0_yymm.tar.gz`, in which *yy* is the year and *mm* is the month. For example, the current released package name is `xilinx_dnndk_v3.0_1905.tar.gz`.

The directory structure for the DNNDK release package is shown in the following figure. In the rest of this document, `$dnndk_pkg` is used to represent the name of `xilinx_dnndk_v3.0` for simplicity.

The evaluation boards supported for this release are:

- Xilinx® ZCU102
- Xilinx ZCU104
- Avnet Ultra96

The `host_x86` folder contains files that need to be copied to the host computer running the 64-bit version of Ubuntu 14.04 LTS or Ubuntu 16.04 LTS.

The `<board_name>` folder contains files to be used on the evaluation board. The actual name of the folder corresponds to the DNNDK-supported boards: `Ultra96`, `ZCU102`, or `ZCU104`. Utility tools, Deep-learning Processor Unit (DPU) drivers, DPU runtime and development libraries are device-specific, and will be different for the various boards.

```
$dnndk_pkg
        | -- COPYRIGHT
        | -- host_x86
        |          | -- install.sh
        |          | -- models
        |          | -- pkgs
        |                |-- ubuntu14.04
        |                |-- ubuntu16.04
        | -- board_name
             | -- install.sh
             | -- pkgs
             | -- | -- bin
             | -- | -- driver
             | -- | -- include
             | -- | -- lib
```

Setting Up the Host

The "host_x86" folder contains the Deep Compression Tool (DECENT) and Deep Neural Network Compiler (DNNC) host tools, which allow neural networks to be optimized and accelerated on the DPU inference engine.

IMPORTANT: *Carefully read the capitalized text that follows.*

NOTICE: BY RUNNING THE COMMAND BELOW, YOU WILL CAUSE TO BE DOWNLOADED AND INSTALLED ON YOUR MACHINE CERTAIN THIRD-PARTY OPEN SOURCE SOFTWARE GOVERNED EXCLUSIVELY BY OPEN SOURCE LICENSES. BY RUNNING THE COMMAND BELOW AND DOWNLOADING AND USING THE THIRD-PARTY OPEN SOURCE SOFTWARE, YOU AGREE ON BEHALF OF YOURSELF AND YOUR EMPLOYER (IF APPLICABLE) TO BE BOUND BY THIS INFORMATION AND THE OPEN SOURCE LICENSE AGREEMENTS APPLICABLE TO THE SPECIFIC THIRD-PARTY SOFTWARE INSTALLED ON YOUR MACHINE. IF YOU OR YOUR EMPLOYER DO NOT AGREE TO ALL OF THE INFORMATION AND APPLICABLE OPEN SOURCE LICENSE AGREEMENTS, DO NOT RUN THIS COMMAND.

THE LIST OF THIRD-PARTY OPEN SOURCE SOFTWARE, AND THE COMMAND IS MADE AVAILABLE "AS-IS" AND 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.

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, YOUR USE OF THIS THIRD-PARTY OPEN SOURCE SOFTWARE (INCLUDING YOUR USE OF THE COMMAND), 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.

Installing the GPU Platform Software

The current DNNDK release can be used on the X86 host machine with or without GPU. With GPU support, DECENT is able to run faster.

If GPU is available in the X86 host machine, install the necessary GPU platform software in accordance with your GPU product documentation. Ensure all versions are compatible with the version of DNNDK. For questions, contact your account manager or submit a support case to https://www.xilinx.com.

Caffe Version: Installing Dependent Libraries

Run the following command to install the dependent libraries required by Caffe v1.0.

```
$ apt-get install -y --force-yes build-essential autoconf libtool libopenblas-
dev libgflags-dev libgoogle-glog-dev libopencv-dev protobuf-compiler libleveldb-
dev liblmdb-dev libhdf5-dev libsnappy-dev libboost-all-dev libssl-dev
```

TensorFlow Version: Installing with Anaconda

The DECENT_Q for TensorFlow supports operating system of Ubuntu 14.04 and 16.04 platforms, and have both CPU and GPU versions. Xilinx provides installation packages for common environments listed below, users can download the right package. If you are working under other environments, contact Xilinx.

Table 1: Supported 64-bit Host OS Platforms and Required Packages

OS Platform	Version	Required Packages	Python	Package Name
Ubuntu 14.04 LTS	GPU	CUDA 8.0 (GA2), cuDNN 7.0.5	2.7	tensorflow_gpu-1.9.0-cp27-cp27mu-linux_x86_64.whl
			3.6	tensorflow_gpu-1.9.0-cp36-cp36m-linux_x86_64.whl
	CPU Only	None	2.7	tensorflow-1.9.0-cp27-cp27mu-linux_x86_64.whl
			3.6	tensorflow-1.9.0-cp36-cp36m-linux_x86_64.whl
Ubuntu 16.04 LTS	GPU	CUDA 9.0, cuDNN 7.0.5	2.7	tensorflow_gpu-1.9.0-cp27-cp27mu-linux_x86_64.whl
		CUDA 9.0, cuDNN 7.0.5	3.6	tensorflow_gpu-1.9.0-cp36-cp36m-linux_x86_64.whl
	CPU Only	None	2.7	tensorflow-1.9.0-cp27-cp27mu-linux_x86_64.whl
			3.6	tensorflow-1.9.0-cp36-cp36m-linux_x86_64.whl

Anaconda provides the conda utility to create a virtual environment. It is recommended to create a new anaconda environment and use the `pip install` command to install DECENT_Q in case of affecting the existing TensorFlow versions. Once you have anaconda installed (see https://www.anaconda.com for instructions), use the following commands to install DECENT_Q inside a virtual environment:

```
$ conda create -n decent pip python=2.7 # or python=3.6, etc.
$ source activate decent
(decent)$ pip install ${DECENT_Q_TF_PKG} # Select the right installation package
for your environment
(decent)$ pip install numpy opencv-python sklearn scipy progressbar2
```

where DECENT_Q_TF_PKG is the downloaded installation package.

Note: You can run the tool with `--help` to validate your installation:

```
(decent)$ decent_q  --help
```

Installing the DNNDK Host Tools

After downloading and unpacking the DNNDK package, execute the `./install.sh <board_name>` command under the `host_x86` folder to install the DECENT and DNNC tools on the host. The `<board_name>` must be replaced with the evaluation board to be used (`Ultra96`, `ZCU104`, or `ZCU102` according to the evaluation board to be used.) With `board_name` specified, the suited version DNNC compiler will be automatically installed for the evaluation board.

Note: There are two version DNNC binaries available for Ubuntu 14.04 and 16.04 individually under the host_x86 folder. `dnnc-dpu1.4.0` is for DPU configured with low RAM usage, and `dnnc-dpu1.4.0.1` is for DPU configured with high RAM usage. Running `install.sh` will automatically install the right version of the DNNC binary for the corresponding evaluation board.

Setting Up the Evaluation Board

Supported Evaluation Boards

The following sections describe the evaluation boards supported by DNNDK. The Ultra96 is intended for evaluating low-power, low-throughput deep learning applications. The Xilinx ZCU102 and ZCU104 are geared towards higher throughput deep learning inference requiring low latency.

The SD card system image files for all DNNDK supported evaluation boards are available for download from https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge. Before trying DNNDK v3.0 on the evaluation boards, you must download the updated version image file; otherwise, DNNDK package cannot work on the previous version image file. The throughput performance in FPS for each DNNDK sample is listed for all these evaluation boards.

Note: Before installing DNNDK v3.0 into your evaluation board, download the updated version image file from https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge.

Setting Up the Ultra96 Board

Ultra96 is an Arm®-based, Xilinx Zynq® UltraScale+™ MPSoC development board based on the Linaro 96Boards specification. The 96Board specifications are open and define a standard board layout for development platforms that can be used by software application, hardware device, kernel, and other system software developers. Ultra96 represents a unique position in the 96Boards community with a wide range of potential peripherals and acceleration engines in the programmable logic that is not available from other offerings. The hardware documentation for Ultra96 is available for download on http://zedboard.org/product/ultra96.

The main connectivity interfaces for Ultra96 are shown in the following figure.

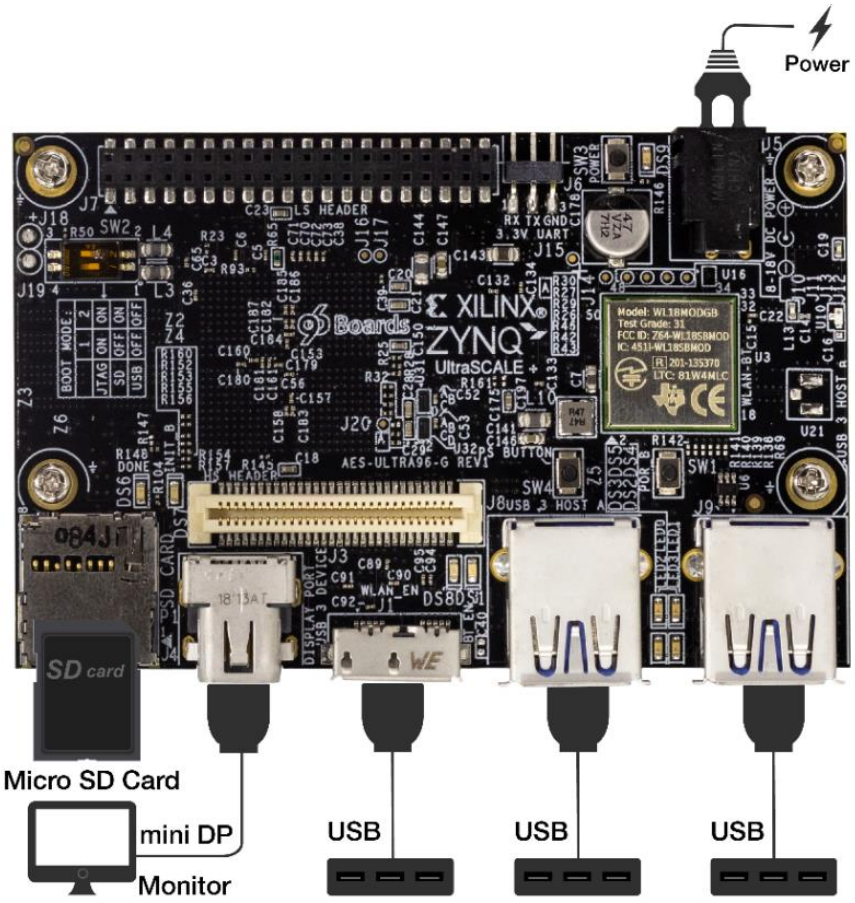

Figure 1: Ultra96 Evaluation Board and Peripheral Connections

The DPU signature information is shown in the following figure. One B2304F DPU core running at 260 MHz is implemented on the Xilinx ZU3 device.

```
root@ultra96:~# dexplorer -w
[DPU IP Spec]
IP  Timestamp    : 2019-05-24 15:30:00
DPU Core Count   : 1

[DPU Core List]
DPU Core         : #0
DPU Enabled      : Yes
DPU Arch         : B2304F
DPU Target       : v1.4.0
DPU Freqency     : 255 MHz
DPU Features     : Avg-Pooling, LeakyReLU/ReLU6, Depthwise Conv
```

Figure 2: Ultra96 DPU Signature Viewed with DExplorer

Refer to Figure 3 and Figure 4 to set up a WiFi network connection for the Ultra96 board.

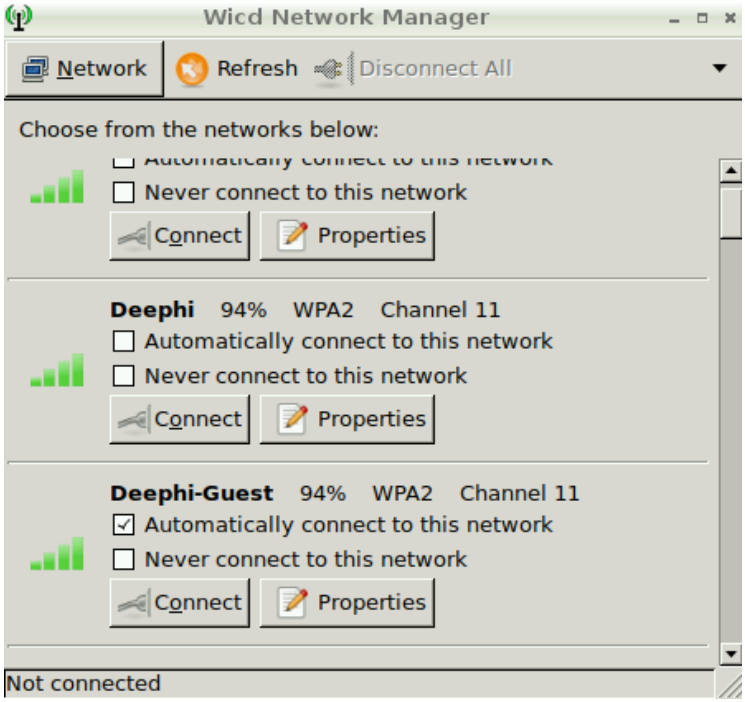

Figure 3: Ultra96 WiFi Connection

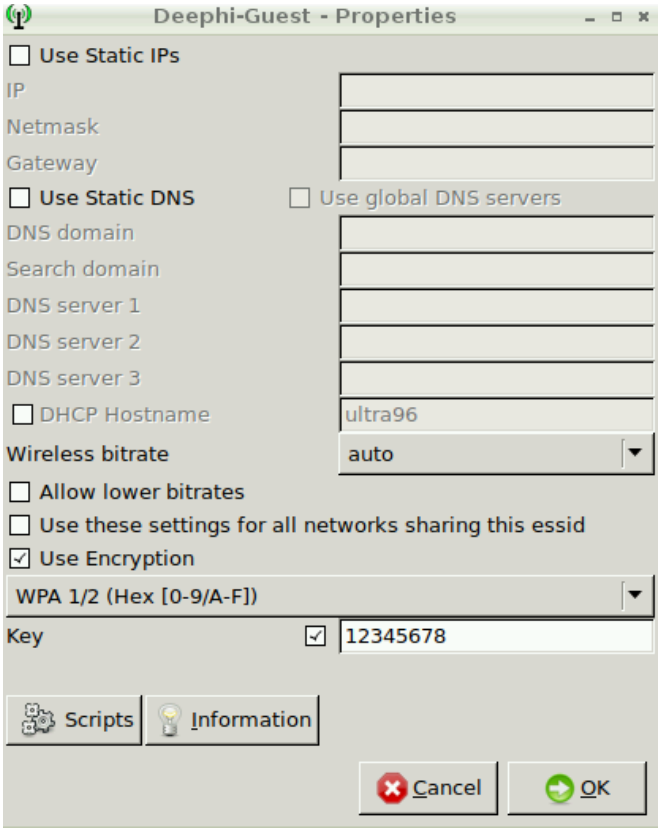

Figure 4: Ultra96 WiFi Configuration

Setting Up the ZCU102 Evaluation Board

The Xilinx ZCU102 evaluation board uses the mid-range ZU9 UltraScale+ device to enable you to jumpstart your machine learning applications. For more information on the ZCU102 board, refer to the ZCU102 product page on the Xilinx website: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html.

The main connectivity interfaces for ZCU102 are shown in the following figure.

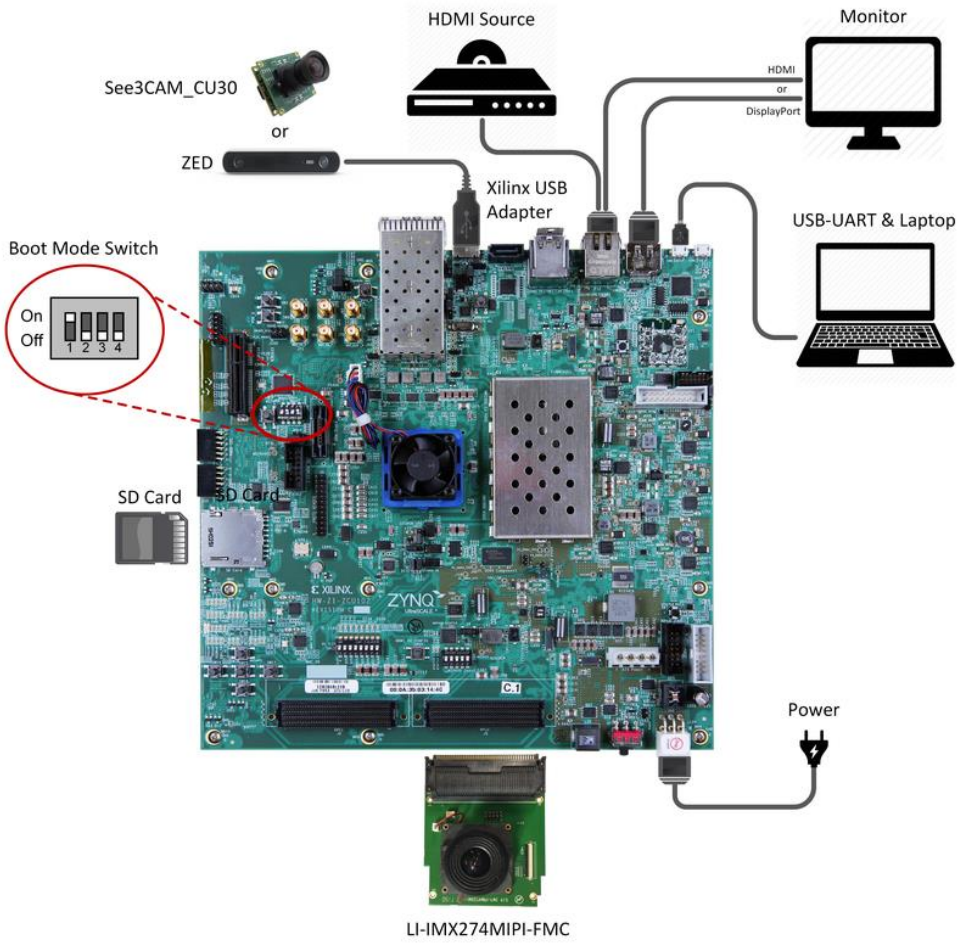

Figure 5: Xilinx ZCU102 Evaluation Board and Peripheral Connections

The DPU signature information is shown in the following figure. Triple B4096F DPU cores running at 333MHz are implemented on the Xilinx ZU9 device.

```
root@zcu102:~# dexplorer -w
[DPU IP Spec]
IP  Timestamp   : 2019-04-16 11:15:00
DPU Core Count  : 3

[DPU Core List]
DPU Core        : #0
DPU Enabled     : Yes
DPU Arch        : B4096F
DPU Target      : v1.4.0
DPU Freqency    : 333 MHz
DPU Features    : Avg-Pooling, LeakyReLU/ReLU6, Depthwise Conv
DPU Core        : #1
DPU Enabled     : Yes
DPU Arch        : B4096F
DPU Target      : v1.4.0
DPU Freqency    : 333 MHz
DPU Features    : Avg-Pooling, LeakyReLU/ReLU6, Depthwise Conv
DPU Core        : #2
DPU Enabled     : Yes
DPU Arch        : B4096F
DPU Target      : v1.4.0
DPU Freqency    : 333 MHz
DPU Features    : Avg-Pooling, LeakyReLU/ReLU6, Depthwise Conv

[DPU Extension List]
Extension Softmax
Enabled         : Yes
```

Figure 6: Xilinx ZCU102 DPU Signature Viewed with DExplorer

Setting Up the ZCU104 Evaluation Board

The Xilinx ZCU104 evaluation board uses the mid-range ZU7 UltraScale+ device to enable you to jumpstart your machine learning applications. For more information on the ZCU104 board, refer to the Xilinx site https://www.xilinx.com/products/boards-and-kits/zcu104.html.

The main connectivity interfaces for ZCU104 are shown in the following figure.

