

DNNDK User Guide

UG1327 (v 2.08 Beta) December 12, 2018



Revision History

The following table shows the revision history for this document.

Section	Revision Summary
11/15/2018	
General updates	Initial Xilinx release.
12/12/2018	
General updates	Updated for v2.08 Beta release.

Revision History	2
Table of Contents	3
Chapter 1: Quick Start.....	5
Get DNNDK	5
Host setup.....	7
Evaluation board setup.....	9
Running DNNDK examples	26
Support	32
Chapter 2: Copyright and Version.....	33
Copyright	33
Version	33
Chapter 3: Upgrade and Porting.....	38
Since v2.08	38
Since v2.07	39
Since v2.06	39
Since v1.10	41
Upgrading from previous versions.....	42
Chapter 4: DNNDK	44
Overview.....	44
Deep learning processor unit (DPU).....	45
DNNDK framework.....	46
Chapter 5: Network deployment overview	48
DECENT overview.....	48
Network compression	49
Network compilation	50
Programming with DNNDK.....	51
Hybrid compilation.....	54
Running	54

Chapter 5: Network compression.....	55
DECENT overview.....	55
DECETN usage.....	55
Working flow	56
Chapter 6: Network compilation.....	58
DNNC overview	58
DNNC usage	58
Compiling ResNet50.....	60
Chapter 7: Programming with DNNDK	63
Programming model.....	63
Programming interface.....	65
Chapter 8: Hybrid compilation	66
Chapter 9: Running	67
Chapter 10: Utilities.....	68
DExplorer.....	68
DSight.....	71
Chapter 11: DNNDK Programming APIs.....	73
Library libn2cube.....	73
Library libdputils.....	138
Legal Notices	144
Please Read: Important Legal Notices	144

Get DNNDK

The DeePhi DNNDK package can be freely downloaded after registration from the DeePhi website <http://www.deephi.com/technology/dnndk>.

Using a DNNDK-supported evaluation board is recommended to allow the user to become familiar with the DNNDK toolchain. Please visit <http://www.deephi.com/technology/boards> for more details about the DNNDK-supported evaluation boards.



The package name for the DNNDK v2.08 release is "deephi_dnndk_v2.08_beta.tar.gz". The directory structure for the DNNDK release package is shown below. In the subsequent part of this document, "\$deephi_dnndk_package" is used to represent the name of "deephi_dnndk_v2.08_beta" for convenience. The evaluation boards supported for this release are the DeePhi DP-8020 and DP-N1 AI Box, the Xilinx ZCU102 and ZCU104, and the Avnet Ultra96.

The "docs" folder contains this user guide. The "common" folder contains image files used by various DNNDK example applications.

The "host_x86" folder contains files which 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 will correspond to the DNNDK-supported boards: "DP-8020", "DP-N1", "Ultra96", "ZCU102", or "ZCU104". Utility tools, DPU drivers, DPU runtime and development libraries are device-specific and will be different for the various boards.

NOTE: Folder "DP-8020" is for the DeePhi DP-8020; "DP-N1" is for the DeePhi DP-N1 AI Box; "Ultra96" is for the Avnet Ultra96; "ZCU102" is for the Xilinx ZCU102; and "ZCU104" is for the Xilinx ZCU104.

```
$deephi_dnndk_package
├── common
├── docs
├── COPYRIGHT
├── host_x86
│   ├── install.sh
│   ├── models
│   └── pkgs
│       ├── ubuntu14.04
│       └── ubuntu16.04
└── board_name
    ├── install.sh
    ├── pkgs
    │   ├── bin
    │   ├── driver
    │   ├── include
    │   └── lib
    ├── samples
    ├── common
    ├── inception_v1
    ├── inception_v1_mt
    ├── resnet50
    ├── resnet50_mt
    ├── mobilenet
    ├── mobilenet_mt
    ├── face_detection
    └── pose_detection
```

```

└── video_analysis
└── adas_detection
└── segmentation

```

Host setup

The “host_x86” folder contains the DECENT (Deep Compression Tool) and DNNC (Deep Neural Network Compiler) programs which allow a neural network to be optimized and accelerated on the DPU inference engine.

After downloading and unpacking the DNNDK package, copy the “host_x86” folder to a 64-bit Ubuntu 14.04 LTS or Ubuntu 16.04 LTS machine with a Nvidia GPU card. Execute the command “./install.sh board_name” under the “host_x86” folder to install the software on the host. board_name must be replaced with DP-8020 or DP-N1 or Ultra96 or ZCU104 or ZCU102 according to the evaluation board to be used.

The DNNDK host side tools require the third-party packages listed in Table 1 to run.

DNNDK v2.08 supports 64-bit Ubuntu 14.04 LTS and Ubuntu 16.04 LTS, with Nvidia cuDNN 7.0.5 and CUDA 8.0, or 9.0, or 9.1.

Table 1: Supported 64-bit host OS platforms and required packages

ID	OS Platform	Required Packages
1	Ubuntu 14.04 LTS	CUDA 8.0 (GA2), cuDNN 7.0.5
2	Ubuntu 16.04 LTS	CUDA 8.0 (GA2), cuDNN 7.0.5
		CUDA 9.0, cuDNN 7.0.5
		CUDA 9.1, cuDNN 7.0.5

Install required 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 libprotobuf-dev protobuf-compiler libleveldb-dev liblmdb-dev libhdf5-dev
  libsnappy-dev libboost-all-dev libyaml-cpp-dev libssl-dev

```

Install CUDA

Get the CUDA package associated with the Ubuntu version on the host platform from the Nvidia website <https://developer.nvidia.com/cuda-toolkit> and install it under the “/usr/local/” directory.

DNNDK User Guide www.xilinx.com

UG1327 (v 2.08 Beta) December 12, 2018

Install cuDNN

Get the appropriate version of cuDNN to match CUDA from <https://developer.nvidia.com/cudnn>. Decompress and copy it into the "/usr/local/" directory using the commands shown below.

```
$ sudo tar -xvzf cudnn-9.1-linux-x64-v7.tgz
$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include
$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
$ sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
```


Evaluation board setup

Introduction

In the following sections, the evaluation boards supported by DNNDK are described. The DeePhi DP-8020, DP-N1 and Ultra96 are intended for evaluating low-power, low-throughput deep learning applications and 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 <http://www.deephi.com/technology/boards>. Before trying DNNDK v2.08 on the evaluation boards, the users need to download the updated version image file; otherwise, DNNDK package can't work on the previous version image file. The throughput performance in FPS for each DNNDK sample is listed for all these evaluation boards.

Note1: Do download the updated version image file from <http://www.deephi.com/technology/boards> before installing DNNDK v2.08 into your evaluation board.

Note2: The performance numbers in FPS for all DNNDK examples on each evaluation board below are end-to-end throughput results for the neural network algorithms used. They are measured with the time span from the image fed into DPU to the completed running of algorithms. And the time spent on image reading and displaying isn't counted.

Note3: The FPS numbers for ResNet-50, Inception-v1 and MobileNet are measured with multithreaded version examples.

DP-8020

The DeePhi DP-8020 evaluation board uses the Xilinx ZU2 Zynq UltraScale+ device. It is intended to demonstrate the capability of the Xilinx ZU2 device to meet the intensive computation workload requirements of edge AI applications, such as surveillance cameras, ADAS/AD, and robotics. The hardware user guide for DP-8020 is available for download on <http://www.deephi.com/technology/boards>.

One B1152F DPU core is implemented in the programmable logic and delivers 495 GOPs INT8 peak performance for deep learning inference acceleration with low latency and low energy consumption. The main connectivity interfaces for the DP-8020 are shown in Figure 1.

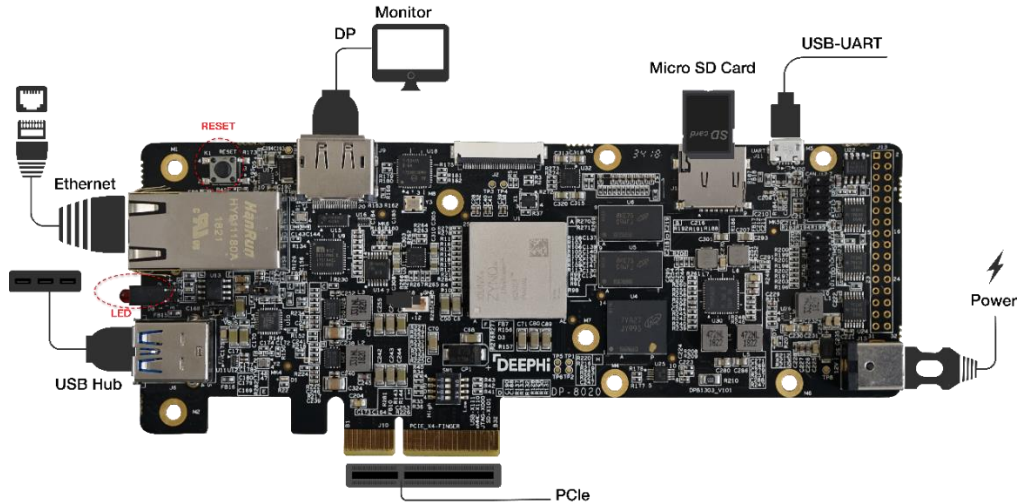


Figure 1: DeePhi DP-8020 evaluation board and peripheral connections

The DExplorer utility tool can be used to display DPU specification information, which is named as DPU signature, covering target version, working frequency, DPU core numbers, etc. In the screenshot below, one B1152F DPU core running at 430 MHz is implemented on the Xilinx ZU2 device.

```
root@dp-8020:~# dexplorer -w
[DPU IP Spec]
IP Timestamp   : 2018-11-15 10:00:00
DPU Core Count : 1
IRQ Base 0     : 121
IRQ Base 1     : 136

[DPU Core List]
DPU Core       : #0
DPU Enabled    : Yes
DPU Arch       : B1152F
DPU Target     : v1.3.7
DPU Frequency  : 430 MHz
DPU IRQ        : 138
DPU Features   : Avg-Pooling, Leaky ReLU, Depthwise Conv
```

Figure 2: DeePhi DP-8020 DPU signature viewed with DExplorer

Please refer to Table 2 for the throughput performance (in frames/sec or fps) for various DNNDK samples on DP-8020.

Table 2: DP-8020 performance

Neural Network	MAC (GOPS)	fps
ResNet-50	7.7	35
Inception-v1	3.2	79
MobileNet	0.56	162

Face detection	1.1	181
Video analysis	5.5	37
Pose detection	5.0	21
ADAS detection	5.5	31
Semantic segmentation	8.8	32

DP-N1

The DP-N1 AI Box designed by DeePhi is based on XILINX XCZU2CG-L1SFVA625I device. With high speed 2GB DDR4 chips, 8GB eMMC flash, mini Display Port, Micro SD, Gigabit Ethernet, and USB 3.0, the DP-N1 enables rapid prototyping of deep learning applications with DNNDK. The hardware user guide for DP-N1 is available for download on <http://www.deephi.com/technology/boards>.

One DPU core B1152F is implemented in the programmable logic and delivers 426 GOPs INT8 peak performance for deep learning inference acceleration. The main connectivity interfaces for the DP-8020 are shown in Figure 3.

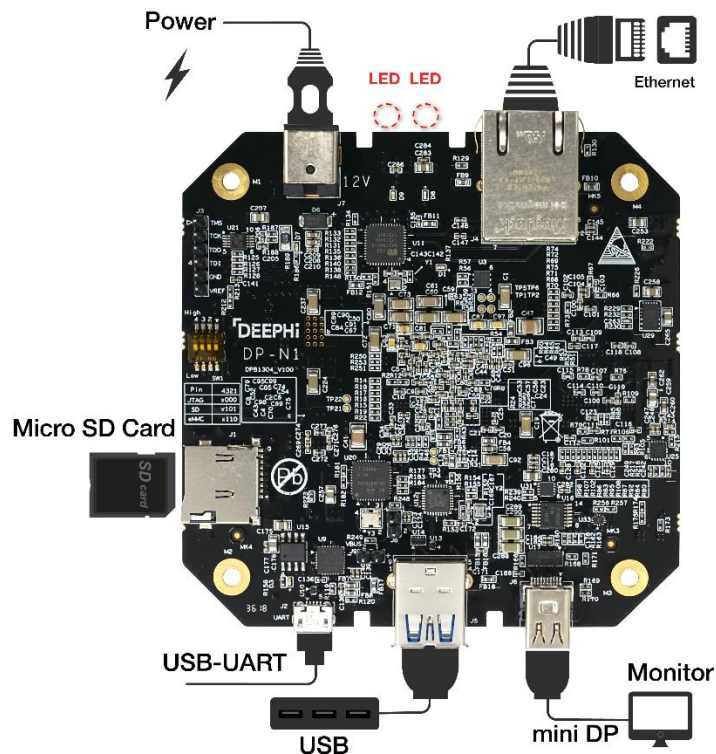


Figure 3: DeePhi DP-N1 evaluation board and peripheral connections

The DPU signature information is shown in the screenshot below. One B1152F DPU core running at 370 MHz is implemented on the Xilinx ZU2 device.