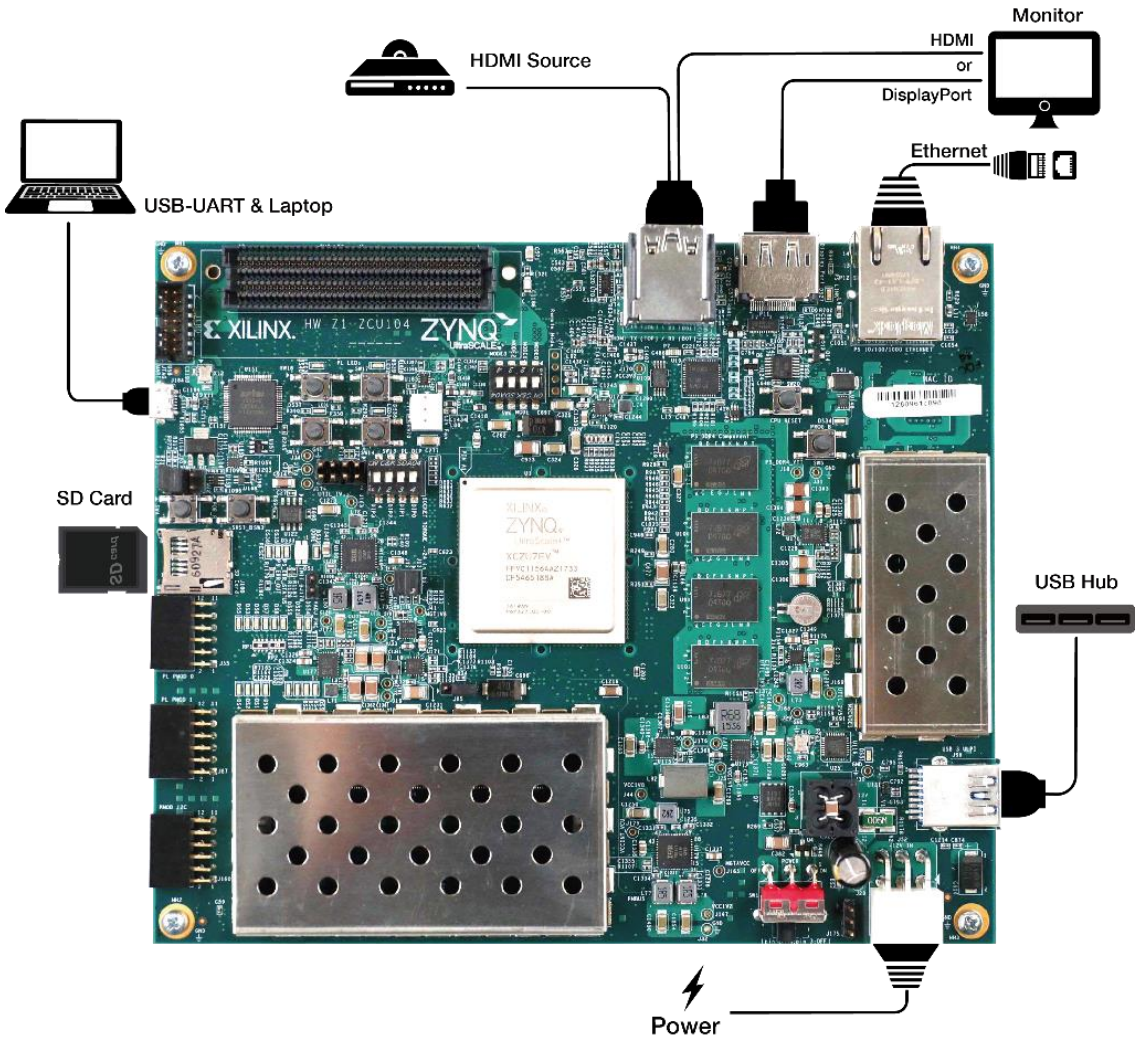

Figure 7: Xilinx ZCU104 Evaluation Board and Peripheral Connections

The DPU signature information is shown in the following figure. Dual B4096F DPU cores running at 333 MHz are implemented on the Xilinx ZU7 device.

```
root@zcu104:~# dexplorer -w
[DPU IP Spec]
IP  Timestamp   : 2019-04-18 15:30:00
DPU Core Count  : 2

[DPU Core List]
DPU Core        : #0
DPU Enabled     : Yes
DPU Arch        : B4096F
DPU Target      : v1.4.0
DPU Freqency    : 333 MHz
DPU Features    : Avg-Pooling, LeakyReLU/ReLU6, Depthwise Conv
DPU Core        : #1
DPU Enabled     : Yes
DPU Arch        : B4096F
DPU Target      : v1.4.0
DPU Freqency    : 333 MHz
DPU Features    : Avg-Pooling, LeakyReLU/ReLU6, Depthwise Conv
```

Figure 8: Xilinx ZCU104 DPU signature viewed with DExplorer

In the following sections, ZCU102 is used as an example to show the steps to setup the DNNDK running environment on evaluation boards.

Flash the OS Image to the SD Card

One suggested software application for flashing the SD card is Etcher. It is a cross-platform tool for flashing OS images to SD cards, available for Windows, Linux and Mac systems. The following example uses Windows.

1. Download Etcher from: https://etcher.io/ and save the file as shown in the following figure.

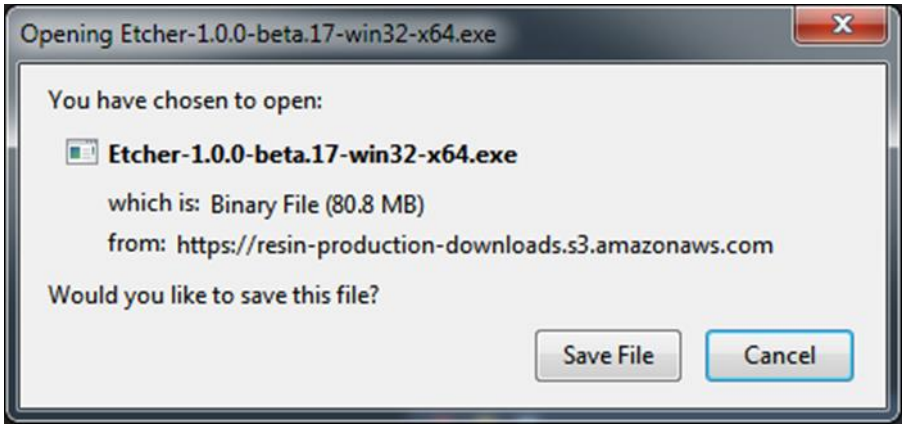

Figure 9: Saving the Etcher File

2. Install Etcher, as shown in the following figure.

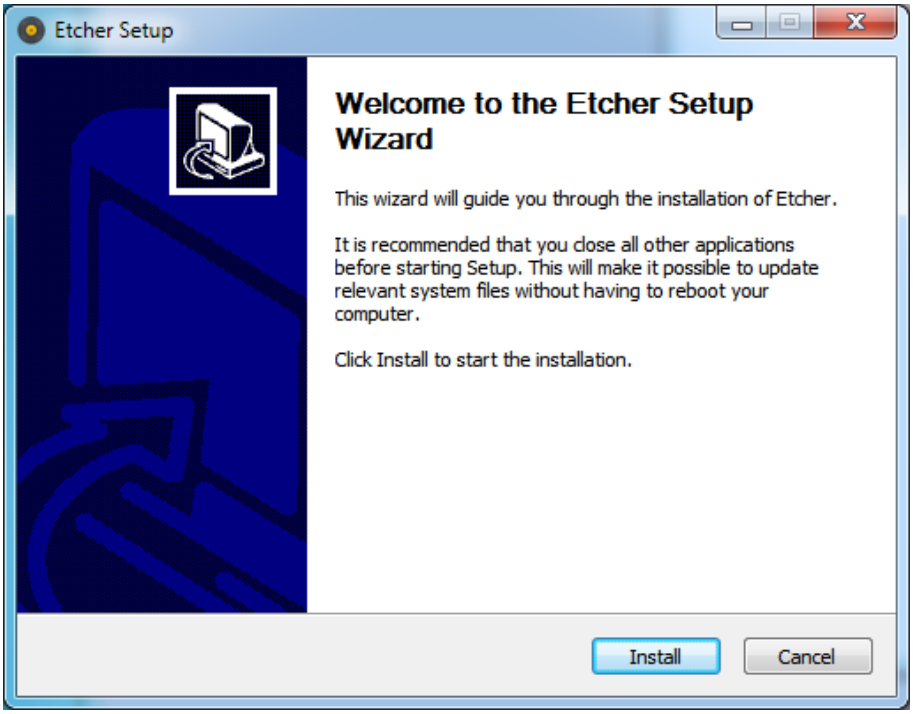

Figure 10: Install Etcher

3. Eject any external storage devices such as USB flash drives and backup hard disks. This makes it easier to identify the SD card. Then, insert the SD card into the slot on your computer, or into the reader.
4. Run the Etcher program by double clicking on the Etcher icon shown in the following figure, or select it from the **Start** menu.

Figure 11: Etcher Icon

Etcher launches, as shown in the following figure.

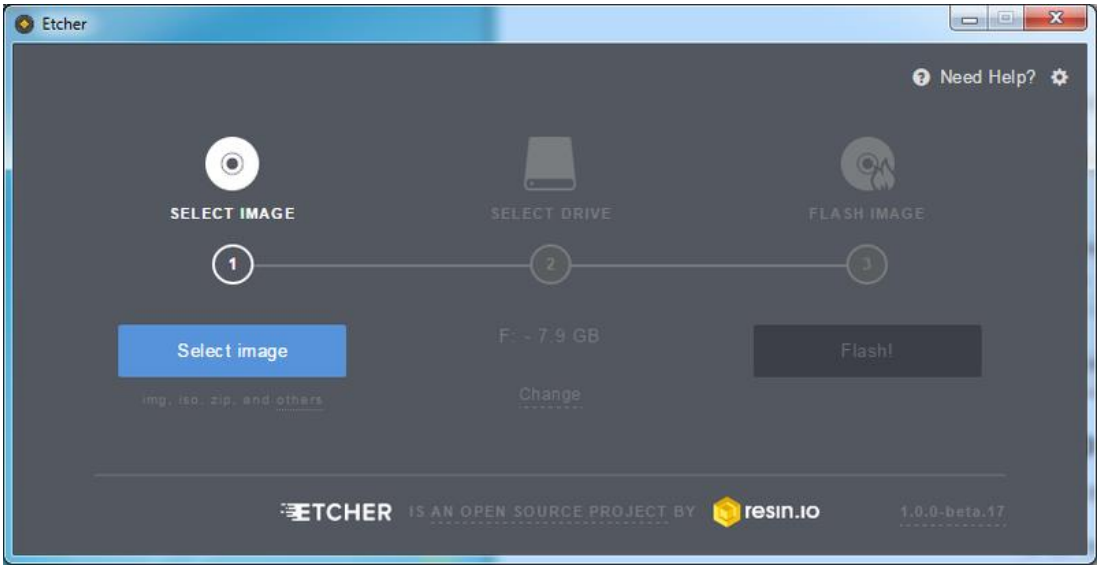

Figure 12: The Etcher Interface

5. Select the image file by clicking Select Image. You can select a .zip or .gz compressed file.
6. Etcher tries to detect the SD drive. Verify the drive designation and check the image size to make sure that it is correct.
7. Click **Flash**.

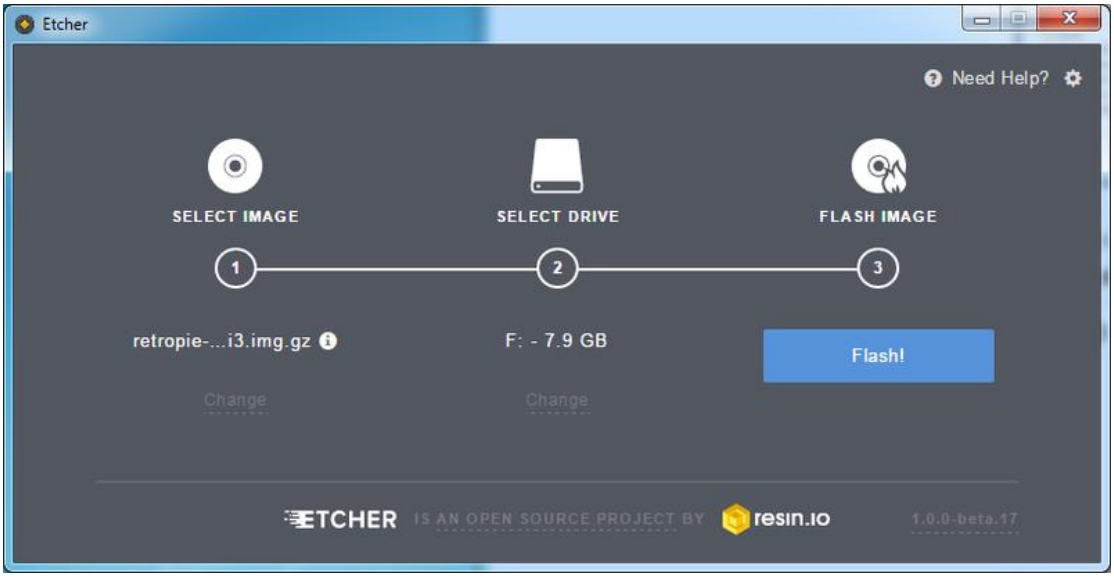

Figure 13: Flash the Card

Booting the Evaluation Board

In this section, a ZCU102 board is used to illustrate how to boot a DNNDK evaluation board. Follow the steps below to boot the evaluation board.

1. Connect the power supply (12V ~ 5A).

2. Connect the UART debug interface to the host and other peripherals as required.
3. Turn on the power, and wait for the system to boot.
4. Log into the system.
5. The system needs to perform some configurations for its first boot. Then reboot the board for these configurations to take effect.

Accessing the Evaluation Board

There are three ways to access the ZCU102 board: via UART, ethernet, or standalone.

Configuring UART

Aside from monitoring the boot process and viewing kernel messages for debugging, you can log into the board through the UART. The configuration parameters of the UART are shown below. A screenshot of a sample boot is shown in the following figure. Log into the system with username "root" and password "root".

- baud rate: 115200 bps
- data bit: 8
- stop bit: 1
- no parity

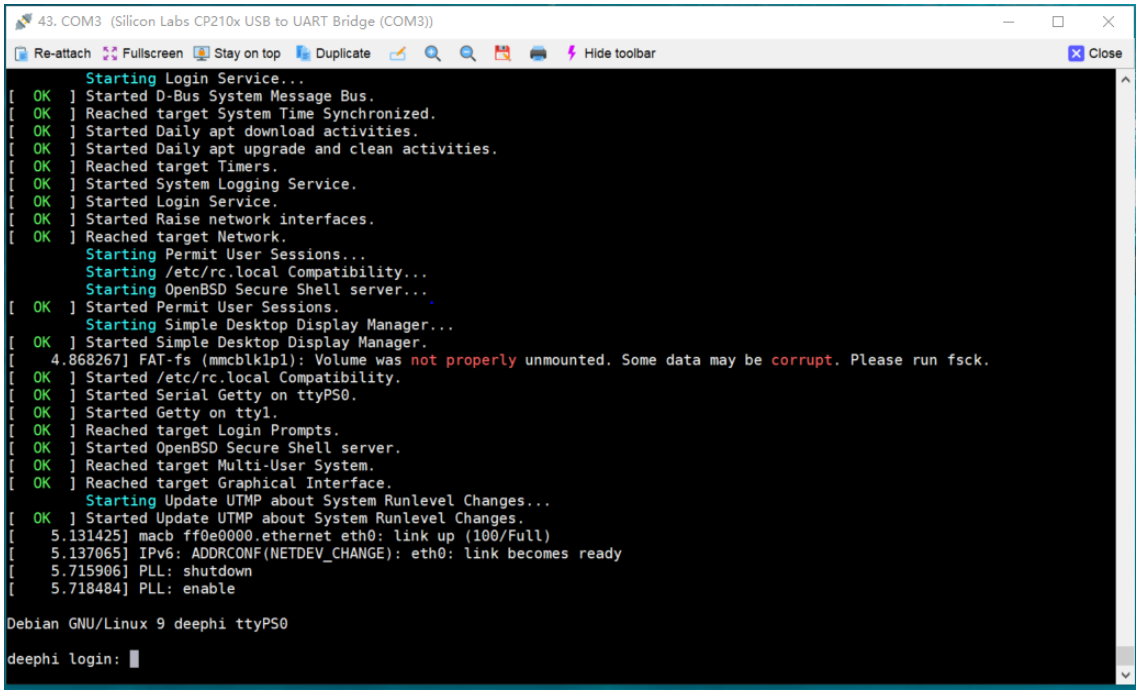

Figure 14: Example of Boot Process

Note: On a Linux system, you can use Minicom to connect to the target board directly; for a Windows system, a USB to UART driver is needed before connecting to the board through a serial port.

Using the Ethernet Interface

The ZCU102 board has an Ethernet interface, and SSH service is enabled by default. You can log into the system using an SSH client after the board has booted.

```
$ ssh root@10.10.134.230
root@10.10.134.230's password:
Linux zcu102 4.14.0-xilinx-v2018.3 #1 SMP Wed Apr 24 01:54:49 UTC 2019 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Apr 29 11:25:31 2019 from 10.10.0.1
root@zcu102:~#
```

Figure 15: Logging into the Evaluation Board Using SSH

Use the `ifconfig` command via the UART interface to set the IP address of the board, then use the SSH client to log into the system.

Using the ZCU102 as a Standalone Embedded System

The ZCU102 board allows a keyboard, mouse, and monitor to be connected. After a successful boot, a Linux GUI desktop is displayed. You can then access the board as a standalone embedded system.

Figure 16: Standalone Linux GUI Screen

Copying DNNDK Tools to the Evaluation Board

With an ethernet connection established, you can copy the DNNDK package from the host machine to the evaluation board.

The steps below illustrate how to setup DNNDK running environment for ZCU102 provided that DNNDK package is stored on a Windows system.

1. Download and install MobaXterm on Windows system. MobaXterm is a terminal for Windows, and is available online at https://mobaxterm.mobatek.net/.
2. Launch MobaXterm and click **Start local terminal** to open a terminal where the filesystem of Windows can be accessed.

3. Extract and copy the package for the board.

   For example, suppose that the DNNDK package is located under the root directory of disk D. In this case, use the following commands to extract and copy the package for the ZCU102 board with IP address 192.168.0.10:

   ```
   cd d:
   tar -xzvf xilinx_dnndk_v3.0_1904
   cd  xilinx_dnndk_v3.0_1904/
   scp -r ./ZCU102  root@192.168.0.10:~/
   ```

4. On the ZCU102 board, change to the *~/ZCU102/* directory and run *install.sh*. The following messages display if the installation completes successfully.

   ```
   Begin to install Xilinx DNNDK ...
   Install DPU driver ...
   Install tools, runtime & libraries ...
   Complete installation successfully.
   ```

Note: Installing DNNDK v3.0 will automatically replace previous releases. There is no need to manually uninstall previous versions.

Support

You can visit the DNNDK community forum on the Xilinx website https://forums.xilinx.com/t5/Deephi-DNNDK/bd-p/Deephi for topic discussions, knowledge sharing, FAQs, and requests for technical support.

Copyright

Some third-party source code is used in the Deep Neural Network Development Kit (DNNDK) toolchain. Refer to the file `$dnndk_pkg/COPYRIGHT` for the related copyright declaration.

Version

Host Package

The "host_x86" directory in the DNNDK release package contains the host side tools Deep Compression Tool (DECENT) and DNNC. The version information is shown below.

If you encounter issues while using DECENT or DNNC, contact the DNNDK support team and provide the version information. You can find this information using the command `decent --version` and `dnnc --version`.

Ultra96 Board Version Information

```
decent-ubuntu16.04-cuda8.0-cudnn7.05 version 1.2.5
Build Label Apr 27 2019 22:57:56
(c) Copyright 2016 - 2019 Xilinx, Inc. All rights reserved.

dnnc version v2.04
DPU Target : v1.4.0.1
Build Label: May 21 2019 14:02:59
Copyright @2019 Xilinx Inc. All Rights Reserved.
```

ZCU102 Board Version Information

```
decent-ubuntu16.04-cuda8.0-cudnn7.05 version 1.2.5
Build Label Apr 27 2019 22:57:56
(c) Copyright 2016 - 2019 Xilinx, Inc. All rights reserved.

dnnc version v2.04
DPU Target : v1.4.0
Build Label: May 21 2019 13:58:53
Copyright @2019 Xilinx Inc. All Rights Reserved.
```

ZCU104 Board Version Information

```
decent-ubuntu16.04-cuda8.0-cudnn7.05 version 1.2.5
Build Label Apr 27 2019 22:57:56
(c) Copyright 2016 - 2019 Xilinx, Inc. All rights reserved.
```

```
dnnc version v2.04
DPU Target : v1.4.0.1
Build Label: May 21 2019 14:02:59
Copyright @2019 Xilinx Inc. All Rights Reserved.
```

NOTE: Currently, the host machine tool DNNC version varies among different DNNDK evaluation boards. It will be unified into a single release in the future.

Target Package

The package for the target Deep-learning Processor Unit (DPU) evaluation board contains several components: DExplorer, DSight, DPU driver, and runtime N^2Cube. The component versions for each evaluation board are shown below. If you encounter issues while running DNNDK applications on evaluation boards, contact the DNNDK support team and provide the version information. You can find this information using the command `dexplorer -v`.

Ultra96 Component Versions

```
DNNDK version  3.0
Copyright @ 2018-2019 Xilinx Inc. All Rights Reserved.

DExplorer version 1.5
Build Label: Apr 25 2019 15:54:41

DSight version 1.4
Build Label: Apr 25 2019 15:54:41

N2Cube Core library version 2.3
Build Label: Apr 25 2019 15:54:55

DPU Driver version 2.2.0
Build Label: May 20 2019 11:25:15
```

ZCU102 Component Versions

```
DNNDK version  3.0
Copyright @ 2018-2019 Xilinx Inc. All Rights Reserved.

DExplorer version 1.5
Build Label: Apr 25 2019 09:50:06

DSight version 1.4
Build Label: Apr 25 2019 09:50:06

N2Cube Core library version 2.3
Build Label: Apr 25 2019 09:50:20

DPU Driver version 2.2.0
Build Label: May 20 2019 11:00:06
```

ZCU104 Component Versions

```
DNNDK version  3.0
Copyright @ 2018-2019 Xilinx Inc. All Rights Reserved.

DExplorer version 1.5
Build Label: Apr 25 2019 10:23:27

DSight version 1.4
Build Label: Apr 25 2019 10:23:27

N2Cube Core library version 2.3
Build Label: Apr 25 2019 10:23:41

DPU Driver version 2.2.0
Build Label: May 20 2019 11:29:59
```

The key improvements and changes in each release of the Deep Neural Network Development Kit (DNNDK) toolchain since the first version 1.07 published in Oct. 2017 are summarized in this chapter. A guide is provided to allow you to port Deep-learning Processor Unit (DPU) applications from a previous DNNDK version to the latest version.