```
root@dp-n1:~# dexplorer -w
[DPU IP Spec]
IP Timestamp   : 2018-11-15 16:00:00
DPU Core Count : 1
IRQ Base 0     : 121
IRQ Base 1     : 136

[DPU Core List]
DPU Core       : #0
DPU Enabled    : Yes
DPU Arch       : B1152F
DPU Target     : v1.3.7
DPU Frequency  : 370 MHz
DPU IRQ        : 138
DPU Features   : Avg-Pooling, Leaky ReLU, Depthwise Conv
```

Figure 4: DeePhi DP-N1 DPU signature viewed with DExplorer

Please refer to Table 3 for the throughput performance (in frames/sec or fps) for various DNNDK samples on DP-N1.

Table 3: DP-N1 performance

Neural Network	MAC (GOPS)	fps
ResNet-50	7.7	32
Inception-v1	3.2	72
MobileNet	0.56	145
Face detection	1.1	168
Video analysis	5.5	24
Pose detection	5.0	19
ADAS detection	5.5	25
Semantic segmentation	8.8	26

Ultra96

Ultra96 is an ARM-based, Xilinx Zynq UltraScale+™ MPSoC development board based on the Linaro 96Boards specification. The 96Boards' 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 user guide for Ultra96 is available for download on <http://zedboard.org/product/ultra96>.

One B1152F DPU core is implemented in the programmable logic of Ultra96 and delivers 383 GOPs INT8 peak performance for deep learning inference acceleration. The main connectivity interfaces for Ultra96 are shown in Figure 5.

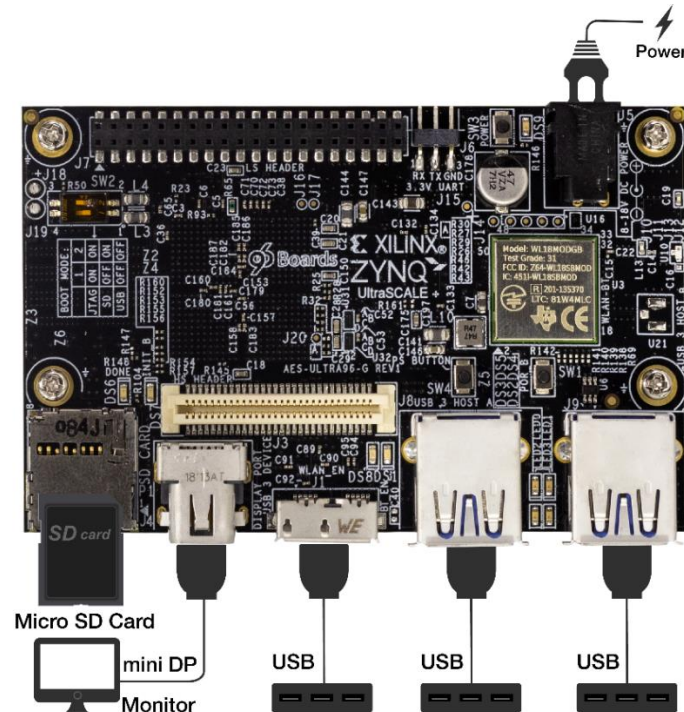


Figure 5: Ultra96 evaluation board and peripheral connections

The DPU signature information is shown in the screenshot below. One B1152F DPU core running at 333 MHz is implemented on the Xilinx ZU3 device.

```
root@ultra96:~# dexplorer -w
[DPU IP Spec]
IP Timestamp   : 2018-12-07 10:45:00
DPU Core Count : 1
IRQ Base 0     : 121
IRQ Base 1     : 136

[DPU Core List]
DPU Core       : #0
DPU Enabled    : Yes
DPU Arch       : B1152F
DPU Target     : v1.3.7
DPU Frequency  : 333 MHz
DPU IRQ        : 138
DPU Features   : Avg-Pooling, Leaky ReLU, Depthwise Conv
```

Figure 6: Ultra96 DPU signature viewed with DExplorer

Please refer to Table 4 for the throughput performance (in frames/sec or fps) for various DNNDK samples on Ultra96.

Table 4: Ultra96 performance

Neural Network	MAC (GOPS)	fps
ResNet-50	7.7	25
Inception-v1	3.2	58
MobileNet	0.56	116
Face detection	1.1	133
Video analysis	5.5	33
Pose detection	5.0	20
ADAS detection	5.5	30
Semantic segmentation	8.8	23

Refer to Figure 7 and Figure 8 to setup WIFI network connection for the Ultra96 board.

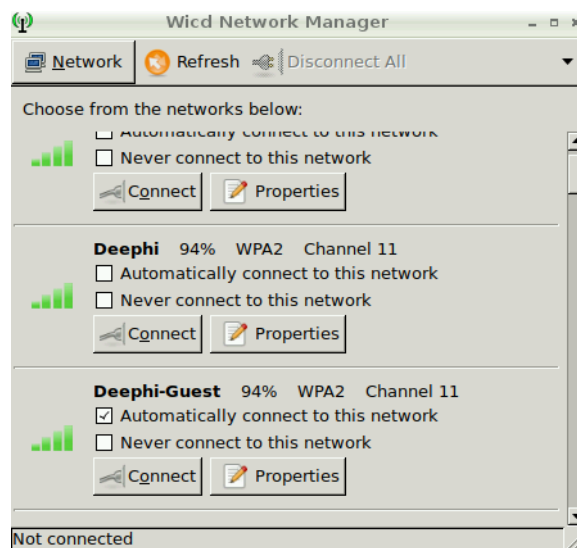


Figure 7: Ultra96 WIFI connection

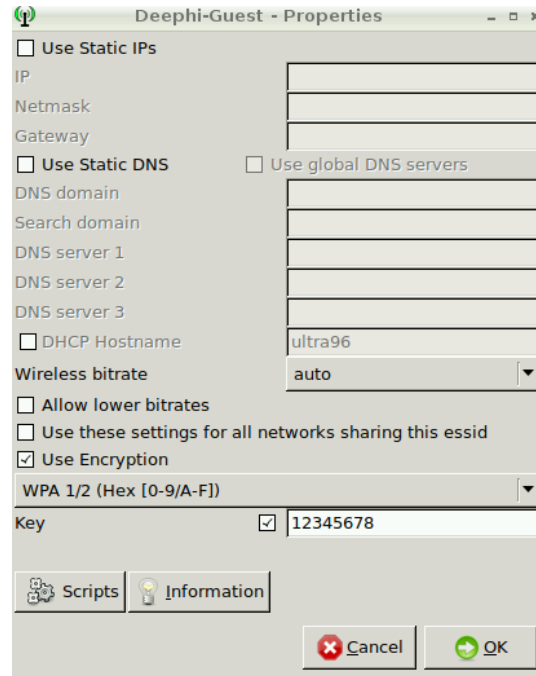


Figure 8: Ultra96 WIFI configuration

ZCU102

The Xilinx ZCU102 evaluation board uses the mid-range ZU9 UltraScale+ device to enable users to jumpstart their machine learning applications. For more information on the ZCU102, please refer to the Xilinx site <https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html>.

Triple B4096F DPU cores are implemented in program logic and delivers 4 TOPs INT8 peak performance for deep learning inference acceleration. The main connectivity interfaces for the ZCU102 are shown in Figure 9.

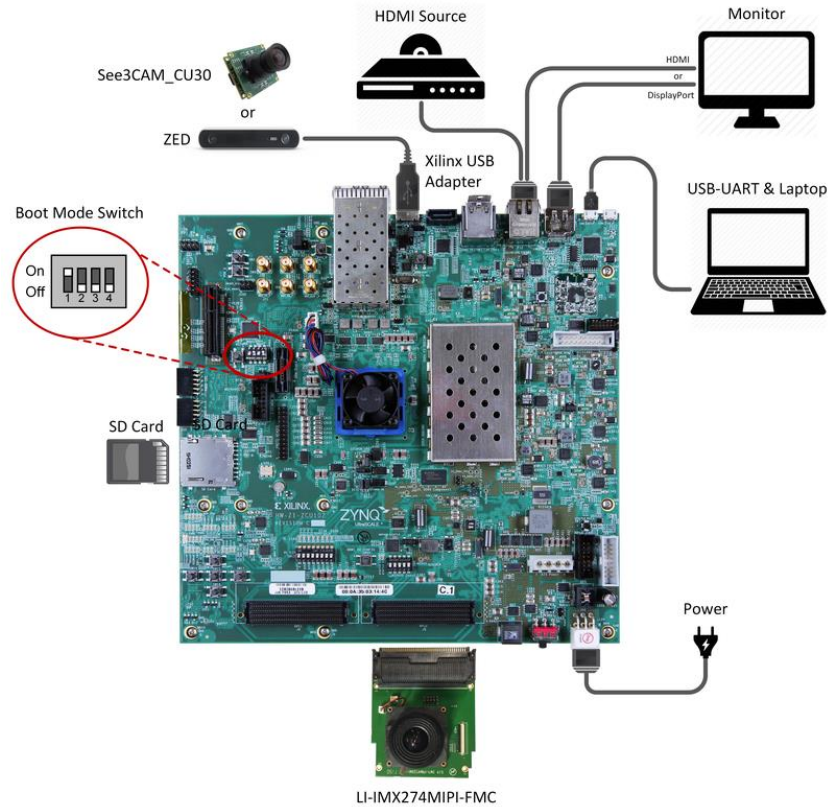


Figure 9: Xilinx ZCU102 evaluation board and peripheral connections

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


```
[DPU IP Spec]
IP Timestamp      : 2018-11-13 15:15:00
DPU Core Count    : 3
IRQ Base 0        : 121
IRQ Base 1        : 136

[DPU Core List]
DPU Core          : #0
DPU Enabled       : Yes
DPU Arch          : B4096F
DPU Target        : v1.3.0
DPU Frequency     : 333 MHz
DPU IRQ           : 138
DPU Features      : Leaky ReLU
DPU Core          : #1
DPU Enabled       : Yes
DPU Arch          : B4096F
DPU Target        : v1.3.0
DPU Frequency     : 333 MHz
DPU IRQ           : 139
DPU Features      : Leaky ReLU
DPU Core          : #2
DPU Enabled       : Yes
DPU Arch          : B4096F
DPU Target        : v1.3.0
DPU Frequency     : 333 MHz
DPU IRQ           : 140
DPU Features      : Leaky ReLU

[DPU Extension List]
Extension Softmax
Enabled           : Yes
IRQ               : 142
```

Figure 10: Xilinx ZCU102 DPU signature viewed with DExplorer

Refer to Table 5 for the throughput performance (in frames/sec or fps) for various DNNDK samples on ZCU102.

Table 5: ZCU102 performance

Neural Network	MAC (GOPS)	fps
ResNet-50	7.7	175
Inception-v1	3.2	428
Face detection	1.1	686
Video analysis	5.5	56

Pose detection	5.0	44
ADAS detection	5.5	62
Semantic segmentation	8.8	64

ZCU104

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

Dual B4096F DPU cores are implemented in program logic and delivers 2.4 TOPs INT8 peak performance for deep learning inference acceleration. The main connectivity interfaces for ZCU104 are shown in Figure 11.

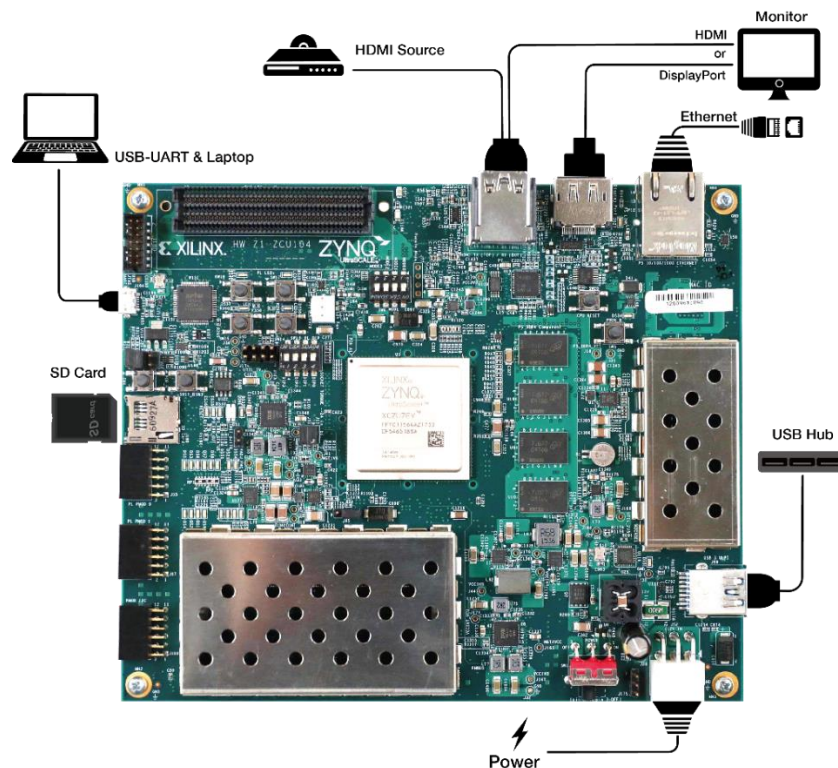


Figure 11: Xilinx ZCU104 evaluation board and peripheral connections

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

```

root@zcu104:~# dexplorer -w
[DPU IP Spec]
IP Timestamp      : 2018-11-07 10:30:00
DPU Core Count    : 2
IRQ Base 0        : 121
IRQ Base 1        : 136

[DPU Core List]
DPU Core          : #0
DPU Enabled       : Yes
DPU Arch          : B4096F
DPU Target        : v1.3.0
DPU Frequency     : 333 MHz
DPU IRQ           : 122
DPU Features      : Leaky ReLU
DPU Core          : #1
DPU Enabled       : Yes
DPU Arch          : B4096F
DPU Target        : v1.3.0
DPU Frequency     : 333 MHz
DPU IRQ           : 123
DPU Features      : Leaky ReLU

```

Figure 12: Xilinx ZCU104 DPU signature viewed with DExplorer

Refer to Table 6 for the throughput performance (in frames/sec or fps) for various DNNDK samples on ZCU104.

Table 6: ZCU104 performance

Neural Network	MAC (GOPS)	fps
ResNet-50	7.7	127
Inception-v1	3.2	299
Face detection	1.1	566
Video analysis	5.5	55
Pose detection	5.0	66
ADAS detection	5.5	53
Semantic segmentation	8.8	63

Note: For the ZCU104 board, the examples of multithreaded ResNet-50, ADAS detection and segmentation are not included in the DNNDK v2.08 package because they demand more power supply, however which can't be

offered by ZCU104 default setting. There is a separate patch available to resolve this power issue. Contact DNNDK support team if these 3 samples are necessary for your project evaluation.

In the following sections, DP-8020 is used as an example to show the steps to setup the DNNDK running environment on evaluation boards.

Flash OS image to 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. In the following let's take Windows as the example to go through the steps.

1. Download Etcher from <https://etcher.io/> and save the file as shown in Figure 13.

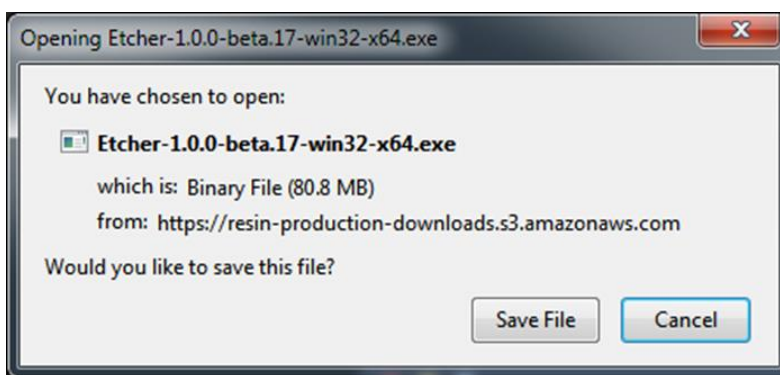


Figure 13: Saving the Etcher file

2. Install Etcher as shown in Figure 14.

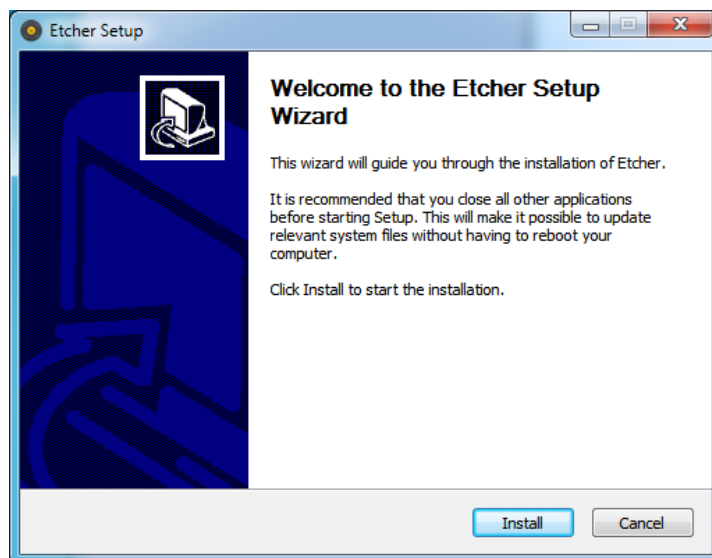


Figure 14: Install Etcher

3. Etcher can operate on any external storage devices such as USB flash drives and backup hard disks. The first step will be to insert the SD card into the slot on your computer or into the reader.

Run the Etcher program by double clicking on the Etcher icon shown in Figure 15, or select it from the Start menu. This will launch Etcher as shown in Figure 16.

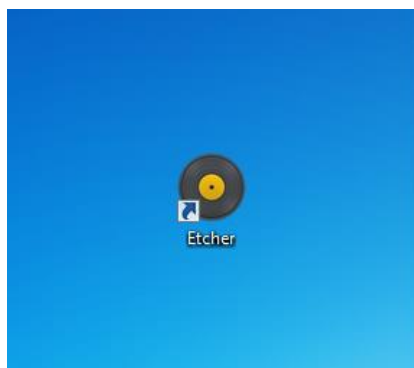


Figure 15: Etcher icon

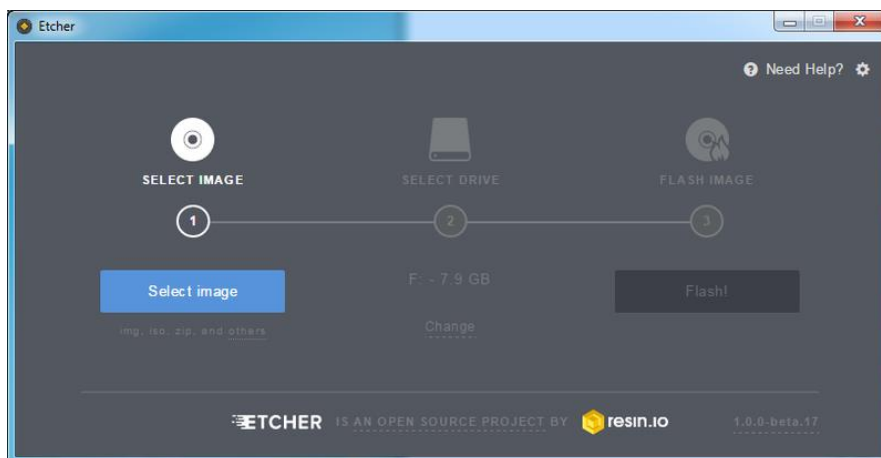


Figure 16: The Etcher GUI

4. Select the image file by clicking Select Image. ".zip" or ".gz" format compressed file may be selected.
5. Etcher will try to detect the SD drive. Verify the drive designation and check the image size and ensure that it is the right. Then, click **Flash** as shown in Figure 17.

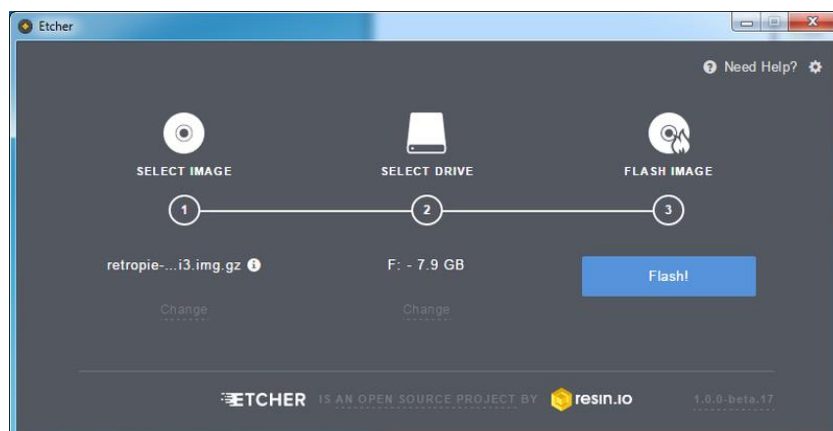


Figure 17: Flash SD card

Booting the evaluation board

For DP-8020, please follow the steps below to boot the evaluation board.

1. Connect the power supply (12V ~ 2A)
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

- The system needs to perform some configurations for its first boot. Then reboot the board for these configurations to take effect.

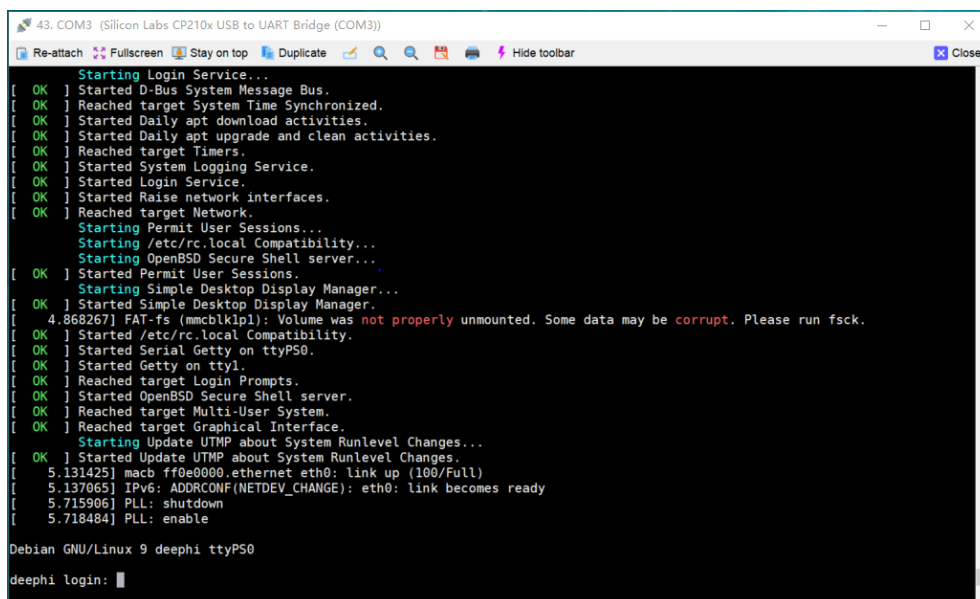
Accessing the evaluation board

There are three ways to access the DP-8020 evaluation board: via UART, ethernet, or standalone.

UART

Aside from monitoring the boot process and viewing kernel messages for debugging, users 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 Figure 18. Log into system with username "root" and password "root".

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



```

43. COM3 (Silicon Labs CP210x USB to UART Bridge (COM3))
[ OK ] Starting Login Service...
[ OK ] Started D-Bus System Message Bus.
[ OK ] Reached target System Time Synchronized.
[ OK ] Started Daily apt download activities.
[ OK ] Started Daily apt upgrade and clean activities.
[ OK ] Reached target Timers.
[ OK ] Started System Logging Service.
[ OK ] Started Login Service.
[ OK ] Started Raise network interfaces.
[ OK ] Reached target Network.
[ OK ] Starting Permit User Sessions.
[ OK ] Starting /etc/rc.local Compatibility.
[ OK ] Starting OpenBSD Secure Shell server.
[ OK ] Started Permit User Sessions.
[ OK ] Starting Simple Desktop Display Manager.
[ OK ] Started Simple Desktop Display Manager.
[ 4.868267] FAT-fs (mmcblk1p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
[ OK ] Started /etc/rc.local Compatibility.
[ OK ] Started Serial Getty on ttyPS0.
[ OK ] Started Getty on tty1.
[ OK ] Reached target Login Prompts.
[ OK ] Started OpenBSD Secure Shell server.
[ OK ] Reached target Multi-User System.
[ OK ] Reached target Graphical Interface.
[ OK ] Starting Update UTMP about System Runlevel Changes...
[ OK ] Started Update UTMP about System Runlevel Changes.
[ 5.131425] macb ff0e0000.ethernet eth0: link up (100/Full)
[ 5.137065] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 5.715906] PLL: shutdown
[ 5.718484] PLL: enable
Debian GNU/Linux 9 deephi ttyPS0
deephi login:

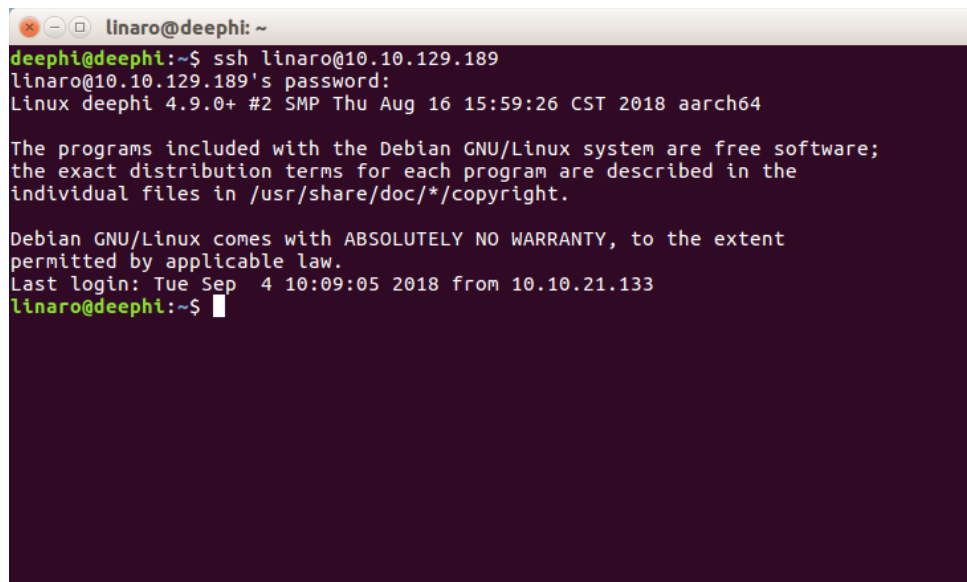
```

Figure 18: Screenshot of boot process

Note: On a Linux system, Minicom may be used 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.

Ethernet

The DP-8020 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.