Since v3.0

The most important feature of DNNDK v3.0 is that popular deep learning framework TensorFlow is officially supported by DNNDK toolchains.

DeePhi DP-N1 AI Box and Ultra96 are removed from the evaluation board list enabled by DNNDK since this release.

Toolchain Changes

DECENT

TensorFlow framework is supported by DECENT. And since DNNDK v3.0, both CPU version and GPU version DECENT binary tools are released for user convenience. CPU version DECENT can perform quantization on CPU-only host machine without GPU installed; however, this requires longer running time.

Added support for fully-grouped deconvolution and GSTiling layers for Caffe version.

Added support for un-matched caffemodel and prototxt for Caffe version.

Removed dependency for YAML and used protobuf file for `-ignore_nodes_file` option for Caffe version.

DNNC

Caffe and TensorFlow are supported by one single DNNC binary tool. For TensorFlow, it is enabled by the new introduced options `--parser` and `--frozen_pb`. Refer to Using DNNC for details.

Example Changes

All samples are removed from DNNDK release package. For more reference, download the Xilinx® AI SDK.

Since v2.08

Two new evaluation boards DeePhi DP-N1 AI Box and Ultra96 have been enabled since the v2.08 release. With the existing three boards DP-8020, ZCU102, and ZCU104, there are now a total of five evaluation boards available. Meanwhile, DNNDK toolchains are under the continuous enhancements to improve performance of the DPU, to make tools easy to use, to make DNNDK more suitable for production environment, and so on.

Toolchain Changes

DECENT

The dependency of Nvidia NCCL library is removed from Deep Compression Tool (DECENT). There is no longer a need to install NCCL on an x86 host machine. DECENT supports an X86 host machine without GPU.

DNNC

The new option `--abi` was introduced since DNNC v2.03 for the purpose of DNNDK forward compatibility. Refer to Using DNNC for a detailed description.

Exception Handling Changes

A new exception handling mode for N^{2}Cube runtime APIs is introduced. For the former releases, N^{2}Cube will output the error message and terminate the running of DNNDK application when any error occurs.

Since v2.08, N2Cube runtime APIs will return corresponding error code and will not exit in case of errors. The API callers must take charge of the following exception handling process, such as logging the error message with API `dpuGetExceptionMessage()`, resource release, and so on.

To keep forward compatibility, the former exception handling manner is the default mode, but it can be changed to new mode using `dpuSetExceptionMode()`.

API Changes

The following four new APIs were introduced into the libn2cube library. Refer to Chapter 12: Programming APIs for details.

- `dpuRunSoftmax()`
- `dpuSetExceptionMode()`
- `dpuGetExceptionMode()`
- `dpuGetExceptionMessage()`

Example Changes

Three new DNNDK examples were added to demonstrate the DPU capabilities and scalabilities, including:

- MobileNet
- Multithreaded MobileNet
- ADAS detection with YOLO-v3 network model

Since v2.07

Only one evaluation board was supported in previous releases. Beginning with the 2.07 release, the supported evaluation boards are DP-8020, ZCU102, and ZCU104.

Toolchain Changes

DNNC

The following minor updates were made to DNNC to improve its ease of use.

1. The kernel graph description file `<net_name>_kernel_graph.jpg` in JPEG format is generated by DNNC if the host system has the `dot` command installed; otherwise, the original `gv` format file with the same name is generated.
2. Instead of simply printing the input/output node name for DPUKernel in the kernel description, the input/output index for node name is also added in `node_name(index)` format. When setting input or getting output using the API provided by N^{2}Cube, the `node_name` and `index` act as unique identifiers.
3. A `dump` option has been added to DNNC for error probing (see Chapter 7: Network Compilationfor more details).

Since v2.06

Toolchain Changes

DECENT

- Added more operators:
  - Dilation convolution
  - Deconvolution
  - Flatten
  - Concat
  - Scale
  - Reorg
- Added more layer fusing patterns:
  - Fuse BatchNorm + Scale + Convolution into Convolution layer
  - Fuse standalone BatchNorm + Scale into Scale layer
- Added support for TensorFlow framework (for DeePhi internal evaluation only)
- Other minor changes:
  - Support auto test for cityscapes segmentation dataset
  - Support for CV_16U image input

DNNC

- Added more operators:
  - Dilation convolution
  - Deconvolution
  - Flatten
  - Concat
  - Scale
  - Reorg
- Implement more optimizing compilation techniques:
  - Add more flexible node fusion strategies
  - Perform adaptive optimizations for concat and flatten operators
- Added support for TensorFlow framework (for Xilinx internal evaluation only)
- Support for DPU ABI v1.7

DExplorer

DExplorer is updated with the new option "-w" to display the DPU signature information, such as DPU architecture version, target version, working frequency, DPU core numbers, and so on.

Changes in API

Multiple IO is supported in this release. Therefore, all the API around input/output tensor such as `dpuGetInputXX/dpuGetOutputXX/dpuSetInputXX/dpuSetOutputXX` have been changed to preserve backward compatibility.

If multiple IO is not required in your application, no change is required.

Otherwise, pass an `idx` to specify a single tensor as the last parameter to the APIs, and refer to the `kernel_graph.gv` file generated by DNNC to get the value of `idx`.

New APIs: `dpuGetInputTensorCnt/ dpuGetOutputTensorCnt` are available to get the total number of input/output tensors for this node.

Example Changes

VGG-16 has been removed in this release due to its large size. New examples have been introduced, including:

- Multithreaded ResNet-50
- Multithreaded Inception-v1
- Semantic segmentation
- Pose estimation

Since v1.10

Toolchain Changes

DECENT

When performing neural network model compression, DECENT will incorporate the fix info into the compressed Caffe model file, instead of generating a separate file `fix_info.txt`. The DNNC compiler will deal with the fix information automatically.

The mean value was encapsulated into the prototxt file during model compression. The DNNC compiler handles this automatically.

Note: Use the DECENT and DNNC version included in the DNNDK release when compressing and compiling network models to avoid unexpected errors.

DNNC

Automatic CPU and DPU mapping.

- DNNC compiler optimization for multiplexed memory access.
- One-click compilation support for more neural networks including ResNet-50, VGG-16, Inception, SSD, YOLO, SqueezeNet, and DenseBox.
- Bug fixes, more hints, and error checking for handling user input.

DExplorer

The DExplorer utility has been added into this release, which can be used to show the DNNDK component version and DPU information.

DSight

The DSight utility is a performance profiling tool that has been added to this release.

Changes in API

New API

`dpuSetInputImage2()` – set input image to DPU Task, bypassing the requirement to specify the mean value parameter. Note the difference between this and the `dpuSetInputImage()` API.

Changed API

`dpuEnableTaskDebug()` has been renamed to `dpuEnableTaskDump()`.

Upgrading from Previous Versions

From v1.10 to v2.06

In this section, upgrading a deep learning application deployed with DNNDK v1.10 to DNNDK v2.06 using ResNet-50 is shown as an example.

1. Recompress the network model using DECENT from DNNDK v2.06 to generate new `deploy.prototxt` and `deploy.caffemodel` files.
2. Recompile the network model to generate ELF files for the new DPU kernel. Note that there is a change in the output file in v2.06. A new file named `kernel_graph.gv` will be generated by DNNC, which shows the structure of the network. It can be converted to JPEG format using a `dot` command (such as `dot -Tjpg -o xxx.jpg kernel_graph.gv`).

3. The `kernel_graph.gv` shows the input and output tensors at each node. Use `dpuGetOutputTensorCnt()` or `dpuGetInputTensorCnt()` to get the number of tensors at a node, and use an extra parameter `idx` to specify the tensor in subsequent processing functions. The `idx` is the index (beginning with 0, from left to right in `x.jpg`) of the tensor for a node. No changes are needed for the deployed code if the network has not changed.

From v1.07 to v1.10

In this section, upgrading a deep learning application deployed with DNNDK v1.07 to DNNDK v1.10 using ResNet-50 is shown as an example.

1. Recompress the network model using DECENT from DNNDK v1.10 to generate new `deploy.prototxt` and `deploy.caffemodel` files.
2. Recompile the network model to generate ELF files for the new DPU kernel. Note that the suffix "`_fixed`" has been removed in the generated files. For example, if the network is compiled with the option `–net_name=resnet50`, v1.07 would have generated `dpu_resnet50_fixed.elf` and `dpu_reset50_fc_fixed.elf`. In v1.10, the generated files will be named `dpu_resnet50_0.elf` and `dpu_resnet50_2.elf`. Make sure that the source code is modified to load the proper DPU kernel file when calling `dpuLoadKernel()`.
3. Modify `Makefile` to change the DPU kernel ELF file name from `dpu_resnet50_fixed.elf` and `dpu_resnet50_fc_fixed.elf` to `dpu_resnet50_0.elf" and "dpu_resnet50_2.elf`.
4. Consider using the new simplified API dpuSetInputImage2() instead of dpuSetInputImage().

Overview

Deep Neural Network Development Kit (DNNDK) is a full-stack deep learning SDK for the Deep-learning Processor Unit (DPU). It provides a unified solution for deep neural network inference applications by providing pruning, quantization, compilation, optimization, and run time support. Key highlights and features are listed below:

Innovative full-stack solution for deep learning inference application development

- A complete set of optimized tool chains, including compression, compilation and runtime.
- Lightweight C/C++ programming APIs.
- Easy-to-use with gradual learning curve.

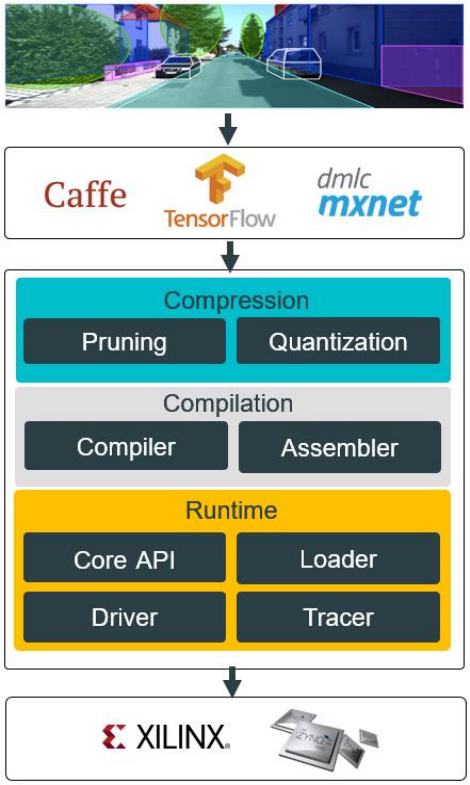

Figure 17: DNNDK Framework

DNNDK makes it simple for users without FPGA knowledge to develop deep learning inference applications by providing a set of lightweight C/C++ APIs while abstracting away the intricacies of the underlying FPGA device.

Deep-Learning Processor Unit

The DPU is designed to accelerate the computing workloads ofDee deep learning inference algorithms widely adopted in various computer vision applications, such as image/video classification, semantic segmentation, and object detection/tracking.

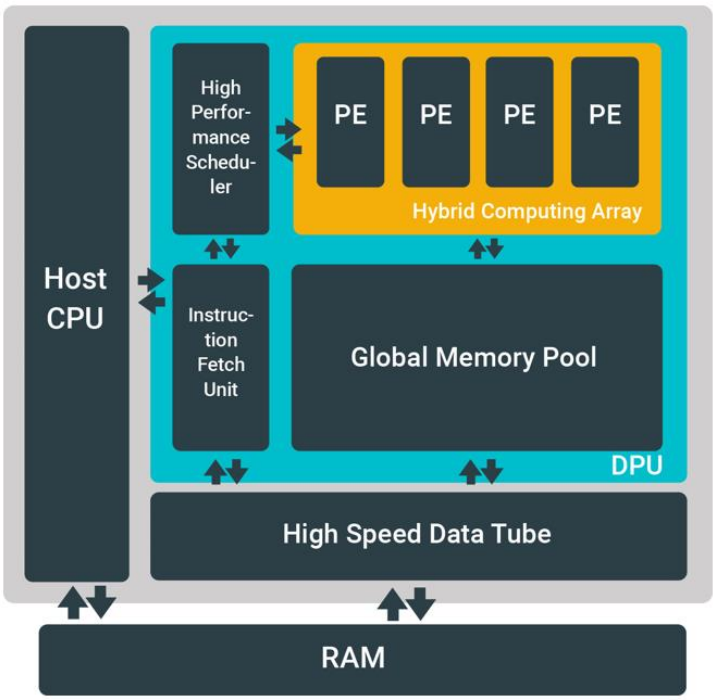

Figure 18: DPU Architecture

An efficient tensor-level instruction set is designed to support and accelerate various popular convolutional neural networks, such as VGG, ResNet, GoogLeNet, YOLO, SSD, and MobileNet, among others. The DPU is scalable to fit various Xilinx® Zynq®-7000 and Zynq UltraScale+™ MPSoC devices from edge to cloud to meet the requirements of many diverse applications.

DNNDK Framework

As shown in this figure, DNNDK is composed of Deep Compression Tool (DECENT), Deep Neural Network Compiler (DNNC), Deep Neural Network Assembler (DNNAS), Neural Network Runtime (N^2Cube), DPU Simulator, and Profiler.

Since v3.0, DNNDK supports both Caffe and TensorFlow with a unified framework. The toolchain flow for deploying Caffe and TensorFlow network models are almost the same, which delivers a familiar user experience.

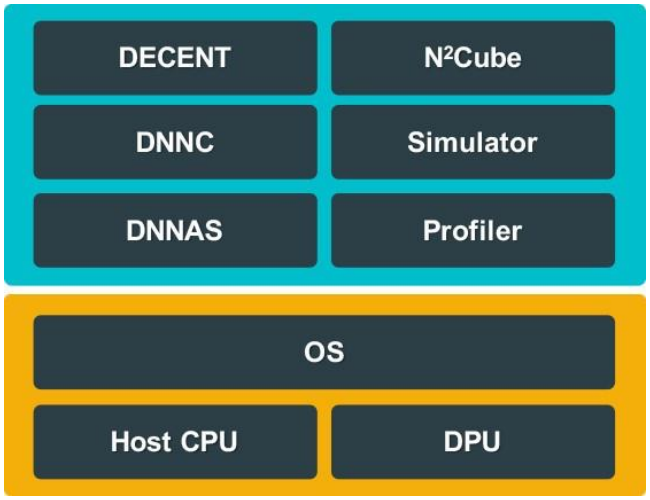

Figure 19: DNNDK Tools

NOTE: The DECENT tool can perform pruning and quantization; however, only quantization is included in this package.

DECENT

The process of inference is computation intensive and requires a high memory bandwidth to satisfy the low latency and high throughput requirement of edge applications.

The Deep Compression Tool, DECENT™, employs coarse-grained pruning, trained quantization and weight sharing to address these issues while achieving high performance and high energy efficiency with very small accuracy degradation.

DNNC

DNNC™ (Deep Neural Network Compiler) is the dedicated proprietary compiler designed for the DPU. It maps the neural network algorithm to the DPU instructions to achieve maxim utilization of DPU resources by balancing computing workload and memory access.

N²Cube

The Cube of Neutral Networks (N²Cube) is the DPU runtime engine. It acts as the loader for the DNNDK applications and handles resource allocation and DPU scheduling. Its core components include DPU driver, DPU loader, tracer, and programming APIs for application development.

N²Cube provides a lightweight set of programming interfaces through a library which abstracts away the details of the underlying hardware implementation.

The DPU driver runs in the kernel space of the Linux OS and includes DPU functions such as task scheduling, and efficient memory management to avoid memory copy overhead between the DPU and the CPU.

The DPU loader is responsible for dynamically loading DPU code and data into the DPU dedicated memory space and performs runtime address relocation.

The DPU performance profiler makes it possible for programmers to analyze the efficiency of DPU code and the utilization of resources layer by layer.

DNNAS

The Deep Neural Network Assembler (DNNAS) is responsible for assembling DPU instructions into ELF binary code. It is a part of the DNNC code generating back end, and cannot be invoked alone.

Profiler

The DPU profiler is composed of two components: DPU tracer and DSight. DPU tracer is implemented in the DNNDK runtime N²cube, and it is responsible for gathering the raw profiling data while running neural networks on DPU. With the provided raw profiling data, DSight can help to generate the visualized charts for performance analysis.

Overview

There are two stages for developing deep learning applications: training and inference. The training stage is used to design a neural network for a specific task (such as image classification) using a huge amount of training data. The inference stage involves the deployment of the previously designed neural network to handle new input data not seen during the training stage.

The DNNDK toolchain provides an innovative workflow to efficiently deploy deep learning inference applications on the DPU with five simple steps:

1. Compress the neural network model.
2. Compile the neural network model.
3. Program with DNNDK APIs.
4. Compile the hybrid DPU application.
5. Run the hybrid DPU executable.

In this chapter, ResNet-50 is used as an example to walk through each step. The Caffe/TensorFlow floating-point models for Resnet-50 and Inception-v1 can be found from DNNDK package as shown in the following table.

Table 2: Neural Network Models in the DNNDK package

Model	Directory
ResNet-50	$dnndk_pkg/host_x86/models/caffe/resnet50 $dnndk_pkg/host_x86/models/tensorflow/resnet50
Inception-v1	$dnndk_pkg/host_x86/models/caffe/inception_v1 $dnndk_pkg/host_x86/models/tensorflow/inception_v1

Note: This chapter only covers model quantization with the Deep Compression Tool (DECENT) because the pruning tool is not included in this release.

Network Compression (Caffe Version)

DECENT takes a floating-point network model, pre-trained weights and a calibration dataset in Caffe format as inputs to generate a lightweight quantized model with INT8 weights.

Table 3: DECENT Input Files

No.	Name	Description
1	float.prototxt	Floating-point model for ResNet-50
2	float.caffemodel	Pre-trained weights file for ResNet-50
3	calibration dataset	A subset of the training set containing 100 to 1000 images

A script file named `decent.sh` is located in `$dnndk_pkg/host_x86/models/resnet50`, shown in the following figure. This invokes the DECENT tool to perform quantization with the appropriate parameters.

```
#working directory
work_dir=$(pwd)
#path of float model
model_dir=${work_dir}
#output directory
output_dir=${work_dir}/decent_output

[ -d "$output_dir" ] || mkdir "$output_dir"

decent     quantize                                  \
           -model ${model_dir}/float.prototxt        \
           -weights ${model_dir}/float.caffemodel    \
           -output_dir ${output_dir}                 \
           -method 1
```

Figure 20: Sample DECENT Quantization Script

Note: Before launching quantization for ResNet-50, you should first prepare the calibration data set used by DECENT. You can download 100 to 1000 images of ImageNet data set from http://academictorrents.com/collection/imagenet-2012 or http://www.image-net.org/download.php and then change the settings for `source` and `root_folder` of `image_data_param` in ResNet-50 prototxt accordingly.

The script might take several minutes to an hour to finish. Once quantization is done, the files `deploy.prototxt` and `deploy.caffemodel` are generated in the `decent_output` directory, which could be fed to DNNC compiler for following compilation process.

Table 4: DECENT Output Files

N0.	Name	Description
1	deploy.prototxt	quantized network description file
2	deploy.caffemodel	quantized Caffe model parameter file (non-standard Caffe format)

Network Compression (TensorFlow Version)

DECENT takes a floating-point network model, pre-trained weights, and a calibration dataset in Caffe format as inputs to generate a lightweight quantized model with INT8 weights.

Table 5: Input files for DECENT_Q

No.	Name	Description
1	frozen_resnet_v1_50.pb	Floating-point frozen graph for ResNet-50
2	calibration dataset	A subset of the ImageNet Dataset (without labels) containing 100 images
3	input_fn	A python function to read images in calibration dataset and do preprocess

A script file named `resnet_v1_50_input_fn.py` can be found in `$dnndk_pkg/host_x86/models/resnet50`, shown in the following figure.

```
calib_image_dir = "../../calibration_data/images/"
calib_image_list = "../../calibratoin_data/calib.txt"
calib_batch_size = 1
def calib_input(iter):
  images = []
  line = open(calib_image_list).readlines()
  for index in range(0, batch_size):
    curline = line[iter * batch_size + index]
    [image_name, label_id] = curline.split(' ')
    image = cv2.imread(calib_image_dir + image_name)
    image = central_crop(image, 224, 224)
    image = mean_image_subtraction(image, MEANS)
    images.append(image)
  return {"input": images}
```

Figure 21: Sample Customized input_fn Script

A script file named `decent_q.sh` can be found in `$dnndk_pkg/host_x86/models/resnet50`, shown in the following figure. This invokes the DECENT_Q tool to perform quantization with the appropriate parameters.

```
decent_q quantize \
  --input_frozen_graph frozen_resnet_v1_50.pb \
  --input_nodes input \
  --input_shapes ?,224,224,3 \
  --output_nodes resnet_v1_50/predictions/Reshape_1 \
  --input_fn resnet_v1_50_input_fn.calib_input \
  --method 1 \
  --gpu 0 \
  --calib_iter 100 \
```

Figure 22: Screenshot of Sample DECENT_Q Quantization Script

The script might take several minutes to finish. When quantization is done, the following two files are generated under the `quantize_results` directory. Then you can use `deploy_model.pb` to compile the model using DNNC.

Table 6: DECENT_Q Output Files

No.	Name	Description
1	deploy_model.pb	Quantized model for DNNC (extended TensorFlow format)
2	quantize_eval_model.pb	Quantized model for evaluation

Network Compilation

DNNC can support both Caffe and TensorFlow model compilation in one binary tool. DNNDK provides different dnnc.sh script files for compiling Caffe ResNet-50 and TensorFlow ResNet-50 models. Running the script file invokes the DNNC tool to perform model compilation with the appropriate options.

Compiling Caffe ResNet-50

The script file for compiling Caffe ResNet-50, `dnnc.sh`, can be found in `$dnndk_pkg/host_x86/models/caffe/resnet50`, as shown in the following figure. For Caffe model compilation, you can use the DNNC default parser; there is no need to specify parser type using the `--parser` option.

```
#!/usr/bin/env bash

net="resnet50"
model_dir="decent_output"
output_dir="dnnc_output"

# Set DPU Arch
DPU_ARCH="4096FA"
# Set CPU Arch
CPU_ARCH="arm64"
# Set DNNC mode
DNNC_MODE="debug"

if [ ! -d "$model_dir" ]; then
    echo "Cannot found directory of $model_dir"
    exit 1
fi

[ -d "$output_dir" ] || mkdir "$output_dir"

echo "Compiling Network: ${net}"
dnnc    --prototxt=${model_dir}/deploy.prototxt      \
        --caffemodel=${model_dir}/deploy.caffemodel \
        --output_dir=${output_dir}                   \
        --net_name=${net}                            \
        --dpu=${DPU_ARCH}                            \
        --mode=${DNNC_MODE}                          \
        --cpu_arch=${CPU_ARCH}
```

Figure 23: DNNC Compilation Script for Caffe ResNet-50

The following figure shows DNNC output information when compilation is successful.

```
Compiling Network: resnet50
[DNNC][Warning] layer [prob] is not supported in DPU, deploy it in CPU instead.

DNNC Kernel Information

1. Overview
kernel numbers  : 2
kernel topology : resnet50_kernel_graph.jpg

2. Kernel Description in Detail
kernel id       : 0
kernel name     : resnet50_0
type            : DPUKernel
nodes           : NA
input node(s)   : conv1(0)
output node(s)  : fc1000(0)

kernel id       : 1
kernel name     : resnet50_1
type            : CPUKernel
nodes           : NA
input node(s)   : prob
output node(s)  : prob
```

Figure 24: DNNC Compilation Log for Caffe ResNet-50

Compiling TensorFlow ResNet-50

The script file for compiling TensorFlow ResNet-50 model, `dnnc.sh`, can be found in `$dnndk_pkg/host_x86/models/tensorflow/resnet50`, as shown in the following figure. For TensorFlow model compilation, you must specify the parser type using `tensorflow` through the `--parser` option; otherwise DNNC will display an error.

```
#!/usr/bin/env bash

net="resnet50"

# Work space directory
work_dir=$(pwd)
# Path of caffe quantization model
model_dir=${work_dir}/quantize_results
# Output directory
output_dir="dnnc_output"
tf_model=${model_dir}/deploy_model.pb

# Set DPU Arch
DPU_ARCH="4096FA"
# Set CPU Arch
CPU_ARCH="arm64"
# Set DNNC mode
DNNC_MODE="debug"

if [ ! -d "$model_dir" ]; then
    echo "Can not found directory of $model_dir"
    exit 1
fi

[ -d "$output_dir" ] || mkdir "$output_dir"

echo "Compiling Network: ${net}"
dnnc    --parser=tensorflow                     \
        --frozen_pb=${tf_model}                 \
        --output_dir=${output_dir}              \
        --dpu=${DPU_ARCH}                       \
        --mode=${DNNC_MODE}                     \
        --cpu_arch=${CPU_ARCH}                  \
        --net_name=${net}
```

Figure 25: DNNC Compilation Script for TensorFlow ResNet-50

The following figure shows DNNC output information when compilation is successful.

```
Compiling Network: resnet50
[DNNC][Warning] layer [resnet_v1_50_SpatialSqueeze] is not supported in DPU, deploy it in CPU instead.
[DNNC][Warning] layer [resnet_v1_50_predictions_Softmax] is not supported in DPU, deploy it in CPU instead.

DNNC Kernel Information

1. Overview
kernel numbers  : 2
kernel topology : resnet50_kernel_graph.jpg

2. Kernel Description in Detail
kernel id       : 0
kernel name     : resnet50_0
type            : DPUKernel
nodes           : NA
input node(s)   : resnet_v1_50_conv1_Conv2D(0)
output node(s)  : resnet_v1_50_logits_Conv2D(0)

kernel id       : 1
kernel name     : resnet50_1
type            : CPUKernel
nodes           : NA
input node(s)   : resnet_v1_50_SpatialSqueeze
output node(s)  : resnet_v1_50_predictions_Softmax
```

Figure 26: DNNC Compilation Log for TensorFlow ResNet-50

Output Kernels

For both Caffe and TensorFlow, DNNC compiles the ResNet-50 model into one DPU kernel, which is an ELF format file containing DPU instructions and parameters for ResNet-50. This also shows the information about layers unsupported by the DPU, as shown in Figure 24, on page 40, and Figure 29, on page 41 . The ResNet-50 network model is compiled and transformed into two different kernels:

- Kernel 0： resnet50_0 （run on DPU）
- Kernel 1： resnet50_1 （deploy on the CPU）

The kernels `resnet50_0` runs on the DPU. DNNC generates an ELF object file for this kernel in the `output_dir` directory, with the name `dpu_resnet50_0.elf`.

The other kernel, `resnet50_1`, is for “Softmax” operations, which are not supported by DPU and must be deployed and run on the CPU.

Programming with DNNDK

To develop deep learning applications on the DPU, three types of work must be done:

- Use DNNDK APIs to manage DPU kernels.
  - DPU kernel creation and destruction.
  - DPU task creation.
  - Managing input and output tensors.
- Implement kernels not supported by the DPU on the CPU.
- Add pre-processing and post-processing routines to read in data or calculate results.

The sample code for managing the DPU kernels and tasks are programmed in the `main()` function.

```
int main(void) {
    /* DPU Kernels/Tasks for running ResNet-50 */
    DPUKernel* kernelConv;
    DPUTask* taskConv;
    /* Attach to DPU driver and prepare for running */
    dpuOpen();
    /* Create DPU Kernels for CONV Nodes in ResNet-50 */
    kernelConv = dpuLoadKernel(KERNEL_CONV);
    /* Create DPU Tasks for CONV Nodes in ResNet-50 */
    taskConv = dpuCreateTask(kernelConv, 0);

    /* Run CONV Kernel for ResNet-50 */
runResnet50(taskConv);
    /* Destroy DPU Tasks & release resources */
    dpuDestroyTask(taskConv);
    /* Destroy DPU Kernel & release resources */
    dpuDestroyKernel(kernelConv);
    /* Detach DPU driver & release resources */
    dpuClose();
    return 0;
```

The main() operations include:

- Call dpuOpen() to open the DPU device.
- Call dpuLoadKernel() to load the DPU kernel reset50_0.
- Call dpuCreateTask() to create task for each DPU kernel.
- Call dpuDestroyKernel() and dpuDestroyTask() to destroy DPU kernel and task.
- Call dpuClose() to close the DPU device.

The main image classification work is done in the function runResnet50(), which performs the following operations:

- Fetches an image using the OpenCV function imread() and set it as the input to the DPU kernel resnet50_0 by calling the dpuSetInputImage2() API.
- Calls dpuRunTask() to run the taskConv convolution operation in the ResNet-50 network model.
- Does softmax on the CPU using the output of the fully connected operation as input.
- Outputs the top-5 classification category and the corresponding probability.

```
Mat image = imread(baseImagePath + imageName);
dpuSetInputImage2(taskConv, CONV_INPUT_NODE, image);

dpuRunTask(taskConv);

/* Get FC result and convert from INT8 to FP32 format */
dpuGetOutputTensorInHWCFP32(taskConv, FC_OUTPUT_NODE,
                                FCResult, channel);
CPUCalcSoftmax(FCResult, channel, softmax);
TopK(softmax, channel, 5, kinds);
```

Compiling the Hybrid Executable

To generate the hybrid executable, change to the $dnndk_pkg/samples/resnet50 directory, and run make. This compiles the application source code to CPU binary code, and then links it against the DPU kernels dpu_resnet50_0.elf and dpu_resnet50_2.elf.

Running the Application

In the $dnndk_pkg/samples/resnet50 directory, execute ./resnet50 to run this application, and see its output.

DECENT Overview

The Deep Compression Tool (DECENT) includes two capabilities: Coarse-Grained Pruning and trained quantization. These reduce the number of required operations and quantize the weights. The entire working flow of DECENT is shown in the following figure. In this release, only the quantization tool is included. Contact the Xilinx support team if pruning tool is necessary for your project evaluation.

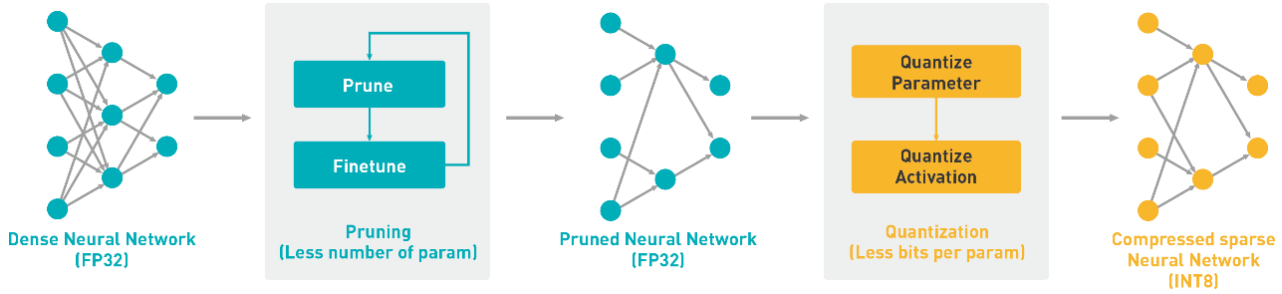

Figure 27: DECENT Pruning and Quantization Flow

Generally, 32-bit floating-point weights and activation values are used when training neural networks. By converting the 32-bit floating-point weights and activations to 8-bit integer (INT8), the DECENT quantize tool can reduce the computing complexity without losing prediction accuracy. The fixed-point network model requires less memory bandwidth, thus providing faster speed and higher power efficiency than the floating-point model. This tool supports common layers in neural networks, such as convolution, pooling, fully-connected, and batchnorm among others.

There is no need to retrain the network after the quantization process, instead only a small set of images is needed to analyze the distribution of activation values for calibration. The quantization time ranges from several minutes to an hour, depending on GPU availability, neural network complexity, and the size of the calibration image set.

DECENT_Q supports Caffe and TensorFlow.

- *DECENT (Caffe Version) is based on Caffe 1.0.*
- *DECENT_Q (TensorFlow Version) is based on TensorFlow 1.9.*

Note: The output file of DECENT is an extended Caffe/TensorFlow model, which can only be used as input to the DNNC compiler.

DECENT Working Flow

The overall workflow of model quantization are as follows：

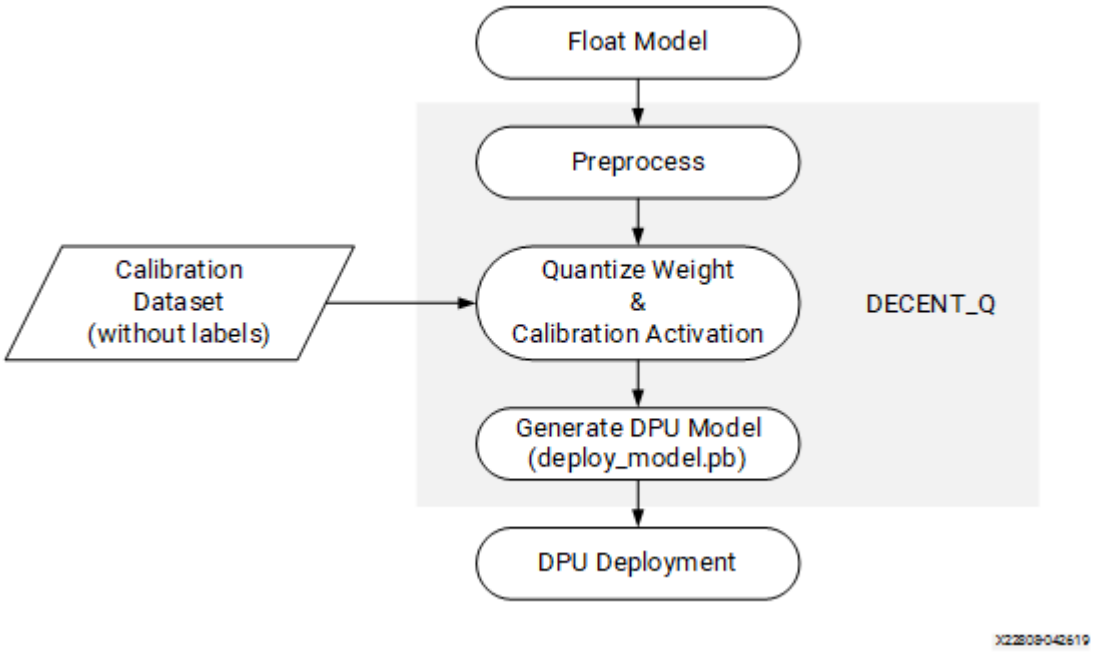

Figure 28: DECENT Workflow

As shown above, the DECENT takes a float model as input (prototxt & caffemodel for Caffe version, frozen GraphDef file for TensorFlow version), does some preprocessing (folds batchnorms and removes useless nodes), and then quantizes the weights/biases and activations to the given bit width.

To improve the precision of the quantized graph, DECENT needs to run some iterations of inference of the model to calibrate the activation; therefore, a calibration dataset input is need. Because there is no need for back propagation, no image labels are needed.

After calibration, the quantized model is transformed to DPU deployable model named `deploy.prototxt/deploy.caffemodel` by Caffe version DECENT or `deploy_model.pb` for TensorFlow version DECENT, which follows the data format of DPU. Then it can be compiled by DNNC compiler and deployed to DPU. Note that the quantized model cannot be taken in by standard vision Caffe or TensorFlow framework.

DECENT (Caffe Version) Usage

The options supported by DECENT are shown below.

Table 7: DECENT Options List

Name	Type	Optional	Default	description
model	String	Required	-	Floating-point prototxt file (e.g. "float.prototxt").
weights	String	Required	-	The pre-trained floating-point weights (e.g." float.caffemodel").
weights_bit	Int32	Optional	8	Bit width for quantized weight and bias.
data_bit	Int32	Optional	8	Bit width for quantized activation.
method	Int32	Optional	0	Quantization methods, including 0 for non-overflow and 1 for min-diffs. Method 0 demands shorter execution time compared with method 1. For non-overflow method, make sure no values are saturated during quantization. The results can be easily affected by outliers. For min-diffs method, it allows saturation for quantization to get lower quantization difference, higher endurance to outliers. It usually ends with narrower ranges than non-overflow method.
calib_iter	Int32	Optional	100	Max iterations for calibration.
auto_test	Bool	Optional	FALSE	Run test after calibration, test dataset required.
test_iter	Int32	Optional	50	Max iterations for testing.
output_dir	String	Optional	fix_results	Output directory for the fixed-point results.
gpu	String	Optional	0	GPU device id for calibration and test
ignore_layers	String	Optional	none	List of layers to ignore during quantization.
ignore_layers_file	String	Optional	none	Protobuf file which defines the layers to ignore during quantization, starting with 'ignore_layers:'

DECENT (Caffe Version) Working Flow

Prepare the Neural Network Model

Before running DECENT, prepare the Caffe model in floating-point format and calibration data set, including:

- Caffe floating-point network model prototxt file.
- Pre-trained Caffe floating-point network model caffemodel file.
- Calibration data set. The calibration set is usually a subset of the training set or actual application images (at least 100 images). Make sure to set the source and root_folder in image_data_param to the actual calibration image list and image folder path, as shown in the following figure.

```
# ResNet-50
name: "ResNet-50"
layer {
  name: "data"
  type: "ImageData"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: false
    mean_value: 104
    mean_value: 107
    mean_value: 123
  }
  image_data_param {
    source: "./data/imagenet_256/calibration.txt"
    root_folder: "./data/imagenet_256/calibration_images/"
    batch_size: 10
    shuffle: false
    new_height: 224
    new_width: 224
  }
}
```

Figure 29: Sample Caffe Layer for Quantization

Note: Only the 3-mean-value format is supported by DECENT. Convert to the 3-mean-value format as required.

Run DECENT

Run the following command line to generate a fixed-point model:

```
$decent quantize -model float.prototxt -weights float.caffemodel [options]
```

In the above command line, `[options]` stands for optional parameters. The three commonly used options are shown below:

- **weights_bit**: Bit width for quantized weight and bias (default is 8).
- **data_bit**: Bit width for quantized activation (default is 8).
- **method**: Quantization method.
  "0" indicates non-overflow method, and "1" indicates min-diffs mothed. The default method is 0. Non-overflow method makes sure no values are saturated during quantization. The results can be easily affected by outliers. Min-diffs method allows saturation for quantization to get lower quantization difference, higher endurance to outliers. It usually ends with narrower ranges than non-overflow method.

Output

After successful execution of the above command, two files are generated (under the default directory `./quantize_results/`), which can be used as input files to DNNC:

- fixed-point model network (`deploy.prototxt`)
- fixed-point weights (`deploy.caffemodel`)

DECENT (TensorFlow Version) Usage

The options supported by DECENT_Q are shown in tables 8 and 9.

Table 8: DECENT Required Options List

Option Name	Type	Description
--input_frozen_graph	String	TensorFlow frozen `GraphDef` file of the floating-point model.
--input_nodes	String	The name list of input nodes, comma separated.
--output_nodes	String	The name list of output nodes, comma separated.
--input_shapes	String	The shape list of input_nodes. Must be a 4-dimension shape for each node, comma separated, such as 1,224,224,3; support unknown size for batchsize, such as ?,224,224,3. In case of multiple input_node options, assign shape list of each node, separated by':'. For example, ?,224,224,3:?,300,300,1.

--input_fn	String	The function that provides input data for the input_nodes option, used with calibration dataset. The function format is `module_name.input_fn_name`, such as `my_input_fn.input_fn`. The `input_fn` command should take an `int` object as input, which indicates the calibration step number, and should return a `dict'(input_node_name, numpy.Array)'` object for each call, which will be fed into the input nodes of the model. The shape of `numpy.Array` should be consistent with `input_shapes`. Meanwhile, two preset input functions are provided: `default`: a simple image load function to load raw image files and do preprocessings. Mean subtraction, central crop, resize and normalization are supported. Should be used with the `[DefaultInputFnConfig]` command, described below. `random`: a function to produce random numbers for all inputs.

Table 9: DNNC Optional Option List

Name	Type	Description
Quantize Configuration		
--weight_bit	Int32	Bit width for quantized weight and bias. Default: 8
--activation_bit	Int32	Bit width for quantized activation. Default: 8
--method	Int32	the method for quantization 0: non-overflow method. Makes sure no values are saturated during quantization; might get worse results in case of outliers. 1: min-diffs method. Allows saturation for quantization to get lower quantization difference; higher endurance to outliers. Usually ends with narrower ranges than non-overflow method. Choices: [0, 1] Default: 1
--calib_iter	Int32	The iterations of calibration, total number of images for calibration = `calib_iter * batch_size`. Default: `100`
--ignore_nodes	String	The name list of nodes to be ignored during quantization. Ignored nodes will be left unquantized during quantization.
--skip_check	Int32	If set to 1, the check for float model will be skipped. This can be useful when only part of the input model is quantized. Default: 0

--output_dir	String	The directory to save the quantization results. Default: `./quantize_results`
Default Input Function Configurations		
--batch_size	Int32	[DefaultInputFnConfig] The batch_size of calibration, total number of images for calibration = `calib_iter * batch_size`. Default: 1
--image_dir	String	[DefaultInputFnConfig] The directory holding the images
--image_list	String	[DefaultInputFnConfig] The text file of list of image filenames.
--size_type	Int32	[DefaultInputFnConfig] The type of image resize, options are: 0: central crop to input_shapes 1: resize to input_shapes Choices: [0, 1] Default: 0
--means	String	[DefaultInputFnConfig] The means list of images per channel, comma separated. Images will be subtracted by means for each channel.
--scales	String	[DefaultInputFnConfig] The scales of images per channel, comma separated. Images will be multiplied by scale for each channel.
--normalize	Int32	[DefaultInputFnConfig] if set to 1, images will be converted from [0, 255] to [-1,1]. Choices: [0, 1] Default: 0
Session Configurations		
--gpu	String	The gpu device's id used for quantization, comma separated.
--gpu_memory_fraction	float	The gpu memory fraction used for quantization, between 0-1. Default: 0.5
Others		
--help		Show all available options of DECENT_Q.
--version		Show DECENT_Q version information.