```
linaro@deephi: ~
deephi@deephi:~$ ssh linaro@10.10.129.189
linaro@10.10.129.189's password:
Linux deephi 4.9.0+ #2 SMP Thu Aug 16 15:59:26 CST 2018 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: Tue Sep  4 10:09:05 2018 from 10.10.21.133
linaro@deephi:~$
```

Figure 19: Logging in to the evaluation board using SSH

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

Standalone

The DP-8020 board allows a keyboard, mouse and monitor to be connected. After a successful boot, the Linux windows desktop will be displayed as shown in Figure 20. Then the board can be access as a standalone embedded system.

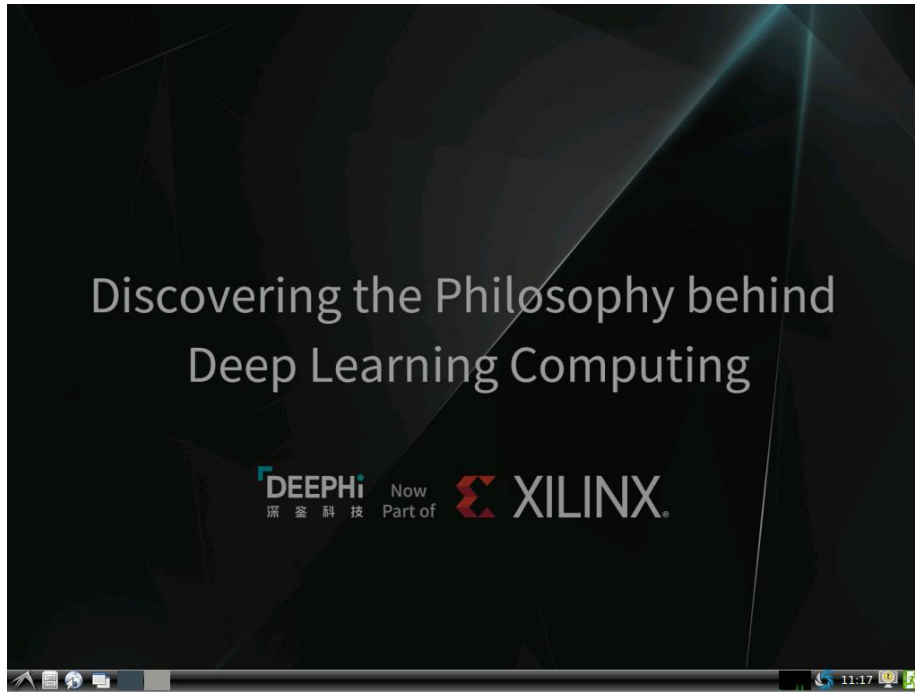


Figure 20: Screenshot of standalone Linux desktop

Copy 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. To minimize the size of DNNDK package, the directory “\$deephi_dnndk_package/board_name/samples/common” for all DNNDK evaluation boards is symbol link to the unique directory “\$deephi_dnndk_package/common”, which contains the dependent image files for all DNNDK examples. Extraction of DNNDK package on Windows system will break these symbol links and bring issues while running DNNDK examples on the evaluation boards.

The steps below illustrate how to setup DNNDK running environment for DP-8020 provided that DNNDK package is stored on Windows system.

1. Download and install MobaXterm (<https://mobaxterm.mobatek.net/>) on Windows system
2. Launch MobaXterm and click “Start local terminal” to open a terminal, where the filesystem of Windows can be accessed.
3. Supposed that DNNDK package is located under the root directory of disk “D”, run the following steps to extract and copy the package for the DP-8020 board with IP address 192.168.0.10.

```
cd d:
tar -xzf deephi_dnndk_v2.08_beta.tar.gz
cd deephi_dnndk_v2.08_beta/
scp -r ./DP-8020 root@192.168.0.10:~/
```

4. On the DP-8020 board, change to the “~/DP-8020/” directory and run “install.sh”. The following messages will be shown if the installation completes successfully. Then reboot the board for the installation to take effect.

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

Note1: DO reboot the board after installation

Note2: Installing DNNDK v2.08 will replace previous releases automatically. There is no need to manually uninstall previous versions. And DO reboot the board after installation

Running DNNDK examples

This section will illustrate how to run each example included in the DNNDK package, using the DeePhi DP-8020 board as a reference as well.

All examples may be compiled and launched after successful installation of the DNNDK package. They are stored under the “\$deephi_dnndk_package/DP-8020/samples” folder. In the following sections, we will use the string “\$dnndk_sample_base” to represent the folder name of “\$deephi_dnndk_package/DP-8020”.

Make sure to enable X11 forwarding with the following command (supposed that host machine IP address is 192.168.0.10) when logging in to the board using an SSH terminal since all the examples require a Linux windows system to work properly.