Preparing the Neural Network Model

Before running DECENT_Q, prepare the frozen TensorFlow model in floating-point format and calibration set, including:

Table 10: Input Files for DECENT_Q

No.	Name	Description
1	`frozen_graph.pb`	Floating-point frozen graph for ResNet-50.
2	`calibration dataset`	A subset of the training set containing 100 to 1000 images.
3	`Input_fn`	An input function to convert the calibration dataset to the frozen_graph 's input data during quantize calibration. Usually will do some data preprocessing and augmentation.

How to Get the Frozen Graph

In most situations, training a model with TensorFlow gives you a folder containing a GraphDef file (usually ending with the `.pb` or `.pbtxt` extension) and a set of checkpoint files. What you need for mobile or embedded deployment is a single GraphDef file that has been 'frozen', or had its variables converted into inline constants so everything is in one file. To handle the conversion, TensorFlow provided `freeze_graph.py`, which is automatically installed with DECENT_Q.

An example of command-line usage is:

```
$ freeze_graph \
    --input_graph=/tmp/inception_v1_inf_graph.pb \
    --input_checkpoint=/tmp/checkpoints/model.ckpt-1000 \
    --input_binary=true \
    --output_graph=/tmp/frozen_graph.pb \
    --output_node_names=InceptionV1/Predictions/Reshape_1
```

Note: Because the operations of data preprocessing and loss functions are not needed for inference and deployment, the `frozen_graph.pb` should only include the main part of the model. In particular, the data preprocessing operations should be taken in the `Input_fn` to generate correct input data for quantize calibration.

Note: Type `freeze_graph --help` for more options.

Note: The input and output node names will vary depending on the model, but you can inspect and estimate them with `decent_q`. An example of command-line usage is:

```
$ decent_q inspect --input_frozen_graph=/tmp/inception_v1_inf_graph.pb
```

Another way to get the graph's input and output name is by visualizing the graph. Both **tensorboard** and **netron** can do this. An example of using netron is:

```
$ pip install netron
$ netron /tmp/inception_v3_inf_graph.pb
```

How to Get the Calibration Dataset and Input Function

The calibration set is usually a subset of the training/validation dataset or actual application images (at least 100 images for performance). The input function is a python importable function to load calibration dataset and perform data preprocessing. DECENT_Q can accept both pre-defined input functions or custom one.

Pre-Defined Input Function

In convenience, DECENT_Q provides two pre-defined input functions for some common cases, listed in the following table

Table 11: Pre-Defined Input Functions for DECENT_Q

No.	Name	Description
1	default	A simple input function to load raw image files (opencv readable files) and perform preprocessing. Should be used with the `DefaultInputFnConfig`. Supported preprocessing methods are: • Mean subtraction (Channel-wise) • Scale multiplication (Channel-wise) • Central crop • Resize • Normalize [0, 255] to [-1, 1]
2	random	An input function to produce random numbers for all inputs.

Custom input function: If the pre-defined input functions cannot satisfy the preprocessing requirements for the model or the dataset are not raw image files, a custom input function should be provided.

The function input is format is `module_name.input_fn_name`, such as `my_input_fn.calib_input`. The `input_fn` should take an `int` object as input indicating the calibration step number, and should return a `dict'(input_node_name, numpy.Array)'` object for each call, which will be fed into the input nodes of the model. The shape of `numpy.array` should be consistent with `input_shapes`.

The pseudo code example looks like the following sample:

```
$ "my_input_fn.py"
def calib_input(iter):
"""A function that provides input data for the calibratoin
Args:
iter: A `int` object, indicating the calibration step number
Returns:
    dict(input_node_name, numpy.array): a `dict` object, which will be fed into
the model
"""
  image = load_image(iter)
  preprocessed_image = do_preprocess(image)
  return {"input": preprocessed_images}
```

Run DECENT_Q

Run the following command line to quantize the model:

```
$decent_q quantize \
   --input_graph_def  frozen_graph.pb \
   --input_nodes  ${input_nodes} \
   --input_shapes  ${input_shapes} \
   --output_nodes   ${output_nodes} \
   --input_fn  Pre-defined|Custom input_fn \
   [options]
```

In the above command line, the [options] are placeholders for optional parameters. See Table 5 for more information. The three commonly used options are shown below:

- **weights_bit**: Bit width for quantized weight and bias (default is 8).
- **activation_bit**: Bit width for quantized activation (default is 8).
- **method**: Quantization method. "0" indicates non-overflow method, and "1" indicates min-diffs mothed. The default method is 0.
  Non-overflow method makes sure no values are saturated during quantization. The results can be easily affected by outliers. Min-diffs method allows saturation for quantization to get lower quantization difference, higher endurance to outliers. It usually ends with narrower ranges than non-overflow method.

Output

After successful execution of the above command, two files are generated (under the default directory `./quantize_results/`), which can be used as the input files to DNNC, as shown in the following table

Table 12: DECENT_Q Output Files

N0.	Name	Description
1	deploy_model.pb	Quantized model for DNNC (extended TensorFlow format)
2	quantize_eval_model.pb	Quantized model for evaluation

DNNC Overview

The architecture of the Deep Neural Network Compiler (DNNC) compiler is shown in the following figure. The front-end parser is responsible for parsing the Caffe/TensorFlow model and generates an intermediate representation (IR) of the input model. The optimizer handles optimizations based on the IR, and the code generator maps the optimized IR to DPU instructions.

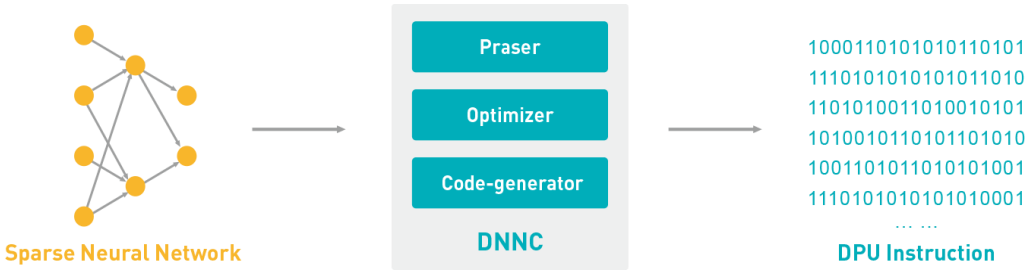

Figure 30: DNNC Components

Using DNNC

DNNC Options

The Deep Neural Network Compiler (DNNC) requires parameters to control the compilation for neural network models. These parameters are divided into two categories. The first group (shown in Table 13), shows the required parameters, and the next group (shown in Table 14), shows optional parameters.

DNNC Compilation Mode

DNNC supports two different compilation modes for different purposes:

- **Deployment mode:** This mode is primarily used for deployment purpose. DNNC must be provided with both `deploy.prototxt` and `deploy.caffemodel` files using the `--prototxt` and `--caffemodel` options before compiling Caffe models. In this mode, ELF files are generated by DNNC for model deployment.
- **Dummy mode:** This mode is enabled by providing DNNC with only the `deploy.prototxt` file before compiling Caffe models, and should be used for compilation test purposes only. In this mode, DNNC displays some warnings and no ELF files are generated.

Table 13: DNNC Required Option List

Parameters	Description
`--parser`	Select different parser type for DNNC. Currently DNNC supports two different parsers: • **caffe**: Caffe parser that can parse Caffe *. prototxt and *. caffemodel model files. When this parser type is selected, '`--prototxt`' and '`--caffemodel`' options must be provided with parameters; • **tensorflow**: TensorFlow parser that can parse TensorFlow frozen pb model files. When this parser type is selected, '`--frozen_pb`' must be provided with parameters. **Note:** For compiling Caffe models, you can use DNNC's default Caffe parser. There is no need to specify parser type through this option, but when compiling TensorFlow models, you must select TensorFlow parser type using this option, or DNNC will display an error.
`--prototxt`	Path of Caffe prototxt file. **Notes:** • This option is only required for Caffe model type parser. See `--parser` option for details. • When the `--caffemodel` option is not provided with a parameter, DNNC compiles the model with dummy mode. See DNNC Compilation Mode for details.
`--caffemodel`	Path of caffemodel file. **Notes:** • This option is only required for Caffe model type parser. See `--parser` option for details. • When this option is not provided with a parameter, DNNC compiles the model with dummy mode. See DNNC Compilation Mode for details.
`--frozen_pb`	Path of TensorFlow frozen pb file. **Notes:** • This option is only required for TensorFlow model type parser. See `--parser` option for details. The `frozen_pb` file provided must come from DECENT's output, or some unexpected errors will occur.
`--output_dir`	Path of output directory
`--net_name`	Name of neural network
`--dpu`	DPU arch type (supported list: 512FA, 800FA, 1024FA, 1152FA, 1600FA, 2304FA, 3136FA, 4096FA)

`--cpu_arch`	CPU target (supported list: arm32, arm64)

Table 14: DNNC Optional Option List

Parameters	Description
--help	Show all available options of DNNC.
--version	Show DNNC version information. The DPU target version supported by DNNC tool is also displayed. **Note**: For each target version DPU, the suited version DNNC should be used. Check if DPU and DNNC are matchable with the help of running command "dexplorer -w" on evaluation board and "dnnc --version" on host machine separately.
--save_kernel	Whether save kernel description in file or not
--abi	Indicate the ABI version for DPU ELF generated by DNNC. 0 for DNNC to produce legacy ABI version DPU ELF. For prior version N2Cube, it only supports the legacy version DPU ELF. With option "--abi=0", newer version DNNC can generate legacy DPU ELF for forward compatibility. 1 for DNNC to produce the latest ABI version DPU ELF. **Note**: this option is available since DNNC v2.03 and is deprecated in v3.0. If ABI version is specified as 0 in this option in DNNDK v3.0, DNNC will discard it, output a warning message, and use ABI version 1 instead.
--mode	Compilation mode of DPU kernel - debug or normal. debug: the layers of the network model run one by one under the scheduling of N2Cube. With the help of DExplorer, the users can perform debugging or performance profiling for each layer of DPU kernel compiled in debug mode. normal: all layers of network model are packaged into one single DPU execution unit, and there isn't any interrupt involved during each execution. Compared with debug mode, Normal mode DPU kernel delivers better performance and should be used during production release phase.
--dump	Dump different type information, use commas as delimiter when multiple types are given: graph: original graph and transformed graph in DOT format. weights: weights and bias data for different layers. ir: immediate representation for different layer in DNNC. quant_info: quaternization information for different layers. log: other compilation log generated by DNNC. all: dump all listed above. **Note**: all the dumped files except for graph type is decrypted by DNNC. In case of network compilation errors, these dump files can be delivered to DNNDK support team for further analysis.

Compiling ResNet50

To illustrate some basic concepts in DNNC, we will use ResNet-50 as compilation example and suppose that both Average-Pooling and SoftMax operator are not supported in the target DPU. When compiling a neural network, the required options should be specified to DNNC compiler. Refer to the script files provided in the DNNDK release package to become familiar with the use of various DNNC options.

Once the compilation is successful, DNNC will generate ELF objects and kernel information for deployment. These files are located under the folder specified by the parameter `output_dir`. The following figure shows a screenshot of the DNNC output when compiling the ResNet-50 network.

```
[DNNC][Warning] Only max pooling is supported, but [pool5] layer has average pooling type.
[DNNC][Warning] layer [pool5] is not supported in DPU, deploy it in CPU instead.
[DNNC][Warning] layer [prob] is not supported in DPU, deploy it in CPU instead.

DNNC Kernel Information

1. Overview
kernel numbers  : 4
kernel topology : resnet50_kernel_graph.jpg

2. Kernel Description in Detail
kernel id       : 0
kernel name     : resnet50_0
type            : DPUKernel
nodes           : NA
input node(s)   : conv1(0)
output node(s)  : res5c_branch2c(0)

kernel id       : 1
kernel name     : resnet50_1
type            : CPUKernel
nodes           : NA
input node(s)   : pool5
output node(s)  : pool5

kernel id       : 2
kernel name     : resnet50_2
type            : DPUKernel
nodes           : NA
input node(s)   : fc1000(0)
output node(s)  : fc1000(0)

kernel id       : 3
kernel name     : resnet50_3
type            : CPUKernel
nodes           : NA
input node(s)   : prob
output node(s)  : prob
```

Figure 31: DNNC Output for ResNet-50

Due to the limited number of operations supported by the DPU (see Table 15: Operations Supported by the DPU), DNNC automatically partitions the target neural network into different kernels when operations exist that are not supported by DPU. You are responsible for the data transfer and communication between different kernels, using APIs provided by N^2Cube that can be used for retrieving input and output address based on the input and output nodes of the kernel.

The kernel description information generated by DNNC includes two parts. The first part describes the number of kernels and the topology:

- **Kernel number**: The number of kernels generated by DNNC after compilation. Different neural networks will be compiled to different number of kernels depending on which operators can be supported in the DPU, and each kernel will be described in detail in the second part.
- **Kernel topology**: The kernel topology description file describes the kernels in the kernel graph view when compilation is finished. The `kernel_graph` file is saved in standard JPEG format with file extension `.jpg` in the output directory specified by the `–output_dir` option. If graphviz is not installed on the host system, DNNC outputs a `DOT` (graph description language) format file with extension `.gv` instead.
  You can convert the `.gv` format file to a JPEG file using the following command:

```
dot -Tjpg -o kernel_graph.jpg kernel_graph.gv
```

For example, the kernel graph in JPEG format for ResNet-50 is shown in the following figure. The kernel graph node describes the kernel id and its type, while the edge shows the relationship between different kernels in two tuples. The first item represents the output tensor from the source kernel, while the second item shows the input tensor to the destination kernel. The tuple contains two parts, the name of input/output node binding to the tensor, and the tensor index of the input/output node. Using the node name and index provided in the tuple, users can use the APIs provided by N^{2}Cube to get the input or output tensor address.

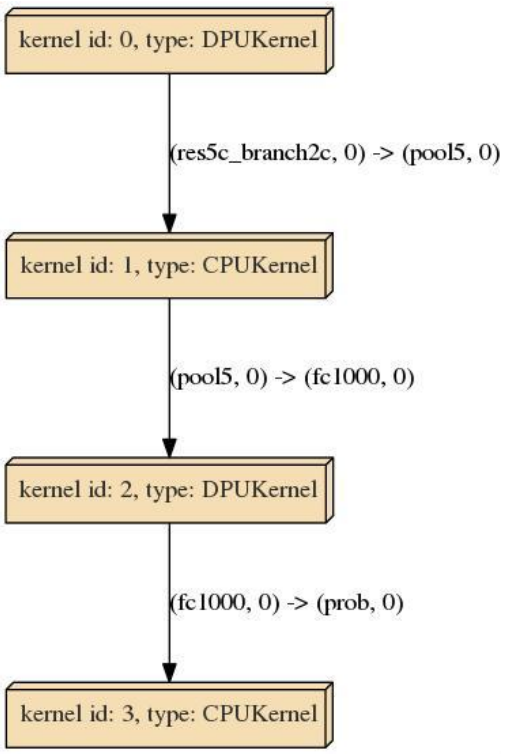

Figure 32: DPU Kernel Graph for ResNet-50

The second part describes each kernel in detail:

- **Kernel id**: ID of the current kernel. Every kernel has a unique id assigned by DNNC.
- **Kernel name**: Name of the current kernel. Each kernel supported by the DPU has a corresponding ELF object file with a name that is the same as the kernel name prefixed by `dpu_` with extension `.elf`. For example, the `dpu_resnet50_0.elf` and `dpu_resnet50_2.elf` correspond to kernels with names `resnet50_0` and `resnet50_2`, respectively. The kernel name will be used in the application code, allowing N^{2}Cube to identify different kernels correctly.
- **Type**: The kernel type. Three types of kernel are supported by DNNC, see Table 16: DNNC Kernel Types for details.
- **Nodes:** All nodes included in the current kernel. For kernels supported by the DPU, "NA" is used to avoid printing all names.
- **Input nodes**: All input nodes of the current kernel. For kernels not supported by the DPU, the user must get the output of the preceding kernel through output nodes and feed them into input nodes of the current node using APIs provided by N^{2}Cube.
- **Output nodes**: All output nodes of the current kernel. The address and size of output nodes can be extracted using APIs provided by N^{2}Cube.

Table 15: Operations Supported by the DPU

Type	Limitations
Convolution	Support kernel-w and kernel-h values ranging from 1 to 8 in any combination.
ReLU	No limitations
Pooling	Max-pooling is supported, and kernel size must be 2x2 or 3x3. Average-pooling is supported by some version DPU IPs. Use "dexplorer -w" to check if it is enabled or not.
Concat	Only concatenation in channel axis is supported.
Element-wise	No limitations
InnerProduct	No limitations

Note: Operations supported by the DPU core might vary due to the differences in the amount of available hardware logic resources between FPGA devices.

Table 16: DNNC Kernel Types

Type	Description
DPUKernel	Kernel running on the DPU.
CPUKernel	Kernel running on a CPU; must be implemented by the user.
ParamKernel	Same as CPUKernel; except that DNNC will also generate weights and bias parameters.

Programming Model

Understanding the DPU programming model makes it easier to develop and deploy deep learning applications on the DPU platform. The related concepts include “DPU Kernel”, “DPU Task”, “DPU Node” and “DPU Tensor”. DPU kernel and task are two core concepts for DPU programming.

DPU Kernel

After being compiled by Deep Neural Network Compiler (DNNC) compiler, the neural network model is transformed into an equivalent DPU assembly file, which is then assembled into one ELF object file by Deep Neural Network Assembler (DNNAS). DPU ELF object file is regarded as DPU kernel, which becomes one execution unit from the perspective of runtime N^2Cube after invoking the API `dpuLoadKernel()`. N^2Cube will load DPU kernel, including DPU instructions and network parameters, into the DPU dedicated memory space and allocate hardware resources. After that, each DPU kernel can be instantiated into several DPU tasks by calling `dpuCreateTask()` to enable the multithreaded programming.

DPU Task

Each DPU task is a running entity of a DPU kernel. It has its own private memory space so that multithreaded applications can be used to process several tasks in parallel to improve efficiency and system throughput.

DPU Node

A DPU node is considered a basic element of a network model deployed on the DPU. Each DPU node is associated with input, output and some parameters. Every DPU node has a unique name to allow APIs exported by DNNDK to access its information.

There are three types of nodes: boundary input node, boundary output node, and internal node.

- A **boundary input node** is a node that does not have any precursor in the DPU kernel topology; it is usually the first node in a kernel. Sometimes there might be multiple boundary input nodes in a kernel.
- A **boundary output node** is a node that does not have any successor nodes in the DPU kernel topology.
- All other nodes that are not both boundary input nodes and boundary output nodes are considered as internal nodes.

After compilation, DNNC gives information about the kernel and its boundary input/output nodes. The following figure shows an example after compiling Inception-v1; `conv1_7x7_s2` is the boundary input node, and `inception_5b_output` is the boundary output node for DPU Kernel 0.

```
Compiling network: inception_v1
[DNNC][Warning] Only max pooling is supported, but [pool5_7x7_s1] layer has average pooling type.
[DNNC][Warning] layer [pool5_7x7_s1] is not supported in DPU, deploy it in CPU instead.
[DNNC][Warning] layer [loss3_loss3] is not supported in DPU, deploy it in CPU instead.
[DNNC][Warning] Fail to convert gv file to jpg because 'dot' is not installed in current system. Try to
 install it using 'sudo apt-get install graphviz'. The original gv file is saved in 'inception_v1_kerne
l_graph.gv'.

DNNC Kernel Information

1. Overview
kernel numbers  : 4
kernel topology : inception_v1_kernel_graph.jpg

2. Kernel Description in Detail
kernel id       : 0
kernel name     : inception_v1_0
type            : DPUKernel
nodes           : NA
input node(s)   : conv1_7x7_s2(0)
output node(s)  : inception_5b_output(0)

kernel id       : 1
kernel name     : inception_v1_1
type            : CPUKernel
nodes           : NA
input node(s)   : pool5_7x7_s1
output node(s)  : pool5_7x7_s1

kernel id       : 2
kernel name     : inception_v1_2
type            : DPUKernel
nodes           : NA
input node(s)   : loss3_classifier(0)
output node(s)  : loss3_classifier(0)

kernel id       : 3
kernel name     : inception_v1_3
type            : CPUKernel
nodes           : NA
input node(s)   : loss3_loss3
output node(s)  : loss3_loss3
```

Figure 33: Sample DNNC Compilation Log

When using `dpuGetInputTensor*/dpuSetInputTensor*`, the `nodeName` parameter is required to specify the boundary input node. When a `nodeName` that does not correspond to a valid boundary input node is used, DNNDK gives an error message:

```
[DNNDK] Node "inception_5b_output" is not a Boundary Input Node for Kernel
inception_v1_0.
[DNNDK] Refer to DNNDK user guide for more info about "Boundary Input Node".
```

Similarly, when using `dpuGetOutputTensor*/dpuSetOutputTensor*`, an error is generated when a "`nodeName`" that does not correspond to a valid boundary output node is used:

```
[DNNDK] Node "conv1_7x7_s2" is not a Boundary Output Node for Kernel
inception_v1_0.
[DNNDK] Please refer to DNNDK user guide for more info about "Boundary Output
Node".
```

DPU Tensor

DPU tensor is a collection of multi-dimensional data that is used to store information while running. Tensor properties (such as height, width, channel, and so on) can be obtained using APIs exported by DNNDK.

Programming Interface

DNNDK offers a set of lightweight C/C++ programming APIs encapsulated in several libraries to smooth the deep learning application development for the DPU. For detailed description of each API, refer to Chapter 12: Programming APIs.

It is common to exchange data between a CPU and the DPU when programming for the DPU. For example, data preprocessed by a CPU can be sent to the DPU for acceleration; and the output produced by the DPU might need to be copied back to a CPU for further processing. To handle this type of operation, DNNDK provides a set of APIs to make it easy for data exchange. Some examples are shown below.

APIs to set input tensor for a computation layer or node:

- `dpuSetInputTensor()`
- `dpuSetInputTensorInCHWInt8()`
- `dpuSetInputTensorInCHWFP32()`
- `dpuSetInputTensorInHWCInt8()`
- `dpuSetInputTensorInHWCFP32()`

APIs to get output tensor from a n a computation layer or node:

- `dpuGetOutputTensor()`
- `dpuGetOutputTensorInCHWInt8()`
- `dpuGetOutputTensorInCHWFP32()`
- `dpuGetOutputTensorInHWCInt8()`
- `dpuGetOutputTensorInHWCFP32()`

Deep learning applications developed for the DPU are heterogeneous programs, which will contain code running on a host CPU (such as x86 or Arm), and code running on the DPU. The compilation process for DPU-accelerated deep learning applications is depicted in the following figure.

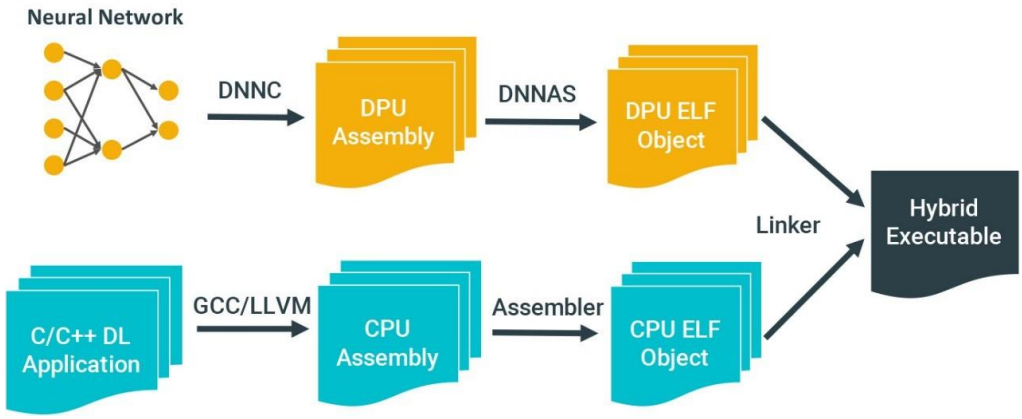

Figure 34: Hybrid Compilation Process

Code that runs on a CPU is programmed in the C/C++ language, which is then processed by a compiler, such as GCC or LLVM. At the same time, computation intensive neural networks are compiled by Deep Neural Network Compiler (DNNC)into DPU binary code for acceleration. In the final stage, CPU and DPU code are linked together by a linker (e.g. GCC) to produce a single hybrid binary executable, which contains all the required information for heterogeneously running on both CPU and DPU.

Note: The hybrid executable cannot be stripped using a strip tool. Otherwise, a "DPU Kernel load failed" error occurs.

As described in previous sections, a deep learning application is compiled and linked into a hybrid binary executable. It looks like the same as normal applications. Under the hood, a standard Linux loader and the DPU loader handles how to load and run the hybrid deep learning application. The running model of DPU deep learning applications is shown in the following figure. It is composed of DPU Loader, DPU profiler, DPU runtime library, and DPU driver.

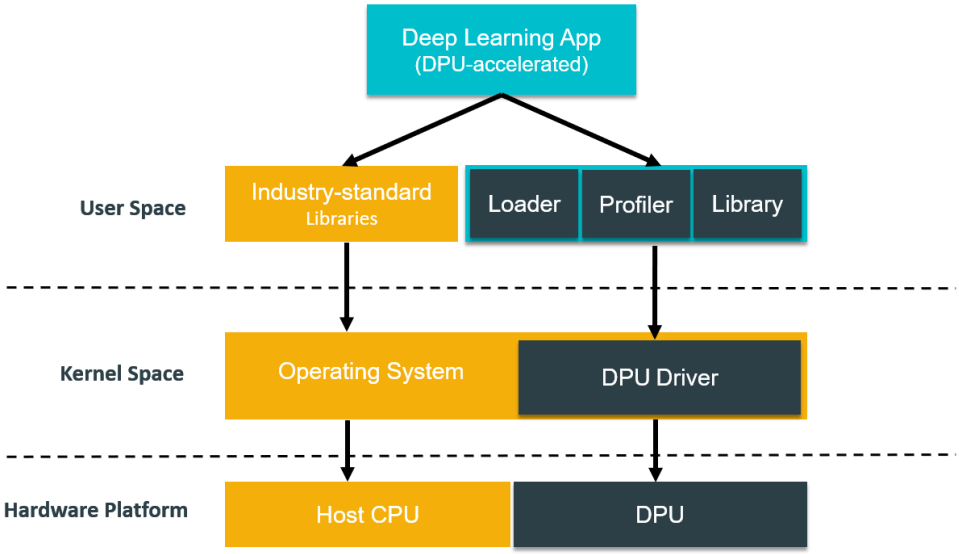

Figure 35: DPU Runtime

The DPU loader handles the transfer of DPU kernels from the hybrid ELF executable into memory and dynamically relocates the memory of DPU code.

This chapter describes the tools included within the DNNDK package, including DPU configuration and checking tool DExplorer, and profiling tool DSight.

DExplorer

DExplorer is a utility running on the target board. It provides DPU running mode configuration, DNNDK version checking, DPU status checking, and DPU core signature checking. The following figure shows the help information about the usage of DExplorer.

```
Usage: dexplorer <option>
 Options are:
  -v  --version    Display version info for each DNNDK component
  -s  --status     Display the status of DPU cores
  -w  --whoami     Display the info of DPU cores
  -m  --mode       Specify DNNDK N2Cube running mode: normal, profile, or debug
  -t  --timeout    Specify DPU timeout limitation in seconds under integer range of [1, 100]
  -h  --help       Display this information
```

Figure 36: DExplorer Usage Options

Check DNNDK version

Running "dexplore -v" will display version information for each component in DNNDK, including N^{2}cube, DPU driver, DExplorer, and DSight.

Check DPU status

DExplorer provides DPU status information, including running mode of N^{2}cube, DPU timeout threshold, DPU debugging level, DPU core status, DPU register information, DPU memory resource and utilization. Figure 37 shows a screenshot of DPU status.

```
root@dp-n1:~# dexplorer -s
[DPU cache]
Enabled

[DPU mode]
normal

[DPU timeout limitation (in seconds)]
5

[DPU Debug Info]
Debug level     : 9
Core 0 schedule : 0
Core 0 interrupt: 0

[DPU Resource]
DPU Core        : 0
State           : Idle
PID             : 0
TaskID          : 0
Start           : 0
End             : 0

[DPU Registers]
VER             : 0x05c1c6bd
RST             : 0x000000ff
ISR             : 0x00000000
IMR             : 0x00000000
IRSR            : 0x00000000
ICR             : 0x00000000

DPU Core        : 0
HP_CTL          : 0x07070f0f
ADDR_IO         : 0x00000000
ADDR_WEIGHT     : 0x00000000
ADDR_CODE       : 0x00000000
ADDR_PROF       : 0x00000000
```

Figure 37: DExplorer Status

Configure DPU Running Mode

DNNDK runtime N^{2}cube supports three kinds of DPU execution modes to help developers to debug and profile DNNDK applications.

Normal Mode

In normal mode, the DPU application can get the best performance without any overhead.

Profile Mode

In this mode, the DPU will turn on the profiling switch. When running deep learning applications in profile mode, N^{2}cube will output to the console the performance data layer by layer while executing the neural network; at the same time, a profile with the name "dpu_trace_[PID].prof" will be produced under the current folder. This file can be used with the DSight tool. The following figure shows a screenshot of this mode.

```
[DNNDK] Performance profile - DPU Kernel "resnet50_0" DPU Task "resnet50_0-5"
================================================================================================
 ID                      NodeName Workload(MOP) Mem(MB) RunTime(ms) Perf(GOPS) Utilization    MB/S
  0                         conv1        236.0     0.4        5.28       44.7       19.4%      67
  1                 res2a_branch2a         25.7     0.4        0.23      113.2       49.2%    1719
  2                  res2a_branch1        102.8     1.0        0.95      108.5       47.2%    1044
  3                 res2a_branch2b        231.2     0.4        1.34      172.0       74.8%     314
  4                 res2a_branch2c        102.8     1.0        1.62       63.3       27.5%     616
  5                 res2b_branch2a        102.8     1.0        0.65      159.3       69.3%    1518
  6                 res2b_branch2b        231.2     0.4        1.34      172.0       74.8%     314
  7                 res2b_branch2c        102.8     1.0        1.62       63.6       27.6%     619
  8                 res2c_branch2a        102.8     1.0        0.64      159.8       69.5%    1523
  9                 res2c_branch2b        231.2     0.4        1.35      171.9       74.7%     313
 10                 res2c_branch2c        102.8     1.0        1.62       63.3       27.5%     616
 11                 res3a_branch2a         51.4     0.9        0.41      125.3       54.5%    2197
 12                  res3a_branch1        205.5     1.3        1.49      137.8       59.9%     870
 13                 res3a_branch2b        231.2     0.3        1.14      202.6       88.1%     294
 14                 res3a_branch2c        102.8     0.6        1.22       84.6       36.8%     466
 15                 res3b_branch2a        102.8     0.5        0.58      176.6       76.8%     939
 16                 res3b_branch2b        231.2     0.3        1.14      202.6       88.1%     294
 17                 res3b_branch2c        102.8     0.6        1.21       84.6       36.8%     466
 18                 res3c_branch2a        102.8     0.5        0.58      176.0       76.5%     936
 19                 res3c_branch2b        231.2     0.3        1.14      202.6       88.1%     294
 20                 res3c_branch2c        102.8     0.6        1.21       84.6       36.8%     466
 21                 res3d_branch2a        102.8     0.5        0.58      176.0       76.5%     936
 22                 res3d_branch2b        231.2     0.3        1.14      202.5       88.0%     294
 23                 res3d_branch2c        102.8     0.6        1.21       84.9       36.9%     468
 24                 res4a_branch2a         51.4     0.6        0.49      105.9       46.1%    1164
 25                  res4a_branch1        205.5     1.1        2.01      102.0       44.4%     551
 26                 res4a_branch2b        231.2     0.7        1.34      172.9       75.2%     497
 27                 res4a_branch2c        102.8     0.5        1.01      101.8       44.3%     508
 28                 res4b_branch2a        102.8     0.5        0.71      145.1       63.1%     700
 29                 res4b_branch2b        231.2     0.7        1.34      172.9       75.2%     497
 30                 res4b_branch2c        102.8     0.5        1.00      102.9       44.7%     513
 31                 res4c_branch2a        102.8     0.5        0.71      144.5       62.8%     697
 32                 res4c_branch2b        231.2     0.7        1.34      172.7       75.1%     496
 33                 res4c_branch2c        102.8     0.5        1.01      101.7       44.2%     508
 34                 res4d_branch2a        102.8     0.5        0.71      145.3       63.2%     701
 35                 res4d_branch2b        231.2     0.7        1.34      172.3       74.9%     495
 36                 res4d_branch2c        102.8     0.5        1.02      100.9       43.9%     504
 37                 res4e_branch2a        102.8     0.5        0.71      145.3       63.2%     701
 38                 res4e_branch2b        231.2     0.7        1.34      172.8       75.1%     496
 39                 res4e_branch2c        102.8     0.5        1.01      101.8       44.3%     508
 40                 res4f_branch2a        102.8     0.5        0.70      146.6       63.7%     707
 41                 res4f_branch2b        231.2     0.7        1.34      172.8       75.1%     496
 42                 res4f_branch2c        102.8     0.5        1.01      101.5       44.1%     507
 43                 res5a_branch2a         51.4     0.7        0.70       73.6       32.0%    1044
 44                  res5a_branch1        205.5     2.3        2.81       73.3       31.9%     835
 45                 res5a_branch2b        231.2     2.3        1.77      130.6       56.8%    1304
 46                 res5a_branch2c        102.8     1.2        1.32       77.8       33.8%     875
 47                 res5b_branch2a        102.8     1.1        1.01      101.8       44.3%    1120
 48                 res5b_branch2b        231.2     2.3        1.77      130.7       56.8%    1304
 49                 res5b_branch2c        102.8     1.2        1.33       77.3       33.6%     869
 50                 res5c_branch2a        102.8     1.1        1.01      101.9       44.3%    1121
 51                 res5c_branch2b        231.2     2.3        1.78      130.0       56.5%    1298
 52                 res5c_branch2c        102.8     1.2        1.28       80.2       34.9%     900

            Total Nodes In Avg:
                            All        7711.9    44.4       64.62      119.3       51.9%     687
================================================================================================
```

Figure 38: N²Cube Profile Mode

Debug Mode

In this mode, the DPU dumps raw data for each DPU computation node during execution, including DPU instruction code in binary format, network parameters, DPU input tensor and output tensor. This makes it easy to debug and locate issues in a DPU application.

Note: Profile mode and debug mode are only available to neural network models compiled into debug mode DPU ELF objects by the Deep Neural Network Compiler (DNNC) compiler.

DPU Signature

New DPU cores have been introduced to meet various deep learning acceleration requirements across different Xilinx® FPGA devices. For example, DPU architectures B1024F, B1152F, B1600F, B2304F, and B4096F are available. Each DPU architecture can implement a different version of the DPU instruction set (which is named as a DPU target version) to support the rapid improvements in deep learning algorithms.

The DPU signature refers to the specification information of a specific DPU architecture version, covering target version, working frequency, DPU core numbers, harden acceleration modules (such as softmax), etc. The `-w` option can be used to check the DPU signature. Figure 39 shows a screenshot of a sample run of `dexplorer -w`.

```
root@zcu102:~# dexplorer -w
[DPU IP Spec]
IP  Timestamp   : 2019-04-16 11:15:00
DPU Core Count  : 3

[DPU Core List]
DPU Core        : #0
DPU Enabled     : Yes
DPU Arch        : B4096F
DPU Target      : v1.4.0
DPU Freqency    : 333 MHz
DPU Features    : Avg-Pooling, LeakyReLU/ReLU6, Depthwise Conv
DPU Core        : #1
DPU Enabled     : Yes
DPU Arch        : B4096F
DPU Target      : v1.4.0
DPU Freqency    : 333 MHz
DPU Features    : Avg-Pooling, LeakyReLU/ReLU6, Depthwise Conv
DPU Core        : #2
DPU Enabled     : Yes
DPU Arch        : B4096F
DPU Target      : v1.4.0
DPU Freqency    : 333 MHz
DPU Features    : Avg-Pooling, LeakyReLU/ReLU6, Depthwise Conv

[DPU Extension List]
Extension Softmax
Enabled         : Yes
```

Figure 39: Sample DPU Signature

DSight

DSight is the DNNDK performance profiling tool. It is a visual performance analysis tool for neural network model profiling. The following figure shows its usage.

```
root@xlnx:~# dsight -h
Usage: dsight <option>
 Options are:
  -p  --profile    Specify DPU trace file for profiling
  -v  --version    Display DSight version info
  -h  --help       Display this information
```

Figure 40: DSight Help Info

By processing the log file produced by the N^{2}cube tracer, DSight can generate an html file, which provides a visual analysis interface for the neural network model. The steps below describe how to use the profiler:

1. Set N^{2}Cube to profile mode using the command `dexplorer -m profile`.
2. Run the deep learning application. When finished, a profile file with the name `dpu_trace_[PID].prof` is generated for further checking and analysis (`PID` is the process ID of the deep learning application).

3. Generate the html file with the DSight tool using the command: `dsight -p dpu_trace_[PID].prof`. An html file with the name `dpu_trace_[PID].html` is generated.
4. Open the generated html file with web browser.

Figure 41: DSight Profiling Charts

DNNDK provides a lightweight set of C/C++ programming APIs for deep learning application developers. It consists of two dynamic libraries, DPU runtime N^{2}Cube library `libn2cube` and DPU utility library `libdputils`. The exported APIs for them are individually contained in header file `n2cube.h` and `dputils.h`, which are described in detail in this chapter.

Notes:

- For simplification, you only need to include the header file `dnndk.h` into DNNDK applications. It includes both `n2cube.h` and `dputils.h` by default.
- The subsequent sections give detailed description to each API. Item "**AVAILABILITY**" indicates which DNNDK version the corresponding API became available.

Library libn2cube

Overview

Library libn2cube is the DNNDK core library. It implements the functionality of DPU loader, and encapsulates the system calls to invoke the DPU driver for DPU Task scheduling, monitoring, profiling, and resources management. The exported APIs are briefly summarized in the table below.

NAME	libn2cube.so
DESCRIPTION	DPU runtime library
ROUTINES	dpuOpen() - Open & initialize the usage of DPU device dpuClose() - Close & finalize the usage of DPU device dpuLoadKernel() - Load a DPU Kernel and allocate DPU memory space for its Code/Weight/Bias segments dpuDestroyKernel() - Destroy a DPU Kernel and release its associated resources dpuCreateTask() - Instantiate a DPU Task from one DPU Kernel, allocate its private working memory buffer and prepare for its execution context dpuRunTask() - Launch the running of DPU Task

	dpuDestroyTask() - Remove a DPU Task, release its working memory buffer and destroy associated execution context **dpuEnableTaskDump()** - Enable dump facility of DPU Task while running for debugging purpose **dpuEnableTaskProfile()** - Enable profiling facility of DPU Task while running to get its performance metrics **dpuGetTaskProfile()** - Get the execution time of DPU Task **dpuGetNodeProfile()** - Get the execution time of DPU Node **dpuGetInputTensorCnt()** - Get total number of input Tensor of one DPU Task **dpuGetInputTensor()** - Get input Tensor of one DPU Task **dpuGetInputTensorAddress()** - Get the start address of one DPU Task's input Tensor **dpuGetInputTensorSize()** - Get the size (in byte) of one DPU Task's input Tensor **dpuGetInputTensorScale()** - Get the scale value of one DPU Task's input Tensor **dpuGetInputTensorHeight()** - Get the height dimension of one DPU Task's input Tensor **dpuGetInputTensorWidth()** - Get the width dimension of one DPU Task's input Tensor **dpuGetInputTensorChannel()** - Get the channel dimension of one DPU Task's input Tensor **dpuGetOutputTensorCnt()** - Get total number of output Tensor of one DPU Task **dpuGetOutputTensor()** - Get output Tensor of one DPU Task **dpuGetOutputTensorAddress()** - Get the start address of one DPU Task's output Tensor

	dpuGetOutputTensorSize() - Get the size in byte of one DPU Task's output Tensor **dpuGetOutputTensorScale()** - Get the scale value of one DPU Task's output Tensor **dpuGetOutputTensorHeight()** - Get the height dimension of one DPU Task's output Tensor **dpuGetOutputTensorWidth()** - Get the width dimension of one DPU Task's output Tensor **dpuGetOutputTensorChannel()** - Get the channel dimension of one DPU Task's output Tensor **dpuGetTensorSize()** - Get the size of one DPU Tensor **dpuGetTensorAddress()** - Get the start address of one DPU Tensor **dpuGetTensorScale()** - Get the scale value of one DPU Tensor **dpuGetTensorHeight()** - Get the height dimension of one DPU Tensor **dpuGetTensorWidth()** - Get the width dimension of one DPU Tensor **dpuGetTensorChannel()** - Get the channel dimension of one DPU Tensor **dpuSetInputTensorInCHWInt8()** - Set DPU Task's input Tensor with data stored under Caffe order (channel/height/width) in INT8 format **dpuSetInputTensorInCHWFP32()** - Set DPU Task's input Tensor with data stored under Caffe order (channel/height/width) in FP32 format **dpuSetInputTensorInHWCInt8()** - Set DPU Task's input Tensor with data stored under DPU order (height/width/channel) in INT8 format **dpuSetInputTensorInHWCFP32()** - Set DPU Task's input Tensor with data stored under DPU order (channel/height/width) in FP32 format **dpuGetOutputTensorInCHWInt8()** - Get DPU Task's output Tensor and store them under Caffe order (channel/height/width) in INT8 format

	dpuGetOutputTensorInCHWFP32() - Get DPU Task's output Tensor and store them under Caffe order (channel/height/width) in FP32 format **dpuGetOutputTensorInHWCInt8()** - Get DPU Task's output Tensor and store them under DPU order (channel/height/width) in INT8 format **dpuGetOutputTensorInHWCFP32()** - Get DPU Task's output Tensor and store them under DPU order (channel/height/width) in FP32 format **dpuRunSoftmax () -** Perform softmax calculation for the input elements and save the results to output memory buffer. **dpuSetExceptionMode()** - Set the exception handling mode for DNNDK runtime N^2Cube. **dpuGetExceptionMode()** - Get the exception handling mode for runtime N^2Cube. **dpuGetExceptionMessage()** - Get the error message from error code (always negative value) returned by N^2Cube APIs.
INCLUDE FILE	**n2cube.h**

APIs

The prototype and parameter for each API of library libn2cube are shown in detail in the following sections.

dpuOpen()

NAME	dpuOpen()
SYNOPSIS	int dpuOpen()
ARGUMENTS	None
DESCRIPTION	Attach and open DPU device file "/dev/dpu" before the utilization of DPU resources.