```
export DISPLAY=192.168.0.10:0.0
```

For each DNNDK example, after entering into its directory, run the “make” command first to generate the hybrid DPU executable file, then launch it just like a normal Linux application.

Note1: The DNNDK examples won’t work through a UART connection due to the lack of Linux windows.

Note2: The monitor flickers while running for some DNNDK examples because DPU demands heavy memory bandwidth.

ResNet-50

“\$DNNDK_SAMPLE_BASE/samples/resnet50” contains an example of image classification using the ResNet-50 network. It reads the images under the “\$dnndk_sample_base/samples/common” directory and outputs the classification result for each input image.

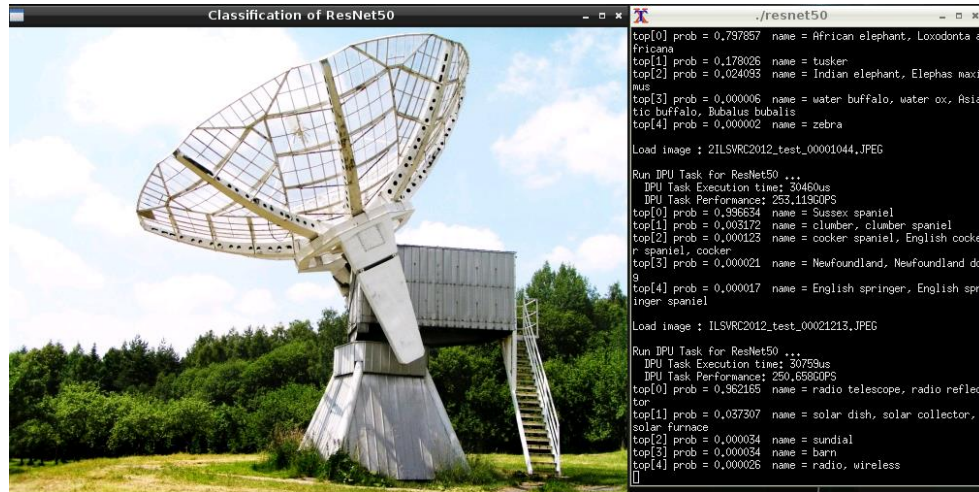


Figure 21: Screenshot of ResNet-50 example

After running “make” and launching it with the command “./resnet50”, the result is shown in Figure 16, including the top-5 labels and corresponding probabilities.

Multithreaded ResNet-50

“\$DNNDK_SAMPLE_BASE/samples/resnet50_mt” contains a ResNet-50 image classification example, programmed in multithreaded mode to achieve higher throughput. It demonstrates how to use DNNDK APIs to develop multithreaded DPU applications for better performance.

With the command “./resnet50 4”, the throughput (in fps) will be reported after processing 1000 input images. The option “4” means four threads will be spawned and share the usage of DPU core. This allows the reduction of DPU idle time, thereby increasing throughput.

Note: The thread number for best throughput of multithread ResNet-50 example varies among evaluation boards since DPU computation power and core number are differently equipped. Use “dexplorer -w” to view DPU signature information for each evaluation board.

Inception-v1

The “\$dnndk_sample_base/samples/inception_v1” directory contains an example for image classification using the Inception-v1 network. After running make, launch it with the command “./inception_v1”. A screenshot of a sample run is shown in Figure 22.

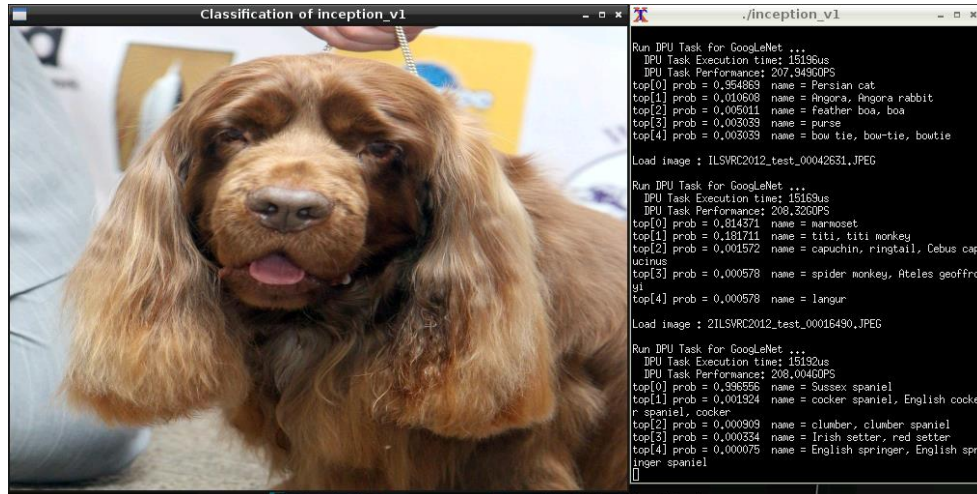


Figure 22: Screenshot of Inception-V1 example

Multithreaded Inception-v1

"\$dnndk_sample_base/samples/inception_v1_mt" contains a multithreaded image classification example of the Inception-v1 network. With the command `"/inception_v1_mt 4"`, it will run with 4 threads. The throughput (in fps) will be reported after it completes.

Note: The thread number for best throughput of multithread Inception-v1 example varies among evaluation boards since the DPU computation power and core number are differently equipped. Use `"dexplorer -w"` to view DPU signature information for each evaluation board.

MobileNet

The "\$dnndk_sample_base/samples/mobilenet" directory contains an example for image classification using the MobileNet network. After running make, launch it with the command `"/mobilenet"`. A screenshot of a sample run is shown in Figure 23.

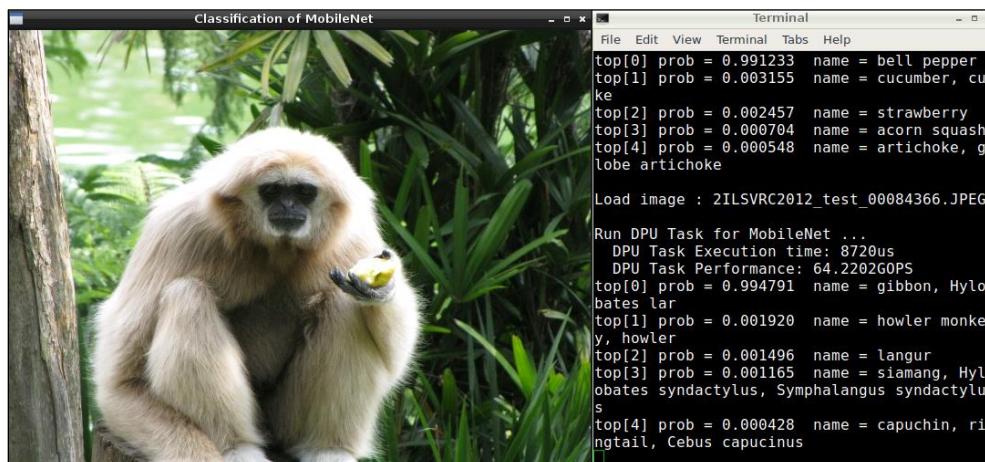


Figure 23: Screenshot of MobieNet example

Note: The MobileNet example is not included in the DNNDK v2.08 release for the ZCU102 and ZCU104 evaluation boards. Contact DNNDK support team if it is necessary for your project evaluation.

Multithreaded MoblieNet

"\$dnndk_sample_base/samples/mobilenet_mt" contains a multithreaded image classification example of the MobileNet network. With the command `./mobilenet_mt 4`, it will run with 4 threads. The throughput (in fps) will be reported after it completes.

Note1: The multithreaded MobileNet example is not included in the DNNDK v2.08 package for the ZCU102 and ZCU104 evaluation boards. Contact DNNDK support team if it is necessary for your project evaluation.

Note2: The thread number for best throughput of multithread MobileNet example varies among evaluation boards since the DPU computation power and core number are differently equipped. Use `"dexplorer -w"` to view DPU signature information for each evaluation board.

Face detection

A face detection example is located under the "\$dnndk_sample_base/samples/face_detection" directory. It reads image frames from a USB camera and annotates the detected faces in real-time. Run make and launch it with the command `./face_detection`. A screenshot of the result is shown in Figure 24.

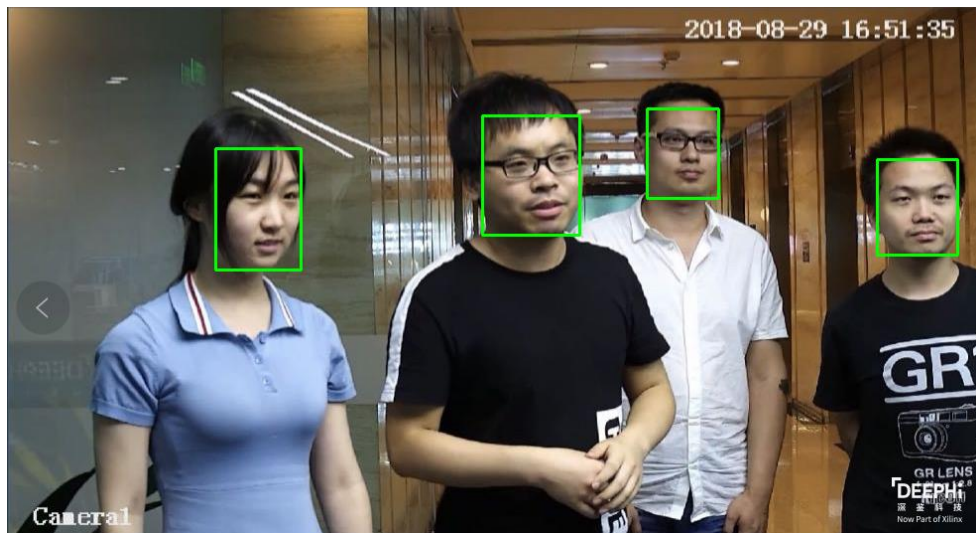


Figure 24: Screenshot of face detection example

Note: For some USB cameras, face detection example may hang after stopping or killing the process. When this occurs, unplug the USB camera and plug in it again.

Pose detection

A pose detection example is located under the "\$dnndk_sample_base/samples/pose_detection" directory. It reads image frames from a video file and annotates the detected human body in real-time.

Run make and launch it with the command `./pose_detection video/pose.mp4` (where video/pose.mp4 is the input video file). A screenshot is shown in Figure 25.

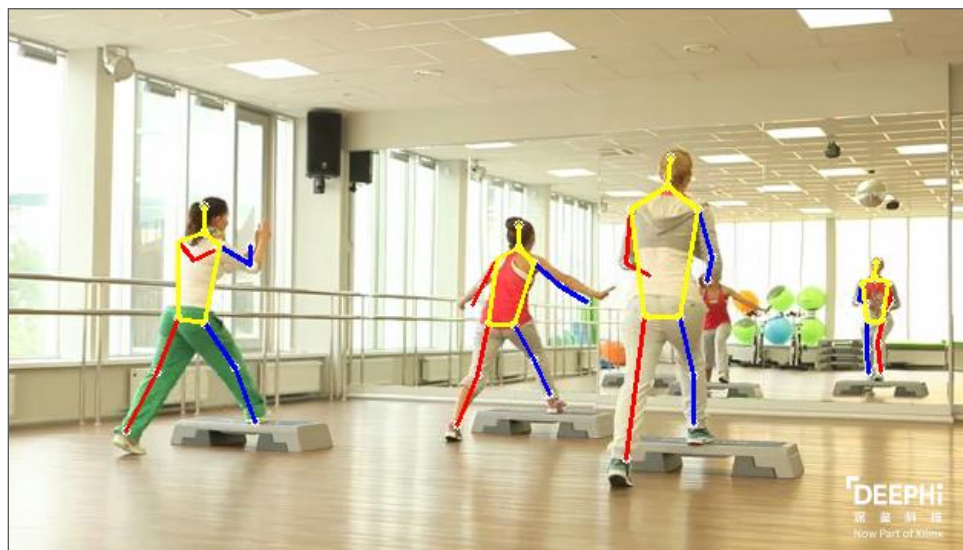


Figure 25: Screenshot of pose detection example

Video analytics

An object detection example is located under the `$dnndk_sample_base/samples/video_analysis` directory. It reads image frames from a video file and annotates detected vehicles and pedestrians in real-time. Run make and launch it with the command `./video_analysis video/structure.mp4` (where video/structure.mp4 is the input video file). A screenshot is shown in Figure 26.

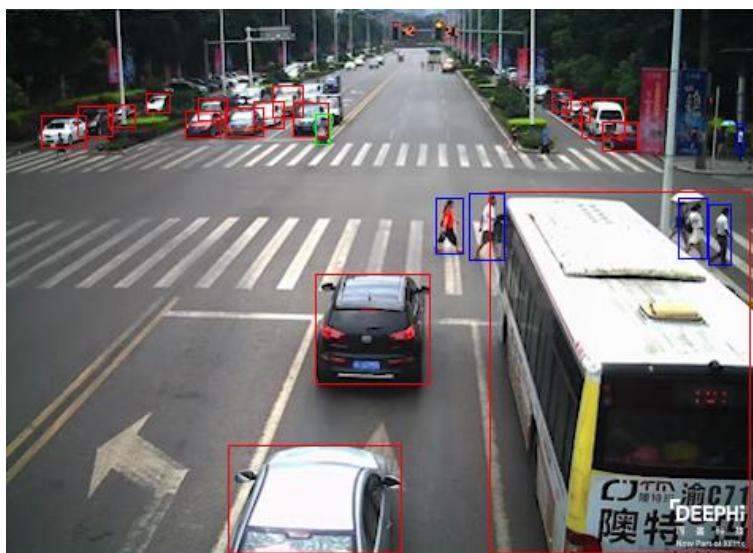


Figure 26: Screenshot of object detection example

ADAS detection

An example of object detection for ADAS (Advanced Driver Assistance Systems) application using YOLO-v3 network model is located under the “\$dnndk_sample_base/samples/adas_detection” directory. It reads image frames from a video file and annotates in real-time. Run make and launch it with the command “./adas_detection video/adas.avi” (where video/adas.mp4 is the input video file). A screenshot is shown in Figure 27.



Figure 27: Screenshot of ADAS detection example

Semantic segmentation

An example of semantic segmentation in the “\$dnndk_sample_base/samples/segmentation” directory. It reads image frames from a video file and annotates in real-time. Run make and launch it with the command “./segmentation video/traffic.mp4” (where video/traffic.mp4 is the input video file). A screenshot is shown in Figure 28.



Figure 28: Screenshot of semantic segmentation example

Support

Welcome to visit DeepPhi DNNDK community forum on Xilinx website

<https://forums.xilinx.com/t5/DeepPhi-DNNDK/bd-p/DeepPhi> for topics discussing, knowledge sharing, FAQ and requests for technical support.

Copyright

Some third-party source code are used in DNNDK toolchain. Please refer to the file `$deephi_dnndk_package/COPYRIGHT` for the related copyright declaration.

Version

Host package

The “host_x86” directory in the DNNDK release package contains the host side tools DECENT and DNNC. The version information is shown below. Contact DNNDK support team and provide the version information with command “decent --version” and “dnnc --version” if you encounter issues while using DECENT or DNNC.

DP-8020, DP-N1 and Ultra96

```
decent version 1.2.4
Build Label Dec 7 2018 02:47:58
(c) Copyright 2016 – 2018 Xilinx, Inc. All rights reserved.
```

```
dnnc version v2.03
DPU Target : v1.3.7
Build Label: Dec 11 2018 11:57:45
Copyright @2018 Xilinx Inc. All Rights Reserved.
```

ZCU102 and ZCU104

```
decent version 1.2.4
Build Label Dec 7 2018 02:47:58
(c) Copyright 2016 – 2018 Xilinx, Inc. All rights reserved.

dnnc version v2.03
DPU Target : v1.3.0
Build Label: Dec 11 2018 11:52:13
Copyright @2018 Xilinx Inc. All Rights Reserved.
```

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

Target package

The package for the DNNDK evaluation boards contains several components: DExplorer, DSight, DPU driver, and runtime N²Cube. The component versions for each evaluation board are shown below. Contact DNNDK support team and provide the version information with command “dexplorer -v” if you encounter issues while running DNNDK applications on evaluation boards.

DP-8020

```
DNNDK version 2.08 beta
Copyright @ 2016-2018 DeePhi Inc. All Rights Reserved.

DExplorer version 1.5
Build Label: Dec 11 2018 20:51:06

DSight version 1.4
Build Label: Dec 11 2018 20:51:07

N2Cube Core library version 2.2
Build Label: Dec 11 2018 20:51:29
```

DPU Driver version 2.0

Build Label: Dec 11 2018 20:51:02

DP-N1

DNNDK version 2.08 beta

Copyright @ 2016-2018 DeePhi Inc. All Rights Reserved.

DExplorer version 1.5

Build Label: Dec 11 2018 20:24:43

DSight version 1.4

Build Label: Dec 11 2018 20:24:44

N2Cube Core library version 2.2

Build Label: Dec 11 2018 20:25:07

DPU Driver version 2.0

Build Label: Dec 11 2018 20:24:39

Ultra96

DNNDK version 2.08 beta

Copyright @ 2016-2018 DeePhi Inc. All Rights Reserved.

DExplorer version 1.5

Build Label: Dec 11 2018 21:12:17

DSight version 1.4

Build Label: Dec 11 2018 21:12:18

N2Cube Core library version 2.2

Build Label: Dec 11 2018 21:12:42

DPU Driver version 2.0

Build Label: Dec 11 2018 21:12:13

ZCU102

DNNDK version 2.08 beta

Copyright @ 2016-2018 DeePhi Inc. All Rights Reserved.

DExplorer version 1.5

Build Label: Dec 11 2018 21:13:45

DSight version 1.4

Build Label: Dec 11 2018 21:13:46

N2Cube Core library version 2.2

Build Label: Dec 11 2018 21:14:07

DPU Driver version 2.0

Build Label: Dec 11 2018 21:13:42

ZCU104

DNNDK version 2.08 beta

Copyright @ 2016-2018 DeePhi Inc. All Rights Reserved.

DExplorer version 1.5

Build Label: Dec 11 2018 21:15:32

DSight version 1.4

Build Label: Dec 11 2018 21:15:33

N2Cube Core library version 2.2

Build Label: Dec 11 2018 21:15:54

DPU Driver version 2.0

Build Label: Dec 11 2018 21:15:28

Chapter 3: Upgrade and Porting

The key improvements and changes in each release of the DNNDK toolchain since the first version 1.07 published in Oct. 2017 are summarized in this chapter. A guide is provided to allow the users to port DPU applications from a previous DNNDK version to the latest version.

Since v2.08

Another two new evaluation boards DeePhi DP-N1 AI Box and Ultra96 are enabled since v2.08 release. Together with the existing three boards DP-8020, ZCU102 and ZCU104, there are total five evaluation boards available now. Meanwhile, DNNDK toolchains are under the continuous enhancements to improve DPU's performance, to make tools easy to use, to make DNNDK more suitable for production environment, etc.

Toolchain changes

DNNC

New option "--abi" is introduced since DNNC v2.03 for the purpose of DNNDK forward compatibility. Please refer to the section DNNC usage for detailed description.

Exception handling changes

A new exception handling mode for N²Cube runtime APIs is introduced. For the former releases, N²Cube 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 won't exit in case of errors. The APIs' callers need to take charge of the following exception handling process, such as logging the error message with API `dpuGetExceptionMessage()`, resource release, etc.

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

API changes

Four new APIs are introduced into library `libn2cube`. Please refer to Chapter 11: DNNDK Programming APIs for details.

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

- `dpuGetExceptionMessage()`

Example changes

Three new DNNDK examples are added to demonstrate DPU's 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 v2.07 release, the supported evaluation boards are DP-8020, ZCU102 and ZCU104.

Toolchain changes

DNNC

Some minor updates are added to DNNC to improve its easy-to-use.

1. The kernel graph description file "`<net_name>_kernel_graph.jpg`" in JPEG format will be generated by DNNC if the "dot" is already installed the host system; otherwise, the original gv format file with the same name will be generated.
2. Instead of simply printing input/output node name for DPUKernel in the kernel description, 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 N²Cube, the "`node_name`" and "`index`" act as unique identifiers.
3. A dump option has been added to DNNC for error probing (see 0 section for more details).

Since v2.06

Toolchain changes

DECENT

1. Support more operators
 - Dilation convolution
 - Deconvolution

- Flatten
 - Concat
 - Scale
 - Reorg
2. Support more layer fusing patterns
 - Fuse BatchNorm + Scale + Convolution into Convolution layer
 - Fuse standalone BatchNorm + Scale into Scale layer
 3. Support for TensorFlow framework (for DeePhi internal evaluation only at present)
 4. Other minor changes
 - Support auto test for cityscapes segmentation dataset
 - Support for CV_16U image input

DNNC

1. Support more operators:
 - Dilation convolution
 - Deconvolution
 - Flatten
 - Concat
 - Scale
 - Reorg
2. Implement more optimizing compilation techniques:
 - Add more flexible node fusion strategies
 - Perform adaptive optimizations for concat and flatten operators
3. Support for TensorFlow framework (for DeePhi internal evaluation only)
4. 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, etc.

API Changes

Multiple IO supported in this release, hence all the DNNDK API around input/output tensor such as `dpuGetInputXXX()/dpuGetOutputXXX()/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".

At the same time, another two new APIs `dpuGetInputTensorCnt()` and `dpuGetOutputTensorCnt()` are introduced to get the total number of input/output tensors for the specific node.

Example changes

VGG-16 has been removed in this release due to its large size. And several new examples have been added, including:

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

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 info 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.

1. DNNC compiler optimization for multiplexed memory access.
2. One-click compilation support for more neural networks including ResNet-50, VGG-16, Inception, SSD, YOLO, SqueezeNet and DenseBox.
3. 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 which has been added to this release.

API Changes

New API

`dpuSetInputImage2()` – set input image to DPU Task, bypassing the requirement to specify the mean values parameter. Pay attention to the difference between it with `dpuSetInputImage()`.

Changed API

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

Upgrading from previous versions

From v1.10 to v2.06

In this section, upgrading DNNDK 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 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 (e.g. “`dot -Tjpg -o xxx.jpg kernel_graph.gv`”).

3. The `kernel_graph.gv` will show 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 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_resnet50_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

DNNDK (Deep Neural Network Development Kit) is a full-stack deep learning SDK for the DPU (Deep-learning Processor Unit). 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

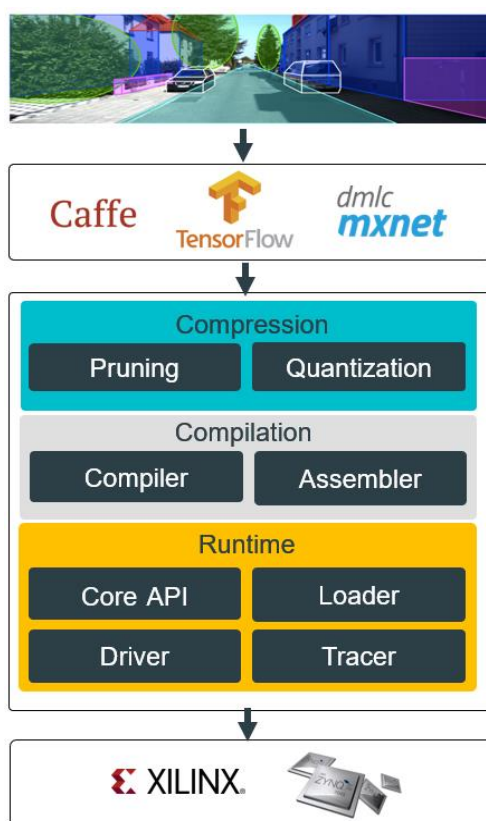


Figure 29: DNNDK Framework

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 (DPU)

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

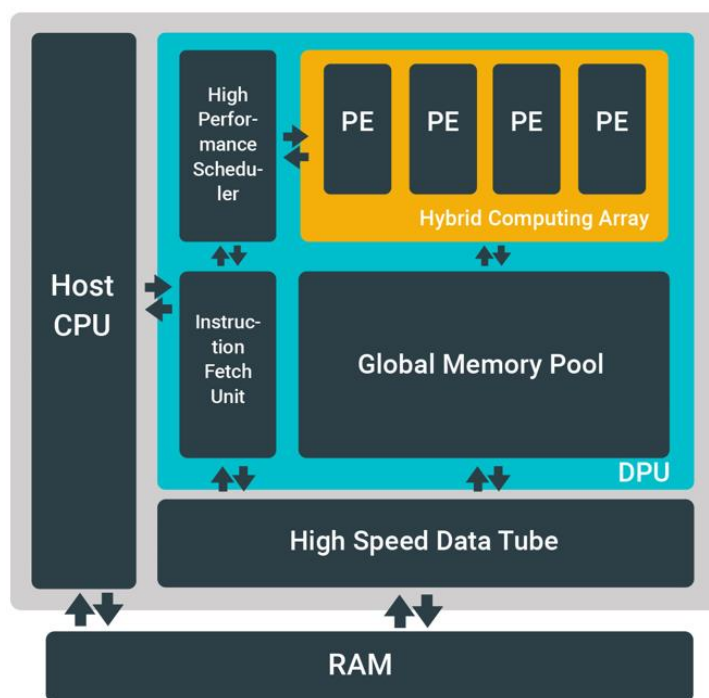


Figure 30: DPU architecture

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

DNNDK framework

As shown in Figure 31, DNNDK is composed of Deep Compression Tool (DECENT), Deep Neural Network Compiler (DNNC), Deep Neural Network Assembler (DNNAS), Neural Network Runtime (N²Cube), DPU Simulator and Profiler.

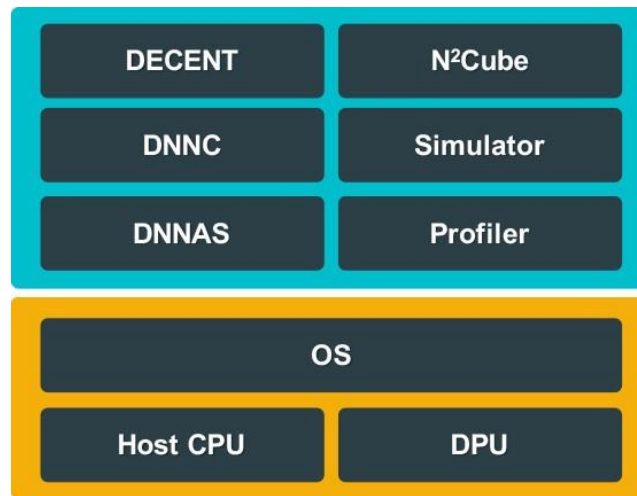


Figure 31: DNNDK Toolchain

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.

DECENT™ (Deep Compression Tool), 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 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

N²Cube (Cube of Neutral Networks) 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

DNNAS (Deep Neural Network Assembler) is responsible for assembling DPU instructions into ELF format binary code. It belongs to the sub-component of the DNNC code generation backend, and can't 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.

DECENT 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 (e.g. 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 5 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 floating-point models for Resnet-50 and Inception-v1 can be found from DNNDK package as Table 7 shows.

Table 7: Neural network models in DNNDK package

Model	Directory
ResNet-50	\$deephi_dnndk_package/host_x86/models/resnet50
Inception-v1	\$deephi_dnndk_package/host_x86/models/inception_v1

Note: This chapter only covers model quantization with DECENT since the pruning tool isn't included in this release.

Network compression

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 8: Input files for DECENT

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 “decen.sh” can be found from `$deephi_dnndk_package/host_x86/models/resnet50`, shown in Figure 32. 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 32: Screenshot of sample DECENT quantization script

Note: Before launching quantitation for ResNet-50, the calibration dataset used by DECENT should be prepared first. You can download 100 to 1000 images from ImageNet training dataset from <http://www.image-net.org/an> and then change the settings for “source” and “root_folder” of “image_data_param” in ResNet-50 prototxt accordingly.

The script may take several minutes to finish. Once quantization is done, two files “deploy.prototxt” and “deploy.caffemodel” will be generated under the “decent_output” directory, which could be fed to DNNC compiler for following compilation process.

Table 9: 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 compilation

Another script file named "dnnc.sh" can be found from `$deephi_dnndk_package/host_x86/models/resnet50`, shown in Figure 33. It invokes the DNNC tool to perform model compilation with the appropriate options.

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

#working directory
work_dir=$(pwd)

#path of fix&prototxt&caffemodel
model_dir=${work_dir}/decent_output
#output directory
output_dir=${work_dir}/resnet50

prototxt=${model_dir}/deploy.prototxt
caffemodel=${model_dir}/deploy.caffemodel

dnnc --prototxt=${prototxt} \
    --caffemodel=${caffemodel} \
    --output_dir=${output_dir} \
    --dpu=1152F \
    --cpu_arch=arm32 \
    --net_name=resnet50
```

Figure 33: Screenshot of sample DNNC compilation script

Running this script will compile the ResNet-50 model into two DPU kernels, which are ELF format files containing DPU instructions and parameters for ResNet-50. This will also show the information about layers unsupported by the DPU, as shown in Figure 34. The ResNet-50 network model is compiled and transformed into four different kernels:

- Kernel 0 : resnet50_0 (run on DPU)
- Kernel 1 : resnet50_1 (deploy on the CPU)
- Kernel 2 : resnet50_2 (run on DPU)
- Kernel 3 : resnet50_3 (deploy on the CPU)

```

Compiling network: resnet50
[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][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 'resnet50_kernel_graph.gv'.

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 34: Screenshot of DNNC compilation log

“resnet50_0” and “resnet50_2” are kernels running on the DPU. DNNC generates an ELF object file for each kernel under the “output_dir” directory with names “dpu_resnet50_0.elf” and “dpu_resnet50_2.elf”.

The other two kernels “resnet50_1” and “resnet50_3” are for “Average Pooling” and “Softmax” operations, which aren’t supported by DPU and need to be deployed and run on CPU.

Programming with DNNDK

To develop deep learning applications on the DPU, there are three types of work need 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

For the ZCU102 board, the ResNet-50 example is located under the “\$deephi_dnndk_package/ZCU102/samples/resnet50/” directory. The code for managing the DPU kernels and tasks are programmed in the function “main”.

```
int main(void) {
    /* DPU Kernels/Tasks for running ResNet-50 */
    DPUKernel* kernelConv;
    DPUKernel* kernelFC;
    DPUTask* taskConv;
    DPUTask* taskFC;

    /* Attach to DPU driver and prepare for running */
    dpuOpen();

    /* Create DPU Kernels for CONV & FC Nodes in ResNet-50 */
    kernelConv = dpuLoadKernel(KERNEL_CONV);
    kernelFC = dpuLoadKernel(KERNEL_FC);

    /* Create DPU Tasks for CONV & FC Nodes in ResNet-50 */
    taskConv = dpuCreateTask(kernelConv, 0);
    taskFC = dpuCreateTask(kernelFC, 0);

    /* Run CONV & FC Kernels for ResNet-50 */
    runResnet50(taskConv, taskFC);

    /* Destroy DPU Tasks & release resources */
    dpuDestroyTask(taskConv);
    dpuDestroyTask(taskFC);

    /* Destroy DPU Kernels & release resources */
    dpuDestroyKernel(kernelConv);
    dpuDestroyKernel(kernelFC);

    /* 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 kernels `reset50_0` and `reset50_2`.
- Call `dpuCreateTask()` to create tasks for each DPU kernel
- Call the CPU functions "CPUCalcAvgPool"(for average pooling) and "CPUCalcSoftmax" (for softmax)
- Call `dpuDestroyKernel()` and `dpuDestroyTask()` to destroy DPU kernels and tasks
- Call `dpuClose()` to close the DPU device

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

- Fetch an image using the OpenCV function `imread()` and set it as the input to the DPU kernel `resnet50_0` by calling the `dpuSetInputImage2()` API.
- Call `dpuRunTask()` to run the `taskConv` convolution operation in the ResNet-50 network model.
- Do average pooling on the output of the previous convolution operation and set the input of the `taskFC` operation as output.
- Call `dpuRunTask` to do the fully connected operation `taskFC` on the DPU.
- Do softmax on the CPU using the output of the fully connected operation as input.
- Output the top-5 classification category and the corresponding probability.

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

dpuRunTask(taskConv);

CPUCalcAvgPool(taskConv, taskFC);

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

Hybrid compilation

To generate the hybrid executable, change to the "*\$deeph_i_dnndk_package/samples/resnet50*" directory and run "make". This will compile the application source code to CPU binary code and then link it against the DPU kernels "dpu_resnet50_0.elf" and "dpu_resnet50_2.elf".

Running

In "*\$deeph_i_dnndk_package/samples/resnet50*", execute "*./resnet50*" to run this application and check its output.