RETURNS	0 on success, or negative value in case of failure. Error message “Fail to open DPU device” is reported if any error takes place.
SEE ALSO	dpuClose()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuClose()

NAME	dpuClose()
SYNOPSIS	int dpuClose()
ARGUMENTS	None
DESCRIPTION	Detach and close DPU device file “/dev/dpu” after utilization of DPU resources.
RETURNS	0 on success, or negative error ID in case of failure. Error message “Fail to close DPU device” is reported if any error takes place.
SEE ALSO	dpuOpen()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuLoadKernel()

NAME	dpuLoadKernel()	
SYNOPSIS	DPUKernel *dpuLoadKernel (const char *netName)	
ARGUMENTS	netName	The pointer to neural network name. Use the names produced by Deep Neural Network Compiler (DNNC) after the compilation of neural network. For each DL application, perhaps there are many DPU Kernels existing in its hybrid CPU+DPU binary executable. For each DPU Kernel, it has one unique name for differentiation purpose.
DESCRIPTION	Load a DPU Kernel for the specified neural network from hybrid CPU+DPU binary executable into DPU memory space, including Kernel's DPU instructions, weight and bias.	
RETURNS	The pointer to the loaded DPU Kernel on success, or report error in case of any failure.	
SEE ALSO	dpuDestroyKernel()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuDestroyKernel()

NAME	dpuDestroyKernel()	
SYNOPSIS	int dpuDestroyKernel (DPUKernel *kernel)	
ARGUMENTS	kernel	The pointer to DPU kernel to be destroyed.
DESCRIPTION	Destroy a DPU kernel and release its related resources.	
RETURNS	0 on success, or report error in case of any failure.	
SEE ALSO	dpuLoadKernel()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuCreateTask()

NAME	dpuCreateTask()
SYNOPSIS	int dpuCreateTask (DPUKernel *kernel, Int mode);

ARGUMENTS	kernel	The pointer to DPU Kernel.
	mode	The running mode of DPU Task. There are 3 available modes: **MODE_NORMAL**: default mode identical to the mode value "0". **MODE_PROF**: output profiling information layer by layer while running of DPU Task, which is useful for performance analysis. **MODE_DUMP**: dump the raw data for DPU Task's CODE/BIAS/WEIGHT/INPUT/OUTPUT layer by layer. The file names are in the following format ("netName" refers to the name of neural network; "layerName" refers to the index ID of layer (or Node)): For **CODE**: netName_layerName_code.txt For **WEIGHT**: netName_layerName_w.txt For **BIAS**: netName_layerName_b.txt For **INPUT**: netName_layerName_i.txt For **OUTPUT**: netName_layerName_o.txt NOTE: profiling and dump functionality is available only for DPU Kernel generated by DNNC in debug mode.
DESCRIPTION	Instantiate a DPU Task from DPU Kernel and allocate corresponding DPU memory buffer.	
RETURNS	0 on success, or report error in case of any failure.	
SEE ALSO		
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuDestroyTask()

NAME	dpuDestroyTask()	
SYNOPSIS	int dpuDestroyTask (DPUTask *task)	
ARGUMENTS	task	The pointer to DPU Task to be destroyed.
DESCRIPTION	Destroy a DPU Task and release its related resources.	
RETURNS	0 on success, or negative value in case of any failure.	
SEE ALSO	dpuCreateTask()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuRunTask()

NAME	dpuRunTask ()	
SYNOPSIS	int dpuRunTask (DPUTask *task);	
ARGUMENTS	task	The pointer to DPU Task.
DESCRIPTION	Launch the running of DPU Task.	
RETURNS	0 on success, or negative value in case of any failure.	

SEE ALSO	
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuEnableTaskProfile()

NAME	dpuEnableTaskProfile()	
SYNOPSIS	int dpuEnableTaskProfile (DPUTask *task);	
ARGUMENTS	task	The pointer to DPU Task.
DESCRIPTION	Set DPU Task in profiling mode. Note that profiling functionality is available only for DPU Kernel generated by DNNC in debug mode.	
RETURNS	0 on success, or report error in case of any failure.	
SEE ALSO	dpuCreateTask() dpuEnableTaskDump()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuEnableTaskDump()

NAME	dpuEnableTaskDump()	
SYNOPSIS	int dpuEnableTaskDump (DPUTask *task);	
ARGUMENTS	task	The pointer to DPU Task.
DESCRIPTION	Set DPU Task in dump mode. Note that dump functionality is available only for DPU Kernel generated by DNNC in debug mode.	
RETURNS	0 on success, or report error in case of any failure.	
SEE ALSO	dpuCreateTask() dpuEnableTaskProfile()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetTaskProfile()

NAME	dpuGetTaskProfile()	
SYNOPSIS	int dpuGetTaskProfile (DPUTask *task);	
ARGUMENTS	task	The pointer to DPU Task.

DESCRIPTION	Get DPU Task's execution time (us) after its running.
RETURNS	The DPU Task's execution time (us) after its running.
SEE ALSO	dpuGetNodeProfile()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuGetNodeProfile()

NAME	dpuGetNodeProfile()	
SYNOPSIS	int dpuGetNodeProfile (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name
DESCRIPTION	Get DPU Node's execution time (us) after DPU Task completes its running.	
RETURNS	The DPU Node's execution time(us) after DPU Task completes its running. Note that this functionality is available only for DPU Kernel generated by DNNC in debug mode.	
SEE ALSO	dpuGetTaskProfile()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetInputTensorCnt()

NAME	dpuGetInputTensorCnt()	
SYNOPSIS	Int dpuGetInputTensorCnt (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get total number of input Tensor of one DPU Task's	
RETURNS	The total number of input tensor for specified Node.	
SEE ALSO	dpuGetOutputTensorCnt()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	V2.06	

dpuGetInputTensor()

NAME	dpuGetInputTensor()	
SYNOPSIS	DPUTensor*dpuGetInputTensor (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single input tensor for the Node, with default value as 0.
DESCRIPTION	Get DPU Task's input Tensor.	
RETURNS	The pointer to Task's input Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetOutputTensor()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuGetInputTensor()

SYNOPSIS	DPUTensor*dpuGetInputTensor (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get DPU Task's input Tensor.	
RETURNS	The pointer to Task's input Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetOutputTensor()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetInputTensorAddress()

NAME	dpuGetInputTensorAddress()	
SYNOPSIS	int8_t* dpuGetInputTensorAddress (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single input tensor for the Node, with default value as 0.
DESCRIPTION	Get the start address of DPU Task's input Tensor.	
RETURNS	The start addresses to Task's input Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetOutputTensorAddress()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuGetInputTensorAddress()	
SYNOPSIS	int8_t* dpuGetInputTensorAddress (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get the start address of DPU Task's input Tensor.	
RETURNS	The start addresses to Task's input Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetOutputTensorAddress()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetInputTensorSize()

NAME	dpuGetInputTensorSize()	
SYNOPSIS	int dpuGetInputTensorSize (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single input tensor for the Node, with default value as 0.
DESCRIPTION	Get the size (in Byte) of DPU Task's input Tensor.	
RETURNS	The size of Task's input Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetOutputTensorSize()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuGetInputTensorSize()	
SYNOPSIS	int dpuGetInputTensorSize (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get the size (in Byte) of DPU Task's input Tensor.	
RETURNS	The size of Task's input Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetOutputTensorSize()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetInputTensorScale()

NAME	dpuGetInputTensorScale()	
SYNOPSIS	float dpuGetInputTensorScale (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single input tensor for the Node, with default value as 0.
DESCRIPTION	Get the scale value of DPU Task's input Tensor. For each DPU input Tensor, it has one unified scale value indicating its quantization information for reformatting between data types of INT8 and FP32.	
RETURNS	The scale value of Task's input Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetOutputTensorScale()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuGetInputTensorScale()	
SYNOPSIS	float dpuGetInputTensorScale (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get the scale value of DPU Task's input Tensor. For each DPU input Tensor, it has one unified scale value indicating its quantization information for reformatting between data types of INT8 and FP32.	
RETURNS	The scale value of Task's input Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetOutputTensorScale()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetInputTensorHeight()

NAME	dpuGetInputTensorHeight()	
SYNOPSIS	int dpuGetInputTensorHeight (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single input tensor for the Node, with default value as 0.
DESCRIPTION	Get the height dimension of DPU Task's input Tensor.	
RETURNS	The height dimension of Task's input Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensorWidth() dpuGetInputTensorChannel() dpuGetOutputTensorHeight() dpuGetOutputTensorWidth() dpuGetOutputTensorChannel()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuGetInputTensorHeight()	
SYNOPSIS	int dpuGetInputTensorHeight (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get the height dimension of DPU Task's input Tensor.	
RETURNS	The height dimension of Task's input Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensorWidth() dpuGetInputTensorChannel() dpuGetOutputTensorHeight() dpuGetOutputTensorWidth() dpuGetOutputTensorChannel()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetInputTensorWidth()

NAME	dpuGetInputTensorWidth()	
SYNOPSIS	int dpuGetInputTensorWidth (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single input tensor for the Node, with default value as 0.
DESCRIPTION	Get the width dimension of DPU Task's input Tensor.	
RETURNS	The width dimension of Task's input Tensor on success, or report error in case of any failure.	

SEE ALSO	dpuGetInputTensorHeight() dpuGetInputTensorChannel() dpuGetOutputTensorHeight() dpuGetOutputTensorWidth() dpuGetOutputTensorChannel()
INCLUDE FILE	n2cube.h
AVAILABILITY	v2.06

NAME	dpuGetInputTensorWidth()	
SYNOPSIS	int dpuGetInputTensorWidth (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get the width dimension of DPU Task's input Tensor.	
RETURNS	The width dimension of Task's input Tensor on success, or report error in case of any failure.	

SEE ALSO	dpuGetInputTensorHeight() dpuGetInputTensorChannel() dpuGetOutputTensorHeight() dpuGetOutputTensorWidth() dpuGetOutputTensorChannel()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuGetInputTensorChannel()

NAME	dpuGetInputTensorChannel()	
SYNOPSIS	int dpuGetInputTensorChannel (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single input tensor for the Node, with default value as 0.
DESCRIPTION	Get the channel dimension of DPU Task's input Tensor.	

RETURNS	The channel dimension of Task's input Tensor on success, or report error in case of any failure.
SEE ALSO	dpuGetInputTensorHeight() dpuGetInputTensorWidth() dpuGetOutputTensorHeight() dpuGetOutputTensorWidth() dpuGetOutputTensorChannel()
INCLUDE FILE	n2cube.h
AVAILABILITY	v2.06

NAME	dpuGetInputTensorChannel()	
SYNOPSIS	int dpuGetInputTensorChannel (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's input Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get the channel dimension of DPU Task's input Tensor.	

RETURNS	The channel dimension of Task's input Tensor on success, or report error in case of any failure.
SEE ALSO	dpuGetInputTensorHeight() dpuGetInputTensorWidth() dpuGetOutputTensorHeight() dpuGetOutputTensorWidth() dpuGetOutputTensorChannel()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuGetOutputTensorCnt()

NAME	dpuGetOutputTensorCnt()	
SYNOPSIS	Int dpuGetOutputTensorCnt (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get total number of output Tensor for the DPU Task.	

RETURNS	The total number of output tensor for the DPU Task.
SEE ALSO	dpuGetInputTensorCnt()
INCLUDE FILE	n2cube.h
AVAILABILITY	V2.06

dpuGetOutputTensor()

NAME	dpuGetOutputTensor()	
SYNOPSIS	DPUTensor*dpuGetOutputTensor (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single output tensor for the Node, with default value as 0.
DESCRIPTION	Get DPU Task's output Tensor.	
RETURNS	The pointer to Task's output Tensor on success, or report error in case of any failure.	

SEE ALSO	dpuGetInputTensor()
INCLUDE FILE	n2cube.h
AVAILABILITY	v2.06

NAME	dpuGetOutputTensor()	
SYNOPSIS	DPUTensor*dpuGetOutputTensor (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get DPU Task's output Tensor.	
RETURNS	The pointer to Task's output Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensor()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetOutputTensorAddress()

NAME	dpuGetOutputTensorAddress()	
SYNOPSIS	int8_t* dpuGetOutputTensorAddress (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single output tensor for the Node, with default value as 0.
DESCRIPTION	Get the start address of DPU Task's output Tensor.	
RETURNS	The start addresses to Task's output Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensorAddress()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuGetOutputTensorAddress()	
SYNOPSIS	int8_t* dpuGetOutputTensorAddress (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get the start address of DPU Task's output Tensor.	
RETURNS	The start addresses to Task's output Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensorAddress()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetOutputTensorSize()

NAME	dpuGetOutputTensorSize()	
SYNOPSIS	int dpuGetOutputTensorSize (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single output tensor for the Node, with default value as 0.
DESCRIPTION	Get the size (in Byte) of DPU Task's output Tensor.	
RETURNS	The size of Task's output Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensorSize()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuGetOutputTensorSize()	
SYNOPSIS	int dpuGetOutputTensorSize (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get the size (in Byte) of DPU Task's output Tensor.	
RETURNS	The size of Task's output Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensorSize()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetOutputTensorScale()

NAME	dpuGetOutputTensorScale()	
SYNOPSIS	float dpuGetOutputTensorScale (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single output tensor for the Node, with default value as 0.
DESCRIPTION	Get the scale value of DPU Task's output Tensor. For each DPU output Tensor, it has one unified scale value indicating its quantization information for reformatting between data types of INT8 and FP32.	
RETURNS	The scale value of Task's output Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensorScale()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuGetOutputTensorScale()	
SYNOPSIS	float dpuGetOutputTensorScale (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get the scale value of DPU Task's output Tensor. For each DPU output Tensor, it has one unified scale value indicating its quantization information for reformatting between data types of INT8 and FP32.	
RETURNS	The scale value of Task's output Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensorScale()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetOutputTensorHeight()

NAME	dpuGetOutputTensorHeight()	
SYNOPSIS	int dpuGetOutputTensorHeight (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single output tensor for the Node, with default value as 0.
DESCRIPTION	Get the height dimension of DPU Task's output Tensor.	
RETURNS	The height dimension of Task's output Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetOutputTensorWidth() dpuGetOutputTensorChannel() dpuGetInputTensorHeight() dpuGetInputTensorWidth() dpuGetInputTensorChannel()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuGetOutputTensorHeight()	
SYNOPSIS	int dpuGetOutputTensorHeight (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get the height dimension of DPU Task's output Tensor.	
RETURNS	The height dimension of Task's output Tensor on success, or report error in case of any failure.	
SEE ALSO	dpuGetOutputTensorWidth() dpuGetOutputTensorChannel() dpuGetInputTensorHeight() dpuGetInputTensorWidth() dpuGetInputTensorChannel()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetOutputTensorWidth()

NAME	dpuGetOutputTensorWidth()	
SYNOPSIS	int dpuGetOutputTensorWidth (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single output tensor for the Node, with default value as 0.
DESCRIPTION	Get the width dimension of DPU Task's output Tensor.	
RETURNS	The width dimension of Task's output Tensor on success, or report error in case of any failure.	

SEE ALSO	dpuGetOutputTensorHeight() dpuGetOutputTensorChannel() dpuGetInputTensorHeight() dpuGetInputTensorWidth() dpuGetInputTensorChannel()
INCLUDE FILE	n2cube.h
AVAILABILITY	v2.06

NAME	dpuGetOutputTensorWidth()	
SYNOPSIS	int dpuGetOutputTensorWidth (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get the width dimension of DPU Task's output Tensor.	
RETURNS	The width dimension of Task's output Tensor on success, or report error in case of any failure.	

SEE ALSO	dpuGetOutputTensorHeight() dpuGetOutputTensorChannel() dpuGetInputTensorHeight() dpuGetInputTensorWidth() dpuGetInputTensorChannel()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuGetOutputTensorChannel()

NAME	dpuGetOutputTensorChannel()	
SYNOPSIS	int dpuGetOutputTensorChannel (DPUTask *task, const char*nodeName, int idx = 0);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	idx	The index of a single output tensor for the Node, with default value as 0.
DESCRIPTION	Get the channel dimension of DPU Task's output Tensor.	

RETURNS	The channel dimension of Task's output Tensor on success, or report error in case of any failure.
SEE ALSO	dpuGetOutputTensorHeight() dpuGetOutputTensorWidth() dpuGetInputTensorHeight() dpuGetInputTensorWidth() dpuGetInputTensorChannel()
INCLUDE FILE	n2cube.h
AVAILABILITY	V2.06

NAME	dpuGetOutputTensorChannel()	
SYNOPSIS	int dpuGetOutputTensorChannel (DPUTask *task, const char*nodeName);	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
DESCRIPTION	Get the channel dimension of DPU Task's output Tensor.	

RETURNS	The channel dimension of Task's output Tensor on success, or report error in case of any failure.
SEE ALSO	dpuGetOutputTensorHeight() dpuGetOutputTensorWidth() dpuGetInputTensorHeight() dpuGetInputTensorWidth() dpuGetInputTensorChannel()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuGetTensorSize()

NAME	dpuGetTensorSize()	
SYNOPSIS	int dpuGetTensorSize (DPUTensor* tensor);	
ARGUMENTS	tensor	The pointer to DPU Tensor.
DESCRIPTION	Get the size (in Byte) of one DPU Tensor.	
RETURNS	The size of Tensor, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensorSize() dpuGetOutputTensorSize()	

INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuGetTensorScale()

NAME	dpuGetTensorScale()	
SYNOPSIS	float dpuGetTensorScale (DPUTensor* tensor);	
ARGUMENTS	tensor	The pointer to DPU Tensor.
DESCRIPTION	Get the scale value of one DPU Tensor.	
RETURNS	The scale value of Tensor, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensorScale() dpuGetOutputTensorScale()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetTensorHeight()

NAME	dpuGetTensorHeight()	
SYNOPSIS	float dpuGetTensorHeight (DPUTensor* tensor);	
ARGUMENTS	tensor	The pointer to DPU Tensor.
DESCRIPTION	Get the height dimension of one DPU Tensor.	
RETURNS	The height dimension of Tensor, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensorHeight() dpuGetOutputTensorHeight()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetTensorWidth()

NAME	dpuGetTensorWidth()	
SYNOPSIS	float dpuGetTensorWidth (DPUTensor* tensor);	
ARGUMENTS	tensor	The pointer to DPU Tensor.
DESCRIPTION	Get the width dimension of one DPU Tensor.	

RETURNS	The width dimension of Tensor, or report error in case of any failure.
SEE ALSO	dpuGetInputTensorWidth() dpuGetOutputTensorWidth()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuGetTensorChannel()

NAME	dpuGetTensorChannel()	
SYNOPSIS	float dpuGetTensorChannel (DPUTensor* tensor);	
ARGUMENTS	tensor	The pointer to DPU Tensor.
DESCRIPTION	Get the channel dimension of one DPU Tensor.	
RETURNS	The channel dimension of Tensor, or report error in case of any failure.	
SEE ALSO	dpuGetInputTensorChannel() dpuGetOutputTensorChannel()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuSetInputTensorInCHWInt8()

NAME	dpuSetInputTensorInCHWInt8()	
SYNOPSIS	int dpuSetInputTensorInCHWInt8 (DPUTask *task, const char*nodeName, int8_t *data, int size, int idx = 0)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node's name.
	data	The pointer to the start address of input data.
	size	The size (in Byte) of input data to be set.
	idx	The index of a single input tensor for the Node, with default value as 0.
DESCRIPTION	Set DPU Task input Tensor with data from a CPU memory block. Data is in type of INT8 and stored in Caffe Blob's order: channel, height and weight.	
RETURNS	0 on success, or report error in case of failure.	