DECENT overview

DECENT (Deep Compression Tool) includes two capabilities: Coarse-Grained Pruning and trained quantization. These reduce the number of required operations and quantize the weights. The whole working flow of DECENT is shown in Figure 35. In this release, only the quantization tool is included. Contact with DNNDK support team if pruning tool is necessary for your project evaluation.

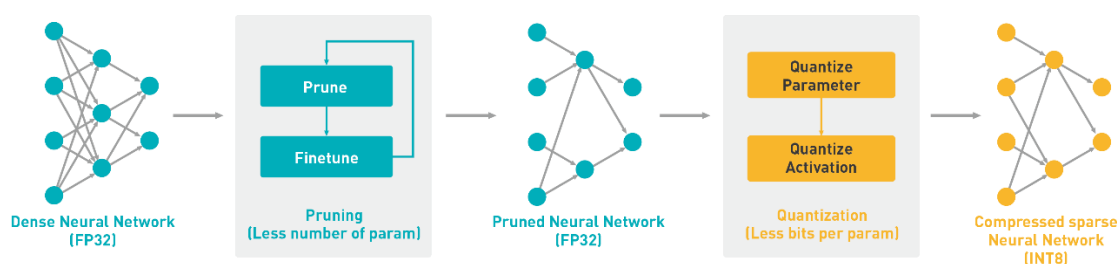


Figure 35: DECENT flow

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 a few seconds to several minutes, depending on the size of the neural network.

DECENT is based on Caffe 1.0 and a GPU is required to run it. In this release, only the Caffe format is supported.

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

DECETN usage

The options supported by DECENT are shown in Table 10.

Table 10: 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	Fix method, 0: OVER_FLOW; 1: DIFF_S.
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	YAML file which defines the layers to ignore during quantization, starting with 'ignore_layers:'

Working flow

Prepare neural network model

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

1. Caffe floating-point network model prototxt file
2. Pre-trained Caffe floating-point network model caffemodel file.
3. 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 Figure 36.


```
# 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 36: Screenshot of 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.

Running

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, the [options] stands for optional parameters. The 3 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 stands for OVER_FLOW and 1 stands for DIFF_S (default is 0, which has a shorter execution time)

Output

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

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

Chapter 6: Network compilation

DNNC overview

The architecture of the DNNC compiler is shown in Figure 37. The front-end parser is responsible for parsing the Caffe 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.

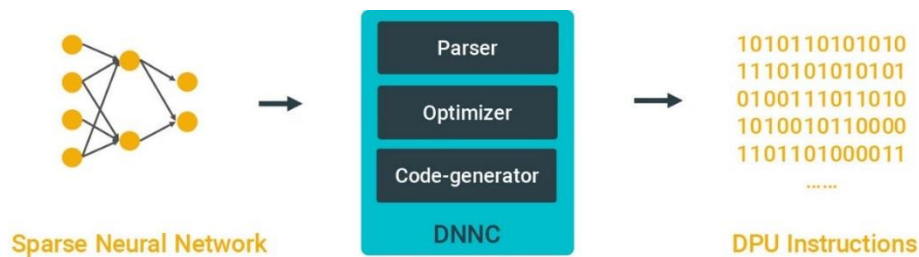


Figure 37: DNNC components

DNNC usage

DNNC requires options to control the compilation process for neural network models. These options are divided into two categories: the first group shows the required ones; and the next group shows optional ones.

Table 11: DNNC required option list

Parameters	Description
--prototxt	Path of Caffe prototxt file
--caffemodel	Path of caffemodel file
--output_dir	Path of output directory
--net_name	Name of neural network
--dpu	DPU target (supported list: 1024FA, 1152FA, 4096FA)
--cpu_arch	CPU target (supported list: arm32, arm64)

Table 12: DNNC optional option list

Parameters	Description
--help	Show all options of DNNC
--version	Show DNNC version
--save_kernel	Whether save kernel description in file or not
--abi	<p>Indicate the ABI version for DPU ELF generated by DNNC.</p> <ul style="list-style-type: none"> - 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. <p>Note: this option is available since DNNC v2.03.</p>
--mode	<p>Compilation mode of DPU kernel - debug or normal.</p> <ul style="list-style-type: none"> - debug: the layers of 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 debug mode DPU kernel. - normal: all layers of network model are packaged into one single DPU execution unit, and there isn't any interrupt involved during running. Normal mode DPU kernel delivers better performance and should be used during production releaser phase.
--dump	<p>Dump different type information, use commas as delimiter when multiple types are given:</p> <ul style="list-style-type: none"> - graph: original graph and transformed graph in gv 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. <p>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.</p>

Compiling ResNet50

When compiling a neural network, the required options should be specified to DNNC compiler. For convenience, the script files to compile ResNet50 and Inceptoin-v1 are provided inside DNNDK release package. They are helpful for uses to become familiar with the usages 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”. Figure 38 shows a screenshot of the DNNC output when compiling the Inception-v1 network.

```
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 38: Screenshot of sample DNNC output

Due to the limited number of operations supported by the DPU (see Table 13), DNNC will automatically partition the target neural network into different kernels when there are operations that are not supported by DPU. The user is responsible for the data transfer and communication between different kernels, using APIs provided by N²Cube 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 file named "kernel_graph" is saved in standard 'DOT (graph description language)' format with file extension 'gv' in the current DNNC working directory. By using the following command, the gv format file can be converted to a JPEG file:

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

For example, the kernel graph in JPEG format for Inception-v1 is shown in. 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 N²Cube to get the input or output tensor address.

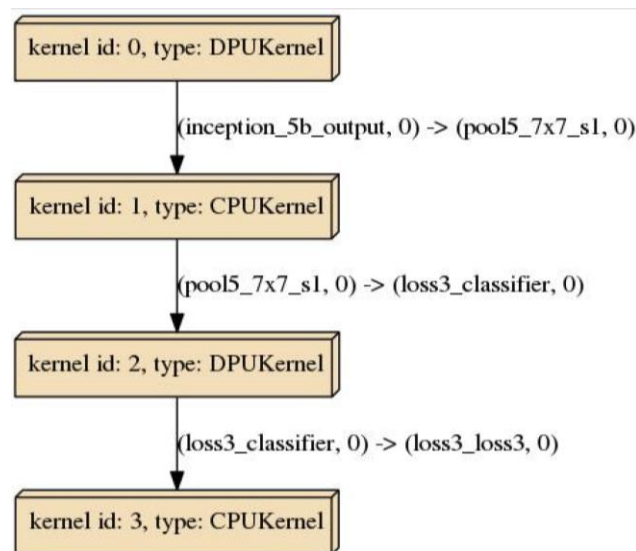


Figure 39: DPU Kernel graph for Inception-v1

The second one 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 will have 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 N²Cube to identify different kernels correctly.
- **Type:** the kernel type. Three types of kernel are supported by DNNC, see Table 14 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 N²Cube.
- **Output nodes:** All output nodes of the current kernel. The address and size of output nodes can be extracted using APIs provided by N²Cube.

Table 13: 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	Only max-pooling is supported, and kernel size must be 2x2 or 3x3
Concat	No limitations
Element-wise	No limitations
InnerProduct	No limitations

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

Table 14: 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 DNNDK 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 “DPU task” are two core concepts for DNNDK programming.

DPU Kernel

After being compiled by DNNC compiler, the neural network model is transformed into an equivalent DPU assembly file, which is then assembled into one ELF object file by DNNAS. DPU ELF object file is regarded as DPU kernel, which becomes one execution unit from the perspective of runtime N²Cube after invoking the API `dpuLoadKernel()`. N²Cube 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 may 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 may be used to process several tasks in parallel to improve efficiency and system throughput.

DPU Node

A DPU node is considered as 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 which doesn't have any precursor in the DPU kernel's topology; it is usually the first node in a kernel. Sometimes there may be multiple boundary input nodes in a kernel.
- A boundary output node is a node which doesn't have any successor node in the DPU kernel's topology.
- All other nodes which aren't both boundary input nodes and boundary output nodes are considered as internal nodes.

After compilation, DNNC will give information about the kernel and its boundary input/output Nodes. Figure 40 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 40: Screenshot of sample DNNC compilation log

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

```

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

```

Similarly, when using `dpuGetOutputTensor*/dpuSetOutputTensor*`, an error will be generated when a “nodeName” which 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 which is used to store information while running. Tensor properties (e.g. height, width, channel etc.) may 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, please refer to Chapter 12.

It is common to exchange data between a CPU and the DPU when programming for the DPU. For example, data preprocessed by a CPU may be sent to the DPU for acceleration; and the output produced by the DPU may 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()`

Chapter 8: Hybrid compilation

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 Figure 41.

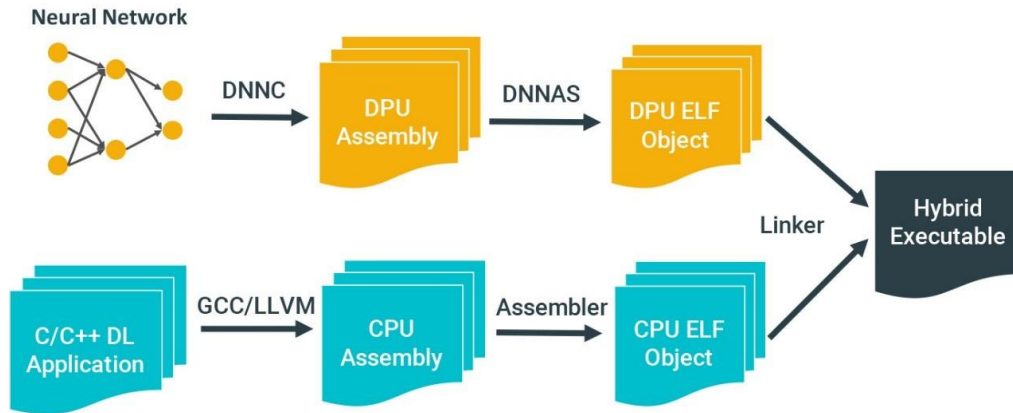


Figure 41: Hybrid compilation process

Code that will run on a CPU are 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 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 will occur.

As described in previous chapters, 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 Figure 42. It is composed of DPU Loader, DPU profiler, DPU runtime library and DPU driver.

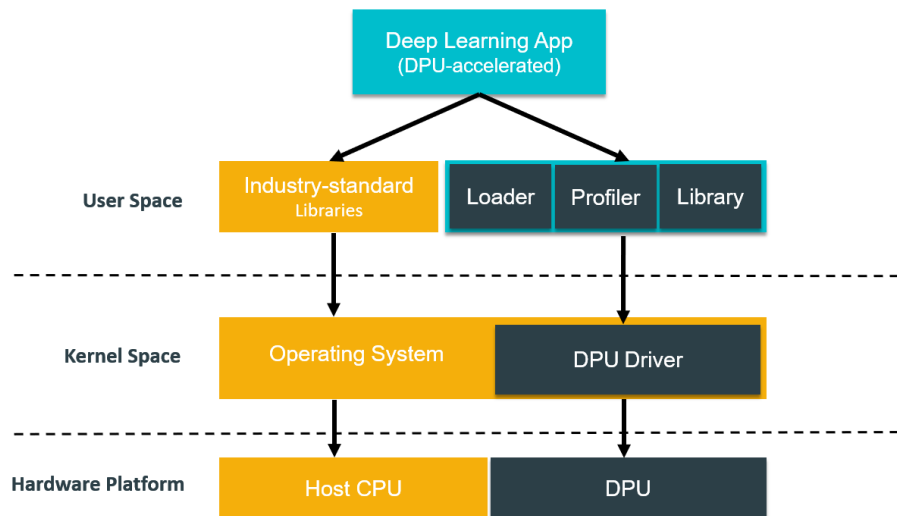


Figure 42: DPU Runtime

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. Figure 43 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 43: DExplorer usage options

Check DNNDK version

Running “dexplorer -v” will display version information for each component in DNNDK, including N²cube, DPU driver, DExplorer and DSight.

Check DPU status

DExplorer provides DPU status information, including running mode of N²cube, DPU timeout threshold, DPU debugging level, DPU core status, DPU register information, DPU memory resource and utilization. Figure 44 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 44: DPU status

Configure DPU running mode

DNNDK runtime N²cube 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, N²cube 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. Figure 45 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%	709
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%	906
Total Nodes In Avg:							
	All	7711.9	44.4	64.62	119.3	51.9%	687

Figure 45: N²Cube profile mode

Debug mode

In this mode, the DPU will dump 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 DNNDK 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 now offered by DeePhi. Each DPU architecture can implement a different version of the DPU instruction set (which is named to a DPU target version by DeePhi) 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 may be used to check the DPU signature. Figure 46 shows a screenshot of a sample run of “dexplorer -w”.

```
root@xlnx:~# dexplorer -w
[DPU IP Spec]
IP Timestamp      : 2018-09-07 20:00:00
DPU Core Count    : 1
IRQ Base 0        : 121
IRQ Base 1        : 136

[DPU Core List]
DPU Core          : #0
DPU Enabled       : Yes
DPU Arch          : B1152F
DPU Target        : v1.3.1
DPU Frequency     : 500 MHz
DPU IRQ           : 138
DPU Features      : Avg-Pooling, Leaky ReLU
```

Figure 46: Screenshot of sample DPU signature

DSight

DSight is the DNNDK performance profiling tool, and it is a visual performance analysis tool for neural network model profiling. Figure 47 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 47: DSight help info

By processing the log file produced by the N²cube 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 N²Cube to profile mode using the command “dexplorer -m profile”.