SEE ALSO	`dpuSetInputTensorInCHWFP32()`
	`dpuSetInputTensorInHWCInt8()`
	`dpuSetInputTensorInHWCFP32()`	

INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuSetInputTensorInCHWInt8()	
SYNOPSIS	int dpuSetInputTensorInCHWInt8 (DPUTask *task, const char*nodeName, int8_t *data, int size)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node's name.
	data	The pointer to the start address of input data.
	size	The size (in Byte) of input data to be set.
DESCRIPTION	Set DPU Task's input Tensor with data from a CPU memory block. Data is in type of INT8 and stored in Caffe Blob's order: channel, height and weight.	
RETURNS	0 on success, or report error in case of failure.	
SEE ALSO	dpuSetInputTensorInCHWFP32() dpuSetInputTensorInHWCInt8() dpuSetInputTensorInHWCFP32()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuSetInputTensorInCHWFP32()

NAME	dpuSetInputTensorInCHWFP32()	
SYNOPSIS	int dpuSetInputTensorInCHWFP32 (DPUTask *task, const char*nodeName, float *data, int size, int idx = 0)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node name.
	data	The pointer to the start address of input data.
	size	The size (in Bytes) of input data to be set.
	idx	The index of a single input tensor for the Node, with default value as 0.
DESCRIPTION	Set DPU Task's input Tensor with data from a CPU memory block. Data is in type of 32-bit-float and stored in Caffe Blob's order: channel, height and weight.	
RETURNS	0 on success, or report error in case of failure.	
SEE ALSO	dpuSetInputTensorInCHWInt8() dpuSetInputTensorInHWCInt8() dpuSetInputTensorInHWCFP32()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuSetInputTensorInCHWFP32()	
SYNOPSIS	int dpuSetInputTensorInCHWFP32 (DPUTask *task, const char*nodeName, float *data, int size)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node's name.
	data	The pointer to the start address of input data.
	size	The size (in Byte) of input data to be set.
DESCRIPTION	Set DPU Task's input Tensor with data from a CPU memory block. Data is in type of 32-bit-float and stored in Caffe Blob's order: channel, height and weight.	
RETURNS	0 on success, or report error in case of failure.	
SEE ALSO	dpuSetInputTensorInCHWInt8() dpuSetInputTensorInHWCInt8() dpuSetInputTensorInHWCFP32()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuSetInputTensorInHWCInt8()

NAME	dpuSetInputTensorInHWCInt8()	
SYNOPSIS	int dpuSetInputTensorInHWCInt8 (DPUTask *task, const char*nodeName, int8_t *data, int size, int idx = 0)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node name.
	data	The pointer to the start address of input data.
	size	The size (in Bytes) of input data to be set.
	idx	The index of a single input tensor for the Node, with default value of 0.
DESCRIPTION	Set DPU Task's input Tensor with data from a CPU memory block. Data is in type of INT8 and stored in DPU Tensor's order: height, weight and channel.	
RETURNS	0 on success, or report error in case of failure.	
SEE ALSO	dpuSetInputTensorInCHWInt8() dpuSetInputTensorInCHWFP32() dpuSetInputTensorInHWCFP32()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuSetInputTensorInHWCInt8()	
SYNOPSIS	int dpuSetInputTensorInHWCInt8 (DPUTask *task, const char*nodeName, int8_t *data, int size)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node's name.
	data	The pointer to the start address of input data.
	size	The size (in Byte) of input data to be set.
DESCRIPTION	Set DPU Task's input Tensor with data from a CPU memory block. Data is in type of INT8 and stored in DPU Tensor's order: height, weight and channel.	
RETURNS	0 on success, or report error in case of failure.	
SEE ALSO	dpuSetInputTensorInCHWInt8() dpuSetInputTensorInCHWFP32() dpuSetInputTensorInHWCFP32()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuSetInputTensorInHWCFP32()

NAME	dpuSetInputTensorInHWCFP32()	
SYNOPSIS	int dpuSetInputTensorInHWCFP32 (DPUTask *task, const char*nodeName, float *data, int size, int idx = 0)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node's name.
	data	The pointer to the start address of input data.
	size	The size (in Byte) of input data to be set.
	idx	The index of a single input tensor for the Node, with default value of 0.
DESCRIPTION	Set DPU Task's input Tensor with data from a CPU memory block. Data is in type of 32-bit-float and stored in DPU Tensor's order: height, weight and channel.	
RETURNS	0 on success, or report error in case of failure.	
SEE ALSO	dpuSetInputTensorInCHWInt8() dpuSetInputTensorInCHWFP32() dpuSetInputTensorInHWCInt8()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuSetInputTensorInHWCFP32()	
SYNOPSIS	int dpuSetInputTensorInHWCFP32 (DPUTask *task, const char*nodeName, float *data, int size)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node's name.
	data	The pointer to the start address of input data.
	size	The size (in Byte) of input data to be set.
DESCRIPTION	Set DPU Task's input Tensor with data from a CPU memory block. Data is in type of 32-bit-float and stored in DPU Tensor's order: height, weight and channel.	
RETURNS	0 on success, or report error in case of failure.	
SEE ALSO	dpuSetInputTensorInCHWInt8() dpuSetInputTensorInCHWFP32() dpuSetInputTensorInHWCInt8()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetOutputTensorInCHWInt8()

NAME	dpuGetOutputTensorInCHWInt8()	
SYNOPSIS	int dpuGetOutputTensorInCHWInt8 (DPUTask *task, const char*nodeName, int8_t *data, int size, int idx = 0)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node name.
	data	The start address of CPU memory block for storing output Tensor's data.
	size	The size (in Bytes) of output data to be stored.
	idx	The index of a single output tensor for the Node, with default value of 0.
DESCRIPTION	Get DPU Task's output Tensor and store its data into a CPU memory block. Data will be stored in type of INT8 and in Caffe Blob's order: channel, height and weight.	
RETURNS	0 on success, or report error in case of failure.	

SEE ALSO	dpuGetOutputTensorInCHWFP32() dpuGetOutputTensorInHWCInt8() dpuGetOutputTensorInHWCFP32()
INCLUDE FILE	n2cube.h
AVAILABILITY	v2.06

NAME	dpuGetOutputTensorInCHWInt8()	
SYNOPSIS	int dpuGetOutputTensorInCHWInt8 (DPUTask *task, const char*nodeName, int8_t *data, int size)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node's name.
	data	The start address of CPU memory block for storing output Tensor's data.
	size	The size (in Byte) of output data to be stored.
DESCRIPTION	Get DPU Task's output Tensor and store its data into a CPU memory block. Data will be stored in type of INT8 and in Caffe Blob's order: channel, height and weight.	
RETURNS	0 on success, or report error in case of failure.	

SEE ALSO	dpuGetOutputTensorInCHWFP32() dpuGetOutputTensorInHWCInt8() dpuGetOutputTensorInHWCFP32()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuGetOutputTensorInCHWFP32()

NAME	dpuGetOutputTensorInCHWFP32()	
SYNOPSIS	int dpuGetOutputTensorInCHWFP32 (DPUTask *task, const char*nodeName, float *data, int size, int idx = 0)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node name.
	data	The start address of CPU memory block for storing output Tensor's data.
	size	The size (in Bytes) of output data to be stored.
	idx	The index of a single output tensor for the Node, with default value of 0.

DESCRIPTION	Get DPU Task's output Tensor and store its data into a CPU memory block. Data will be stored in type of 32-bit-float and in Caffe Blob's order: channel, height and weight.
RETURNS	0 on success, or report error in case of failure.
SEE ALSO	dpuGetOutputTensorInCHWInt8() dpuGetOutputTensorInHWCInt8(), dpuGetOutputTensorInHWCFP32()
INCLUDE FILE	n2cube.h
AVAILABILITY	v2.06

NAME	dpuGetOutputTensorInCHWFP32()	
SYNOPSIS	int dpuGetOutputTensorInCHWFP32 (DPUTask *task, const char*nodeName, float *data, int size)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node's name.
	data	The start address of CPU memory block for storing output Tensor's data.
	size	The size (in Byte) of output data to be stored.

DESCRIPTION	Get DPU Task's output Tensor and store its data into a CPU memory block. Data will be stored in type of 32-bit-float and in Caffe Blob's order: channel, height and weight.
RETURNS	0 on success, or report error in case of failure.
SEE ALSO	dpuGetOutputTensorInCHWInt8() dpuGetOutputTensorInHWCInt8(), dpuGetOutputTensorInHWCFP32()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuGetOutputTensorInHWCInt8()

NAME	dpuGetOutputTensorInHWCInt8()
SYNOPSIS	int dpuGetOutputTensorInHWCInt8 (DPUTask *task, const char*nodeName, int8_t *data, int size, int idx = 0)

ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node's name.
	data	The start address of CPU memory block for storing output Tensor's data.
	size	The size (in Byte) of output data to be stored.
	idx	The index of a single output tensor for the Node, with default value as 0.
DESCRIPTION	Get DPU Task's output Tensor and store its data into a CPU memory block. Data will be stored in type of INT8 and in DPU Tensor's order: height, weight and channel.	
RETURNS	0 on success, or report error in case of failure.	
SEE ALSO	dpuGetOutputTensorInCHWInt8() dpuGetOutputTensorInCHWFP32() dpuGetOutputTensorInHWCFP32()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.06	

NAME	dpuGetOutputTensorInHWCInt8()	
SYNOPSIS	int dpuGetOutputTensorInHWCInt8 (DPUTask *task, const char*nodeName, int8_t *data, int size)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node name.
	data	The start address of CPU memory block for storing output Tensor's data.
	size	The size (in Bytes) of output data to be stored.
DESCRIPTION	Get DPU Task's output Tensor and store its data into a CPU memory block. Data will be stored in type of INT8 and in DPU Tensor's order: height, weight and channel.	
RETURNS	0 on success, or report error in case of failure.	
SEE ALSO	dpuGetOutputTensorInCHWInt8() dpuGetOutputTensorInCHWFP32() dpuGetOutputTensorInHWCFP32()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetOutputTensorInHWCFP32()

NAME	dpuGetOutputTensorInHWCFP32()	
SYNOPSIS	int dpuGetOutputTensorInHWCFP32 (DPUTask *task, const char*nodeName, float *data, int size, int idx = 0)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node name.
	data	The start address of CPU memory block for storing output Tensor's data.
	size	The size (in Bytes) of output data to be stored.
	idx	The index of a single output tensor for the Node, with default value of 0.
DESCRIPTION	Get DPU Task's output Tensor and store its data into a CPU memory block. Data will be stored in type of 32-bit-float and in DPU Tensor's order: height, weight and channel.	
RETURNS	0 on success, or report error in case of failure.	

SEE ALSO	dpuGetOutputTensorInCHWInt8() dpuGetOutputTensorInCHWFP32() dpuGetOutputTensorInHWCInt8()
INCLUDE FILE	n2cube.h
AVAILABILITY	v2.06

NAME	dpuSetInputTensorInHWCFP32()	
SYNOPSIS	int dpuSetInputTensorInHWCFP32 (DPUTask *task, const char*nodeName, float *data, int size, int idx = 0)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node name.
	data	The pointer to the start address of input data.
	size	The size (in Bytes) of input data to be set.
	idx	The index of a single input tensor for the Node, with default value of 0.

DESCRIPTION	Set DPU Task's input Tensor with data from a CPU memory block. Data is in type of 32-bit-float and stored in DPU Tensor's order: height, weight and channel.
RETURNS	0 on success, or report error in case of failure.
SEE ALSO	dpuSetInputTensorInCHWInt8() dpuSetInputTensorInCHWFP32() dpuSetInputTensorInHWCInt8()
INCLUDE FILE	n2cube.h
AVAILABILITY	v2.06

NAME	dpuGetOutputTensorInHWCFP32()	
SYNOPSIS	int dpuGetOutputTensorInHWCFP32 (DPUTask *task, const char*nodeName, float *data, int size)	
ARGUMENTS	task	The pointer to DPU Task
	nodeName	The pointer to DPU Node's name.
	data	The start address of CPU memory block for storing output Tensor's data.
	size	The size (in Byte) of output data to be stored.

DESCRIPTION	Get DPU Task's output Tensor and store its data into a CPU memory block. Data will be stored in type of 32-bit-float and in DPU Tensor's order: height, weight and channel.
RETURNS	0 on success, or report error in case of failure.
SEE ALSO	dpuGetOutputTensorInCHWInt8() dpuGetOutputTensorInCHWFP32() dpuGetOutputTensorInHWCInt8()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuRunSoftmax()

NAME	dpuRunSoftmax ()
SYNOPSIS	int dpuRunSoftmax (int8_t *input, float *output, int numClasses, int batchSize, float scale)

ARGUMENTS	input	The pointer to store softmax input elements in int8_t type.
	output	The pointer to store softmax running results in floating point type. This memory space should be allocated and managed by caller function.
	numClasses	The number of classes that softmax calculation operates on.
	batchSize	Batch size for the softmax calculation. This parameter should be specified with the division of the element number by inputs by numClasses.
	scale	The scale value applied to the input elements before softmax calculation. This parameter typically can be obtained by using DNNDK API dpuGetRensorScale().
DESCRIPTION	Perform softmax calculation for the input elements and save the results to output memory buffer. This API will leverage DPU core for acceleration if harden softmax module is available. Run “dexplorer -w” to view DPU signature information.	
RETURNS	0 for success.	
SEE ALSO	None	
INCLUDE FILE	n2cube.h	
AVAILABILITY	**V2.08**	

dpuSetExceptionMode()

NAME	dpuSetExceptionMode()	
SYNOPSIS	int dpuSetExceptionMode (int mode)	
ARGUMENTS	mode	The exception handling mode for runtime N^{2}Cube to be specified. Available values include: - *N2CUBE_EXCEPTION_MODE_PRINT_AND_EXIT* - *N2CUBE_EXCEPTION_MODE_RET_ERR_CODE*
DESCRIPTION	Set the exception handling mode for DNNDK runtime N^{2}Cube. It will affect all the APIs included in the libn2cube library. If N2CUBE_EXCEPTION_MODE_PRINT_AND_EXIT is specified, the invoked N^{2}Cube APIs will output the error message and terminate the running of DNNDK application when any error occurs. It is the default mode for N^{2}Cube APIs. If N2CUBE_EXCEPTION_MODE_RET_ERR_CODE is specified, the invoked N^{2}Cube APIs only return error code in case of errors. The callers need to take charge of the following exception handling process, such as logging the error message with API dpuGetExceptionMessage(), resource release, etc.	
RETURNS	0 on success, or negative value in case of failure.	
SEE ALSO	dpuGetExceptionMode() dpuGetExceptionMessage()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v2.08	

dpuGetExceptionMode()

NAME	dpuGetExceptionMode()
SYNOPSIS	int dpuGetExceptionMode()
ARGUMENTS	None
DESCRIPTION	Get the exception handling mode for runtime N^{2}Cube.
RETURNS	Current exception handing mode for N^{2}Cube APIs. Available values include: - *N2CUBE_EXCEPTION_MODE_PRINT_AND_EXIT* - *N2CUBE_EXCEPTION_MODE_RET_ERR_CODE*
SEE ALSO	dpuSetExceptionMode() dpuGetExceptionMessage()
INCLUDE FILE	n2cube.h
AVAILABILITY	v2.08

dpuGetExceptionMessage()

NAME	dpuGetExceptionMessage()	
SYNOPSIS	const char *dpuGetExceptionMessage (int error_code)	
ARGUMENTS	error_code	The error code returned by N2Cube APIs.
DESCRIPTION	Get the error message from error code (always negative value) returned by N^{2}Cube APIs.	

RETURNS	A pointer to a const string, indicating the error message for error_code.
SEE ALSO	dpuSetExceptionMode() dpuGetExceptionMode()
INCLUDE FILE	n2cube.h
AVAILABILITY	v2.08

Library libdputils

Overview

Library libdputils.so is the DPU utility library. It wraps up various highly optimized C/C++ APIs to facilitate DL applications development on DPU platform. The exported APIs are briefly summarized in the table below.

NAME	libdputils.so
DESCRIPTION	DPU utility library
ROUTINES	dpuSetInputImage() – set DPU Task's input image with mean values specified from network model dpuSetInputImage2() – set DPU Task's input image without mean values dpuSetInputImageWithScale() – set DPU Task's input image according to the given scale value Note: The three APIs of dpuSetInputImage(), dpuSetInputImage2() and dpuSetInputImageWithScale() automatically rescales the input images to match the input dimension of the used model via invoking OpenCV resize().The image interpolation method adopted is INTER_LINEAR.
INCLUDE FILE	dputils.h

APIs

The prototype and parameter for each API of library libdputils are shown in detail in the following sections.

dpuSetInputImage()

NAME	dpuSetInputImage ()	
SYNOPSIS	int dpuSetInputImage (DPUTask *task, const char *nodeName, const cv::Mat &image, float *mean, int idx = 0)	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node name. Note that the available names of one DPU Kernel or Task output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	image	Input image in OpenCV Mat format. Single channel and 3-channel input image are supported.

	mean	Pointer to mean value array which contains 1 member for single channel input image or 3 members for 3-channel input image. Note: You can get the mean values from the input Caffe prototxt. At present, the format of mean value file is not yet supported.
	idx	The index of a single input tensor for the Node, with default value as 0.
DESCRIPTION	set DPU Task input image	
RETURNS	0 on success, or report error in case of any failure.	
SEE ALSO	dpuSetInputImageWithScale ()	
INCLUDE FILE	dputil.h	
AVAILABILITY	v2.06	

NAME	dpuSetInputImage ()
SYNOPSIS	int dpuSetInputImage (DPUTask *task, const char *nodeName, const cv::Mat &image, float *mean)

ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	image	Input image in OpenCV's Mat format. Single channel and 3-channel input image are supported.
	mean	Pointer to mean value array which contains 1 member for single channel input image or 3 members for 3-channel input image. Note: You can get the mean values from the input Caffe prototxt. At present, the format of mean value file is not yet supported.
DESCRIPTION	set DPU Task's input image	
RETURNS	0 on success, or report error in case of any failure.	
SEE ALSO	dpuSetInputImageWithScale()	
INCLUDE FILE	dputil.h	
AVAILABILITY	v1.07	

dpuSetInputImage2()

NAME	dpuSetInputImage2 ()	
SYNOPSIS	int dpuSetInputImage (DPUTask *task, const char *nodeName, const cv::Mat &image, int idx = 0)	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node name. Note that the available names of one DPU Kernel or Task output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	image	Input image in OpenCV's Mat format. Single channel and 3-channel input image are supported.
	idx	The index of a single input tensor for the Node, with default value as 0.
DESCRIPTION	set DPU Task input image without specify the mean value	

RETURNS	0 on success, or report error in case of any failure.	
SEE ALSO	dpuSetInputImageWithScale ()	
INCLUDE FILE	dputil.h	
AVAILABILITY	v2.06	

NAME	dpuSetInputImage2 ()	
SYNOPSIS	int dpuSetInputImage (DPUTask *task, const char *nodeName, const cv::Mat &image)	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	image	Input image in OpenCV's Mat format. Single channel and 3-channel input image are supported.

DESCRIPTION	set DPU Task's input image without specify the mean value
RETURNS	0 on success, or report error in case of any failure.
SEE ALSO	dpuSetInputImageWithScale ()
INCLUDE FILE	dputil.h
AVAILABILITY	v1.10

dpuSetInputImageWithScale()

NAME	dpuSetInputImageWithScale ()	
SYNOPSIS	int dpuSetInputImageWithScale (DPUTask *task, const char *nodeName, const cv::Mat &image, float *mean, float scale)	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node name. Note that the available names of one DPU Kernel's or Task output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.
	image	Input image in OpenCV Mat format. Single channel and 3-channel input image are supported.

	mean	Pointer to mean array which containing 3 elements **Note**: you can get the mean values from the input Caffe prototxt and mean file is not supported so far.
	scale	Scale value of input image
DESCRIPTION	set DPU Task input image with the mean value and scale specified from network model	
RETURNS	0 on success, or report error in case of any failure.	
SEE ALSO	dpuSetInputImage ()	
INCLUDE FILE	dputil.h	
AVAILABILITY	v1.07	

NAME	dpuSetInputImageWithScale ()	
SYNOPSIS	int dpuSetInputImageWithScale (DPUTask *task, const char *nodeName, const cv::Mat &image, float *mean, float scale)	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Note that the available names of one DPU Kernel's or Task's output Node are listed out after a neural network is compiled by DNNC. If invalid Node name specified, failure message is reported.

	image	Input image in OpenCV's Mat format. Single channel and 3-channel input image are supported.
	mean	Pointer to mean array which containing 3 elements Note: you can get the mean values from the input Caffe prototxt and mean file is not supported so far.
	scale	Scale value of input image
DESCRIPTION	set DPU Task's input image with the mean value and scale specified from network model	
RETURNS	0 on success, or report error in case of any failure.	
SEE ALSO	dpuSetInputImage ()	
INCLUDE FILE	dputil.h	
AVAILABILITY	v1.07	

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..