- Run the deep learning application. When finished, a profile file with the name "dpu_trace_[PID].prof" will be generated for further checking and analysis (PID is the process ID of the deep learning application).
- 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" will be generated.
- Open the generated html file with web browser.



Figure 48: DSight profiling charts

Chapter 11: DNNDK Programming APIs

DNNDK provides a lightweight set of C/C++ programming APIs for deep learning application developers. It consists of two dynamic libraries, DPU runtime N²Cube library "libn2cube" and DPU utility library "libdputils". The exported APIs for them are individually contained in header file "n2cube.h" and "dputils.h", which will be described in detail in this chapter.

Note: For simplification, the users only need to include header file "dnndk.h" into DNNDK applications. It includes both "n2cube.h" and "dputils.h" by default.

Note: The subsequent sections give detailed description to each API. Item "AVAILABILITY" indicates since which DNNDK version the corresponding API becomes available.

Library libn2cube

Overview

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

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

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

	<p>dpuGetOutputTensorInCHWInt8() - Get DPU Task's output Tensor and store them under Caffe order (channel/height/width) in INT8 format</p> <p>dpuGetOutputTensorInCHWFP32() - Get DPU Task's output Tensor and store them under Caffe order (channel/height/width) in FP32 format</p> <p>dpuGetOutputTensorInHWCInt8() - Get DPU Task's output Tensor and store them under DPU order (channel/height/width) in INT8 format</p> <p>dpuGetOutputTensorInHWCFP32() - Get DPU Task's output Tensor and store them under DPU order (channel/height/width) in FP32 format</p>
INCLUDE FILE	n2cube.h

APIs

The prototype and parameter for each API of library libn2cube are depicted 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” will be 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” will be reported if any error takes place.
SEE ALSO	dpuOpen()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuLoadKernel()

NAME	dpuLoadKernel()	
SYNOPSIS	<pre> DPUKernel *dpuLoadKernel (const char *netName) </pre>	
ARGUMENTS	netName	<p>The pointer to neural network name. Please use the names produced by DNNC after the compilation of neural network.</p> <p>For each DL application, perhaps there are many DPU Kernels existing in its hybrid CPU+DPU binary executable. For each</p>

		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	<pre>int dpuDestroyKernel (DPUKernel *kernel)</pre>	
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	<pre>int dpuCreateTask (DPUKernel *kernel, Int mode);</pre>	
ARGUMENTS	kernel	The pointer to DPU Kernel.
	mode	<p>The running mode of DPU Task. There are 3 available modes:</p> <p>MODE_NORMAL: default mode identical to the mode value "0".</p> <p>MODE_PROF: output profiling information layer by layer while running of DPU Task, which is useful for performance analysis.</p> <p>MODE_DUMP: dump the raw data for DPU Task's CODE/BIAS/WEIGHT/INPUT/OUTPUT layer by layer.</p> <p>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)):</p> <p>For CODE: netName_layerName_code.txt</p> <p>For WEIGHT: netName_layerName_w.txt</p> <p>For BIAS: netName_layerName_b.txt</p>

		<p>For INPUT: netName_layerName_i.txt</p> <p>For OUTPUT: netName_layerName_o.txt</p> <p>NOTE: profiling and dump functionality is available only for DPU Kernel generated by DNNC in debug mode.</p>
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	<pre>int dpuDestroyTask (DPUTask *task)</pre>	
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	<pre>int dpuRunTask (DPUTask *task);</pre>	
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	<pre>int dpuEnableTaskProfile (DPUTask *task</pre>	

);	
ARGUMENTS	task	The pointer to DPU Task.
DESCRIPTION	Set DPU Task in profiling mode. Please be noted 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	<pre>int dpuEnableTaskDump (DPUTask *task);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
DESCRIPTION	Set DPU Task in dump mode. Please be noted 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	<pre>int dpuGetTaskProfile (DPUTask *task);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
DESCRIPTION	Get DPU Task's execution time (us) after its running.	
RETURNS	0 on success, or negative value in case of any failure.	
SEE ALSO	dpuGetNodeProfile()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetNodeProfile()

NAME	dpuGetNodeProfile()	
SYNOPSIS	<pre>int dpuGetNodeProfile (</pre>	

	<pre> DPUTask *task, const char*nodeName); </pre>	
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	0 on success, or negative value in case of any failure. Please be noted 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	<pre> Int dpuGetInputTensorCnt (DPUTask *task, const char*nodeName); </pre>	
ARGUMENTS	task	The pointer to DPU Task.

	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre> DPUTensor*dpuGetInputTensor (DPUTask *task, const char*nodeName, int idx = 0); </pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre> DPUTensor*dpuGetInputTensor (DPUTask *task, const char*nodeName); </pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int8_t* dpuGetInputTensorAddress (DPUTask *task, const char*nodeName, int idx = 0);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int8_t* dpuGetInputTensorAddress (DPUTask *task, const char*nodeName);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetInputTensorSize (DPUTask *task, const char*nodeName, int idx = 0);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetInputTensorSize (DPUTask *task, const char*nodeName);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>float dpuGetInputTensorScale (DPUTask *task, const char*nodeName, int idx = 0);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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. Please be noted 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 will be 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	<pre>int dpuGetInputTensorHeight (DPUTask *task, const char*nodeName, int idx = 0);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetInputTensorHeight (DPUTask *task, const char*nodeName);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetInputTensorWidth (DPUTask *task, const char*nodeName, int idx = 0);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetInputTensorWidth (DPUTask *task, const char*nodeName);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetInputTensorChannel (DPUTask *task, const char*nodeName, int idx = 0);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetInputTensorChannel (DPUTask *task, const char*nodeName);</pre>	
ARGUMENTS	task	The pointer to DPU Task.

	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre> Int dpuGetOutputTensorCnt (DPUTask *task, const char*nodeName); </pre>

ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre> DPUTensor*dpuGetOutputTensor (DPUTask *task, const char*nodeName, int idx = 0); </pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre> DPUTensor*dpuGetOutputTensor (DPUTask *task, const char*nodeName); </pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int8_t* dpuGetOutputTensorAddress (DPUTask *task, const char*nodeName, int idx = 0);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int8_t* dpuGetOutputTensorAddress (DPUTask *task, const char*nodeName);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetOutputTensorSize (DPUTask *task, const char*nodeName, int idx = 0);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetOutputTensorSize (DPUTask *task, const char*nodeName);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>float dpuGetOutputTensorScale (DPUTask *task, const char*nodeName, int idx = 0);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	dpuGetOutputTensorScale()	
INCLUDE FILE	n2cube.h	

AVAILABILITY	v2.06	
NAME	dpuGetOutputTensorScale()	
SYNOPSIS	<pre>float dpuGetOutputTensorScale (DPUTask *task, const char*nodeName);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node’s name. Please be noted 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 will be 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	dpuGetOutputTensorScale()	
INCLUDE FILE	n2cube.h	
AVAILABILITY	v1.07	

dpuGetOutputTensorHeight()

NAME	dpuGetOutputTensorHeight()	
SYNOPSIS	<pre>int dpuGetOutputTensorHeight (DPUTask *task, const char*nodeName, int idx = 0);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetOutputTensorHeight (DPUTask *task, const char*nodeName);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetOutputTensorWidth (DPUTask *task, const char*nodeName, int idx = 0);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetOutputTensorWidth (DPUTask *task, const char*nodeName);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetOutputTensorChannel (DPUTask *task, const char*nodeName, int idx = 0);</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetOutputTensorChannel (DPUTask *task, const char*nodeName);</pre>	
ARGUMENTS	task	The pointer to DPU Task.

	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuGetTensorSize (DPUTensor* tensor);</pre>	
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	<pre>float dpuGetTensorScale (DPUTensor* tensor);</pre>	
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	<pre>float dpuGetTensorHeight (DPUTensor* tensor);</pre>	
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	<pre>float dpuGetTensorWidth (DPUTensor* tensor);</pre>	
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	<pre>float dpuGetTensorChannel (DPUTensor* tensor);</pre>	
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	<pre>int dpuSetInputTensorInCHWInt8 (DPUTask *task, const char*nodeName, int8_t *data, int size)</pre>	
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	

dpuSetIntputTensorInCHWFP32()

NAME	dpuSetIntputTensorInCHWFP32()	
SYNOPSIS	<pre>int dpuSetIntputTensorInCHWFP32 (DPUTask *task, const char*nodeName, float *data, int size)</pre>	
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	dpuSetIntputTensorInCHWInt8() dpuSetIntputTensorInHWCInt8() dpuSetIntputTensorInHWCfp32()	
INCLUDE FILE	n2cube.h	

AVAILABILITY	v1.07
--------------	-------

dpuSetInputTensorInHWCInt8()

NAME	dpuSetInputTensorInHWCInt8()	
SYNOPSIS	<pre>int dpuSetInputTensorInHWCInt8 (DPUTask *task, const char*nodeName, int8_t *data, int size)</pre>	
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

dpuSetIntputTensorInHWCFP32()

NAME	dpuSetIntputTensorInHWCFP32()	
SYNOPSIS	<pre>int dpuSetIntputTensorInHWCFP32 (DPUTask *task, const char*nodeName, float *data, int size)</pre>	
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	dpuSetIntputTensorInCHWInt8() dpuSetIntputTensorInCHWFP32()	

	dpuSetInputTensorInHWCInt8()
INCLUDE FILE	n2cube.h
AVAILABILITY	v1.07

dpuGetOutputTensorInCHWInt8()

NAME	dpuGetOutputTensorInCHWInt8()	
SYNOPSIS	<pre>int dpuGetOutputTensorInCHWInt8 (DPUTask *task, const char*nodeName, int8_t *data, int size)</pre>	
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	<p>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.</p>	
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	<pre>int dpuGetOutputTensorInCHWFP32 (DPUTask *task, const char*nodeName, float *data, int size)</pre>	
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	<pre>int dpuGetOutputTensorInHWCInt8 (DPUTask *task, const char*nodeName, int8_t *data, int size)</pre>	
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 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	<pre>int dpuGetOutputTensorInHWCFP32 (DPUTask *task, const char*nodeName, float *data, int size)</pre>

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	<pre>int dpuRunSoftmax (int8_t *input, float *output, int numClasses,</pre>

	<pre> int batchSize, float scale) </pre>	
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	<pre>int dpuSetExceptionMode (int mode)</pre>	
ARGUMENTS	mode	<p>The exception handling mode for runtime N²Cube to be specified. Available values include:</p> <ul style="list-style-type: none"> - <i>N2CUBE_EXCEPTION_MODE_PRINT_AND_EXIT</i> - <i>N2CUBE_EXCEPTION_MODE_RET_ERR_CODE</i>
DESCRIPTION	<p>Set the exception handling mode for DNNDK runtime N²Cube. It will affect all the APIs included in the libn2cube library.</p> <p>If N2CUBE_EXCEPTION_MODE_PRINT_AND_EXIT is specified, the invoked N²Cube APIs will output the error message and terminate the running of DNNDK application when any error occurs. It is the default mode for N²Cube APIs.</p> <p>If N2CUBE_EXCEPTION_MODE_RET_ERR_CODE is specified, the invoked N²Cube 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.</p>	
RETURNS	0 on success, or negative value in case of failure.	
SEE ALSO	<p>dpuGetExceptionMode()</p> <p>dpuGetExceptionMessage()</p>	

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 handling mode for N ² Cube APIs. Available values include: <ul style="list-style-type: none"> - <i>N2CUBE_EXCEPTION_MODE_PRINT_AND_EXIT</i> - <i>N2CUBE_EXCEPTION_MODE_RET_ERR_CODE</i>
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 DPU utility library. It wraps up various highly optimized C/C++ APIs to facilitate DL applications development on DPU platform. The exported APIs is 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
INCLUDE FILE	dputils.h

APIs

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

dpuSetInputImage()

NAME	dpuSetInputImage ()	
SYNOPSIS	<pre>int dpuSetInputImage (DPUTask *task, const char *nodeName, const cv::Mat &image, float *mean)</pre>	
ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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 isn't supported so far.
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	<pre>int dpuSetInputImage (DPUTask *task, const char *nodeName, const cv::Mat &image)</pre>	
ARGUMENTS	task	The pointer to DPU Task.

	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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	<pre>int dpuSetInputImageWithScale (DPUTask *task, const char *nodeName, const cv::Mat &image, float *mean, float scale)</pre>

ARGUMENTS	task	The pointer to DPU Task.
	nodeName	The pointer to DPU Node's name. Please be noted 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 will be 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.

© Copyright 2012-2018 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.